# MESH2FAUST: a Modal Physical Model Generator for the Faust Programming Language – Application to Bell Modeling

**Romain Michon**[1], **Sara R. Martin**[2], **and Julius O. Smith**[1]
[1] Center for Computer Research in Music Acoustics (CCRMA), Stanford University (USA)
[1] Acoustics Research Center, Norwegian University of Science and Technology, Trondheim (Norway)
{rmichon,saram,jos}@ccrma.stanford.edu

## ABSTRACT

*This paper introduces* MESH2FAUST, *a modal physical model generator for the* FAUST *programming language as well as an open-source framework to design 3D objects and turn them into physical models for sound synthesis.*
MESH2FAUST *can convert any volumetric mesh of a 3D object into a* FAUST *modal physical model by extracting modal information from the result of a finite element analysis. A wide range of parameters can be configured to specify the material properties of the model, the behavior of the generated model, etc. This system is evaluated by applying it to bell synthesis.*

## 1. INTRODUCTION

The Finite Element Method can be used to compute mode parameters to synthesize the sound of a wide range of elements [1] using modal synthesis [2, 3]. However, there doesn't yet seem to exist a complete open-source solution to carry out this type of operation, allowing to simply draw a 3D object and turn it into a physical model for sound synthesis.

In this paper, we introduce MESH2FAUST, an open-source modal physical model generator for the FAUST programming language. MESH2FAUST takes a volumetric mesh of a 3D object as its main argument, carries out a finite element analysis, and generates the corresponding FAUST modal physical model. A wide range of parameters can be configured to fine-tune the analysis as well as the behavior of the generated object.

FAUST [4] is a functional programming language for real-time digital signal processing (DSP). It has been used extensively to implement waveguide and modal physical models of musical instruments [5]. The implementation of such models in FAUST is eased by the wide range of functions available in the FAUST DSP libraries [6] and by the block-diagram/signal-oriented syntax of the language.

MESH2FAUST is being developed as part of the FAUST *Physical Modeling Library and Tool Kit* (FPML), [1] allowing to implement a wide range of musical instrument physical models using various techniques. Thus, models generated by MESH2FAUST can be easily integrated to larger models implemented with FPML (e.g., a waveguide bowed string connected to a violin body generated by MESH2FAUST, etc.).

First, we present a brief review of the theory behind finite-element and modal synthesis. Next, we describe the implementation of MESH2FAUST and present a complete open-source framework to model 3D objects and turn them into physical models of musical instruments. Finally, we evaluate our system by applying it to bell modeling and synthesis, and we propose future directions for this work.

[1] https://ccrma.stanford.edu/~rmichon/pmFaust/ – All URLs were verified on July 17, 2017.

## 2. THEORY: FEM

The Finite Element Method (FEM) is a frequently used technique for modeling the dynamic deformation of an object and synthesizing the sound emitted by the object after an excitation. The method consists of meshing the object in small elements defined by several nodes (depending on the desired element type), and then solving the equations of motion for each of the nodes.
The linear deformation equation with no damping can be written as follows:

$$\mathbf{M}\,\ddot{\mathbf{x}}(\mathbf{t}) + \mathbf{K}\,\mathbf{x}(\mathbf{t}) = \mathbf{f}(\mathbf{t}) \tag{1}$$

where $\mathbf{x}(\mathbf{t}) \in \mathbb{R}^{3n}$ corresponds to the vector of displacements at all the nodes, and $\mathbf{M}$ and $\mathbf{K}$ represent respectively the mass and stiffness matrices determined by the object properties.
A first attempt to solve Eq.(1) is to assume that the solutions of the corresponding homogeneous equation $(\mathbf{f}(\mathbf{t}) = \mathbf{0})$ are of the form $u_i(t) = \mathbf{U}_i e^{j\omega_i t}$ where $\mathbf{U}_i \in \mathbb{R}^{3n}$ and $\omega_i \in \mathbb{R}$. The substitution of those potential solutions into the homogeneous equation of Eq.(1) defines the commonly called generalized eigenvalue problem:

$$\mathbf{KU} = \mathbf{\Lambda MU} \tag{2}$$

where $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues $\lambda_i = \omega_i^2$ of Eq.(1), and $\mathbf{U}$ is the modal matrix containing the eigenvectors $\mathbf{U}_i$ of Eq.(1). By solving this problem, both eigenvalues and eigenvectors of the system will be obtained.
Now, the system in Eq.(1) can be decoupled by using the transformation $\mathbf{x} = \mathbf{Uq}$ and Eq.(1) can be rewritten as

$$\ddot{\mathbf{q}} + \mathbf{\Lambda q} = \mathbf{U}^{\mathbf{T}}\mathbf{f}. \tag{3}$$

Thus, the solutions of the decoupled homogeneous modal form $\ddot{\mathbf{q}} + \mathbf{\Lambda q} = \mathbf{0}$ can be implemented using a parallel bank of modes of the form

$$q_i = a_i \sin(2\pi f_i + \theta_i), \tag{4}$$

where $a_i$ is the excited amplitude of the $i$th mode, $f_i$ is its frequency, and $\theta_i$ its initial phase. The excitation force $\mathbf{f}(\mathbf{t})$ is taken to be an impulse at time $t = 0$, and our simulation will start at time 0. To maximize the initial attack without creating an amplitude discontinuity at time 0, we choose our initial phases as $\theta_i = 0$. The excited mode amplitudes $a_i$ depend on the location of the object excitation, and the frequency of the $i$th mode depends on the object geometry and material properties according to

$$f_i = \frac{1}{2\pi}\sqrt{\lambda_i}\,, \tag{5}$$

where $\lambda_i$ denotes the $i$th eigenvalue obtained from solving Eq.(2).

Since the damping matrix was omitted in the above formulation, the modes in Eq.(4) are missing an important factor for sound synthesis which is the exponential decay. In this paper, exponential decays have not been estimated by FEM, as will be explained in the following section.

## 3. IMPLEMENTATION AND USE

### 3.1 Faust Modal Physical Model

Modal synthesis [2] consists of implementing each mode of a linear system as an exponentially decaying sine wave. Each mode can then be configured with its frequency, gain, and resonance duration (T60). Sine waves with an exponential decay are typically implemented using a sine wave oscillator with an exponential envelope or with a resonant bandpass filter [7]. The second option offers more flexibility since any signal can be fed into the model to excite it. This feature is important to be able to create modules compatible with FPML, which is why modal physical models generated by MESH2FAUST use this approach.

Our mode filters are implemented as a biquad section having transfer function

$$H(z) = g\frac{1 - z^{-2}}{1 + \alpha_1 z^{-1} + \alpha_2 z^{-2}} \tag{6}$$

with

$$\begin{aligned}
\alpha_1 &= -2\tau\cos\omega \\
\alpha_2 &= \tau^2 \\
\omega &= \frac{2\pi f}{f_s} \\
\tau &= 0.001^{\frac{1}{t_{60}}}
\end{aligned}$$

having the following parameters:

- $g$: the mode gain
- $f$: the mode frequency
- $t_{60}$: the mode T60

The constrained form of the transfer-function numerator $1-z^{-2} = (1+z^{-1})(1-z^{-1})$ enforces a zero of transmission at both dc and half the sampling rate, thereby giving a bandpass characteristic appropriate for a resonant mode. The denominator parameters similarly enforce a complex-conjugate pole pair with angles $\pm\omega$ and radius $\tau$.

The corresponding FAUST function (`modeFilter`) is used to implement our FAUST "modal model" which takes the position of excitation and the excitation signal as its two arguments:

```
modalModel(exPos) = _ <: par(i,nModes,
    modeFilter(modesFreqs(i),modesT60s(i),
    modesGains(exPos,i))) :> /(nModes)
```

The FAUST-generated [2] block diagram associated with this function can be seen in Figure 1. `modesFreqs(i)`, `modesT60s(i)`, and `modesGains(exPos,i)` are arrays that are formatted by MESH2FAUST (see §3.2). This model is compatible with all the decoupled excitation functions of the FAUST physical modeling library (e.g., hammer, pluck, impulse, etc.).

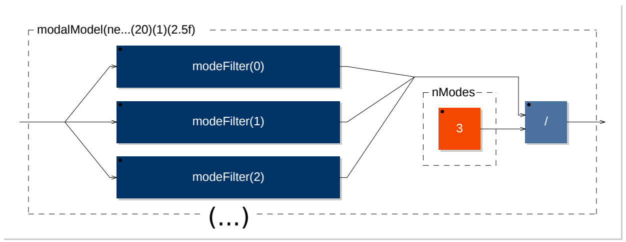

**Figure 1**. Block diagram of a FAUST modal model implementing three modes.

### 3.2 MESH2FAUST

MESH2FAUST is implemented in C++ and works as a UNIX command line application taking a volumetric mesh as its main argument and outputting a FAUST modal physical model. Various parameters can be configured using a wide range flags that are presented in this section. [3]

MESH2FAUST relies on the Vega FEM Library [4] [8] to carry out the finite element analysis needed to compute mode parameters. The provided volumetric mesh must be saved as an "object file" (`.obj`) and its dimensions should be in meters. The volumetric mesh must first be converted into a 3D tetrahedral mesh (see Figure 2). This is easily done by Vega which implements its own tetrahedral mesher [9]. Material properties (Young Modulus in $N/m^2$, Poisson's Ratio, and Density in $kg/m^3$) are applied to the model during this step and can be configured by the user using the `--material` flag.

Next, the corresponding mass and stiffness matrices are generated and fed to the Vega eigen solver. The number of modes to be computed during this step can be configured by the user using the `--nfemmodes` flag and must be smaller than the number of vertices in the volumetric mesh. The result of this operation is a list of eigenvalues

[2] This diagram was generated using the `faust2svg` tool.
[3] A complete list of the MESH2FAUST options is available in its online documentation: `https://github.com/rmichon/pmFaust`.
[4] `http://run.usc.edu/vega/`

and eigenvectors that are ordered linearly starting from the lowest mode.

As mentioned in Section 2, the modes frequencies can be easily calculated from eigenvalues (Eq.(5)), and the mode gains can be computed from the matrix of eigenvectors and the excitation force.

Before the mode gains and frequencies are integrated to the FAUST physical model, they are selected based on a series of user-defined parameters:

- `--nsynthmodes`: number of modes to synthesize
- `--minmode`: lowest mode frequency
- `--maxmode`: highest mode frequency
- `--cb`: mode selection by critical bands
- `--expos`: list of "excitable" vertices
- `--lmexpos`: number of excitation position

`--minmode` and `--maxmode` allow to define the frequency range of the modes to synthesize. If the number of modes within this range is smaller than `--nsynthmodes`, this parameter will be adapted accordingly. Note that `--nsynthmodes` can be different than `--nfemmodes` since some modes might be discarded at the bottom of the spectrum depending on the value of `--minmode`.

If the number of modes in the range defined by `--minmode` and `--maxmode` is greater than `--nsynthmodes`, synthesized modes will be selected by frequency, starting from the lowest mode. `--cb` allows to change this behavior by selecting modes by critical bands. In this case, the frequency range defined by `--minmode` and `--maxmode` will be split into `--nsynthmodes` critical bands and the loudest mode for each of them will be selected. This feature is very useful if the model has lots of modes.

By default, the number of excitation positions in the generated model is the same as the number of vertices in the provided volumetric mesh. If this mesh has a high density, the amount of data to integrate to the model might become a problem. For example, for a mesh with 3E4 vertices and 200 modes to synthesize, the modes gains matrix will have a size of 3E4x200 which corresponds to 6E6 floating point values to be hard-coded in the FAUST physical model source code! Thus, it might be helpful to optimize the model by limiting its number of excitation positions by using `--lmexpos`. In that case, positions are "randomly" selected, however, specific vertices can be selected using `--expos`. Vertex IDs can be easily retrieved using a mesh visualizer such as *meshlab* (see §4).

After this, the resulting FAUST modal physical model (see §3.1) is generated and placed in a FAUST library file (`.lib`).

Currently, MESH2FAUST doesn't compute the damping matrix of the system. Thus, while mode T60s cannot be estimated, they can be optionally empirically computed as a function of the frequency and the gain of the modes relatively to the fundamental. This solution is temporary and we hope to add damping matrix support to our system in the future (see §6).

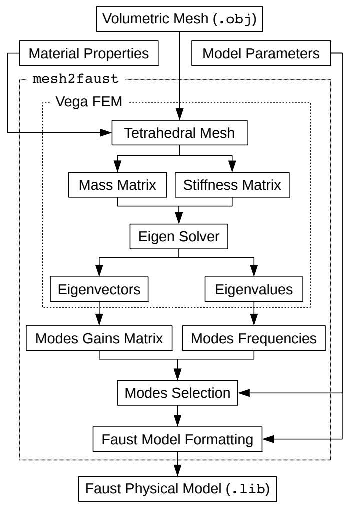

**Figure 2**. Overview of the MESH2FAUST implementation.

## 4. COMPLETE OPEN SOURCE SOLUTION TO FINITE ELEMENT ANALYSIS

Computer Assisted Design (CAD) and FEM tools are widely used in industry for different types of applications. Most of these tools are proprietary and their cost is often prohibitive for personal applications. In this section, we briefly describe a completely open-source (OS) framework/tool chain allowing to quickly design 3D models from scratch and turn them into FAUST physical models using MESH2FAUST.

OpenSCAD[5] is an open-source CAD program in which shapes are specified using a high-level functional programming language. While it allows to design complex 3D objects by combining or differentiating simple 3D elements (e.g., cubes, spheres, cylinders, etc.), it can also linearly or rotationally extrude 2D shapes specified as a polygon (expressed as a set of 2D Cartesian coordinates). Thus, MESH2FAUST comes with a modified version of Daniel Newman's Inkscape to OpenSCAD converter[6] allowing to export Inkscape[7] 2D paths to OpenSCAD. This is very useful to create more complex 3D shapes (see Figure 3) such as the one presented in §5. Various parameters such as the number of points (resolution) in the generated polygon can be configured, etc.

OpenSCAD can render 3D shapes as volumetric meshes using the STL (STereoLithography) format. However, these meshes are highly optimized and can't be used for finite element analysis. For example, flat squared surfaces will be rendered as two triangles regardless of their size which is not good. Instead, unstructured meshes with faces uniformly distributed across the object are better for FEM (the quality of the mesh is crucial to obtaining realistic modal parameters from the FEM). This can be achieved by exporting a high-resolution mesh from OpenSCAD to MeshLab,[8] and then carrying out a quadric edge collapse decimation to make the mesh uniform and specify the number of desired faces. A Laplacian smoother can help refine the

[5] `http://www.openscad.org/`
[6] `http://www.thingiverse.com/thing:25036`
[7] `https://inkscape.org/`
[8] `http://www.meshlab.net/`

quality of the previous operation. §5 presents an example of this system.

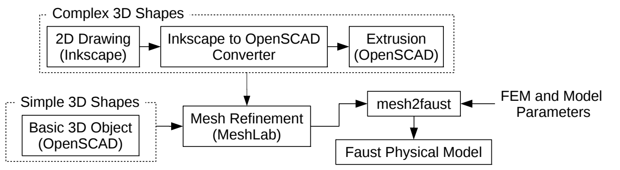

**Figure 3**. Open source framework to make FAUST modal physical models from scratch.

## 5. EVALUATION: BELL PHYSICAL MODELING

Bells can be considered as quasi-linear systems and can be successfully synthesized using modal synthesis. The acoustics of bells is well understood [10] and bell founders have been using FEM for decades to tune bells before making them [11].

In this section, we model a church bell after Rossing's elliptical arc approach using the framework described in the previous sections, and we compare the results of our system with the one published in his paper [10].

A Bezier curve was drawn on top of Rossing's church bell profile in Inkscape and was exported to OpenSCAD using the extension presented in §4. The radius of the bell was set to be 351mm, as in Rossing's paper. The result of this operation was a high definition mesh with about 25E4 vertices (see Figure 4). This number was arbitrarily chosen to provide a good balance between performance and quality.

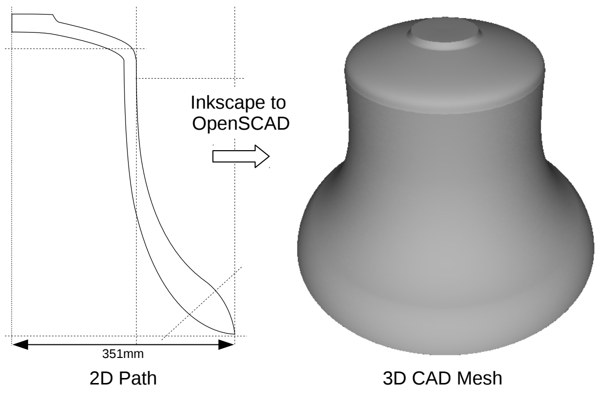

**Figure 4**. Church bell cross section and corresponding CAD model modeled after Rossing's elliptical arc approach.

This high density mesh was restructured in MeshLab using the technique described in §4 and down-sampled to a lower definition mesh with 15E3 vertices (see Figure 5).

This mesh was fed into MESH2FAUST with material parameters corresponding to bell metal [11] (Young's Modulus: 1.05E11 $N/m^2$, Poisson's Ratio: 0.33, and Density: 8600 $kg/m^3$). The results of the FEM modal analysis are presented in Figure 6 and plotted in Figure 7.

Figure 6 compares the theoretical "ideal" partial ratios to prime with the one computed by MESH2FAUST (the computed frequency of the undertone partial is 490.25 Hz). The modes naming conventions are the same as the one used by Rossing [10].

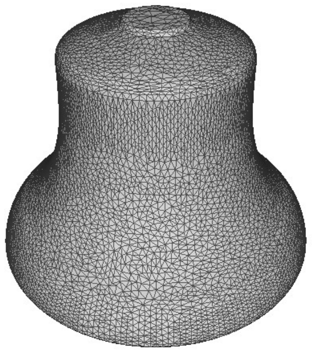

**Figure 5**. Mesh generated in MeshLab after quadric edge collapse decimation and Laplacian smoothing.

We can see that the FEM modes respect relatively well the theoretical mode hierarchy, resulting in very realistic synthesized sounds.

| Modes | Names of Partials | Theoretical Ratios | MTF Ratios |
|---|---|---|---|
| (2,0) | Hum, undertone | 0.500 | 0.500 |
| (2,1#) | Fundamental, prime | 1.000 | 1.012 |
| (3,1) | Tierce, minor third | 1.200 | 1.208 |
| (3,1#) | Quint, fifth | 1.500 | 1.6 |
| (4,1) | Nominal, octave | 2.000 | 1.980 |
| (4,1#) | Major Third, deciem | 2.500 | 2.451 |
| (2,2) | Fourth, undeciem | 2.667 | 2.610 |
| (5,1) | Twelfth, duodeciem | 3.000 | 3.073 |
| (6,1) | Upper octave | 4.000 | 4.11 |

**Figure 6**. Comparison between the theoretical "ideal" mode ratios to prime with the ones computed by MESH2FAUST for the bell mesh presented in Figure 5.

The same procedure was applied for a wide range of bells (e.g., carillon bells, hand bells, church bells from different countries, etc.). The results of this work and the corresponding FAUST-generated web apps synthesizer are available online. [9]

## 6. FUTURE DIRECTIONS

Currently, MESH2FAUST doesn't allow to estimate modes T60s. These exponential decays can be approximated by implementing a damping matrix. The Vega FEM Library already contains all the tools to do it so we plan to integrate this feature to a future version of MESH2FAUST.

## 7. CONCLUSIONS

MESH2FAUST, combined with the framework presented in §4, allows to easily design 3D models of musical instruments and turn them into physical models for sound synthesis. The combined forces of this system with the Faust Physical Modeling Library should hopefully greatly simplify the design and prototyping of novel physically-informed digital musical instruments.

[9] `https://ccrma.stanford.edu/~rmichon/pmFaust/#bells`

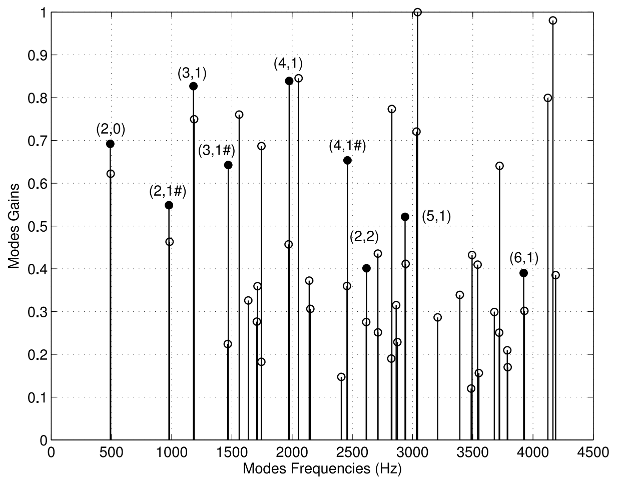

**Figure 7**. First fifty modes computed by MESH2FAUST for the bell mesh presented in Figure 6 for an excitation position matching the strike position of the clapper inside the bell.

MESH2FAUST and the Faust Physical Modeling Library are being developed as part of a project on augmenting mobile devices towards a "hybrid lutherie" [12] where 3D printed and electronic prostheses are added to mobile devices to turn them into musical instruments.

## 8. ACKNOWLEDGMENTS

This paper is part of the HYBRID project which has received funding from The Research Council of Norway through a FRIPRO Mobility Grant, contract no 240278/F20. The FRIPRO Mobility grant scheme (FRICON) is co-funded by the European Unions Seventh Framework Programme for research, technological development and demonstration under Marie Curie grant agreement no 608695.

## 9. REFERENCES

[1] C. Bruyns, "Modal Synthesis for Arbitrarily Shaped Objects," *Computer Music Journal*, vol. 30, no. 3, pp. 22–37, Autumn 2006.

[2] J.-M. Adrien, "The Missing Link: Modal Synthesis," in *Representations of Musical Signals*. Cambridge, USA: MIT Press, 1991, ch. The Missing Link: Modal Synthesis, pp. 269–298.

[3] R. Caussé, J. Bensoam, and N. Ellis, "Modalys, a physical modeling synthesizer: More than twenty years of researches, developments, and musical uses," *Journal of the Acoustical Society of America*, vol. 130, no. 4, 2011.

[4] Y. Orlarey, S. Letz, and D. Fober, *New Computational Paradigms for Computer Music*. Paris, France: Delatour, 2009, ch. "Faust: an Efficient Functional Approach to DSP Programming".

[5] R. Michon and J. O. Smith, "Faust-STK: a set of linear and nonlinear physical models for the Faust programming language," in *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Paris, France, September 2011.

[6] R. Michon, J. Smith, and Y. Orlarey, "New Signal Processing Libraries for Faust," in *Proceedings of the Linux Audio Conference (LAC-17)*, Saint-Etienne, France, May 2017, paper accepted to the conference but not published yet.

[7] J. O. Smith, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*. W3K Publishing, 2010. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/

[8] S. Fun Shing, D. Schroeder, and J. Barbi, "Vega: Nonlinear FEM Deformable Object Simulator," *Computer Graphics Forum*, vol. 32, no. 1, pp. 36–48, February 2013.

[9] H. Si, "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator," *ACM Trans. Math. Softw.*, vol. 41, no. 2, pp. 11:1–11:36, Feb. 2015.

[10] T. D. Rossing and R. Perrin, "Vibrations of Bells," *Applied Acoustics*, vol. 20, no. 1, pp. 41–70, December 1987.

[11] D. Bartocha and C. Baron, "Influence of Tin Bronze Melting and Pouring Parameters on Its Properties and Bells Tone," *Archives of Foundry Engineering*, vol. 16, no. 4, pp. 17–22, 2016.

[12] R. Michon, J. Smith, M. Wright, C. Chafe, J. Granzow, and G. Wang, "Passively Augmenting Mobile Devices Towards Hybrid Musical Instrument Design," in *Proceedings on the New Interfaces for Musical Expression Conference (NIME-17)*, Copenhagen, Denmark, May 2017, paper accepted to the conference but not published yet.