# Physical Audio Modeling

Modeling & Simulation Checkpoint 1

Benjamin Wilfong, Karl Hiner

April 2023

Github Repo: `https://github.gatech.edu/bwilfong3/CSE6730_Team3`

## 1 Project Description

### 1.1 Summary

We aim to create a model and application that converts a 3D volumetric mesh, in conjunction with general material properties, into a dynamic physical audio model, facilitating real-time user interaction for the application of excitation forces and modification of model parameters. While our ultimate model is designed to accommodate a wide variety of volumetric objects, our primary focus during this project will be the specialized domain of metallic bells.

### 1.2 System Description

The system of interest is a metal bell. An example bell and its axisymmetric cross section are given in Figure 1. Rather than modeling the entire 3D volume of the bell, we have elected to use the axisymmetry of the problem and model only the cross-section shown on the right-hand side of Figure 1. This will greatly decrease the computational cost associated with our model. The inputs to our model will be a two-dimensional cross-section, the Young's Modulus of the material, the Poisson's ratio of the material, and the density of the material.

### 1.3 Conceptual Model

Our conceptual model is based on that presented by Michon et. al.[8]. The overall structure of this approach is given in Figure 2. We split our model into two parts: 1) physical audio modeling and 2) finite element modeling. The details of each part follow.

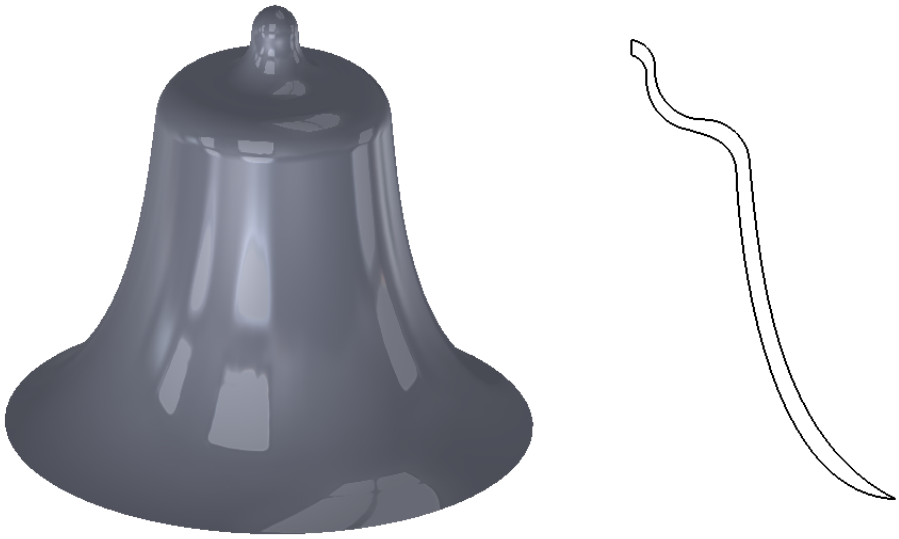

Figure 1: System of Interest

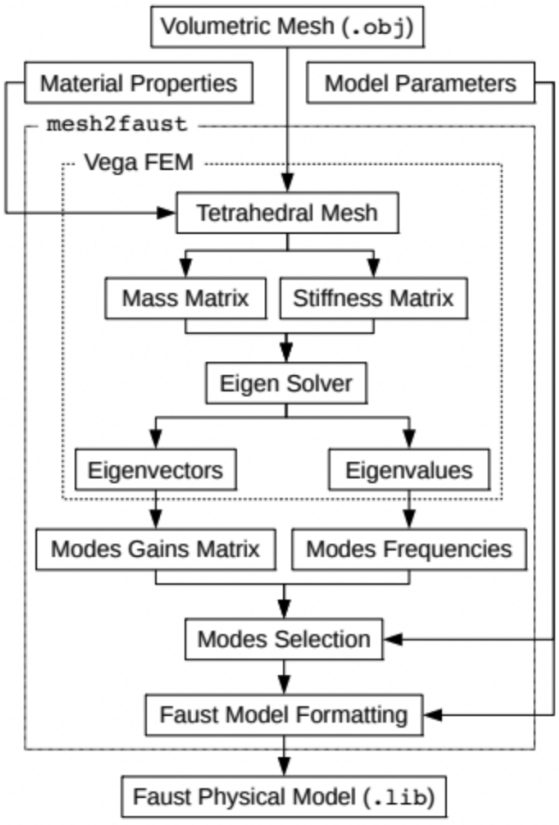


Figure 2: Program Structure [8]

### 1.3.1 Finite Element Modeling

At its core, the finite element method is one of many approaches to numerically solving differential equations. For our purposes, we will be using 2d, axisymmetric, triangular elements that are formulated to solve the linear elasticity equations in cylindrical coordinates:

$$\begin{aligned}\frac{\partial \sigma_{rr}}{\partial r} + \frac{1}{r}\frac{\partial \sigma_{r\theta}}{\partial \theta} + \frac{\partial \sigma_{rz}}{\partial z} + \frac{1}{r}(\sigma_{rr} - \sigma_{\theta\theta}) + F_r &= \rho \frac{\partial^2 u_r}{\partial t^2},\\ \frac{\partial \sigma_{r\theta}}{\partial r} + \frac{1}{r}\frac{\partial \sigma_{\theta\theta}}{\partial \theta} + \frac{\partial \sigma_{\theta z}}{\partial z} + \frac{2}{r}\sigma_{r\theta} + F_\theta &= \rho \frac{\partial^2 u_\theta}{\partial t^2},\\ \frac{\partial \sigma_{rz}}{\partial r} + \frac{1}{r}\frac{\partial \sigma_{\theta z}}{\partial \theta} + \frac{\partial \sigma_{zz}}{\partial z} + \frac{1}{r}\sigma_{rz} + F_z &= \rho \frac{\partial^2 u_z}{\partial t^2}.\end{aligned}$$

In these equations, $\sigma_{ij}$ denotes a stress on the $i^{th}$ face of a differential volume in the $j^{th}$ direction, $F_i$ denotes an external force in the $i^{th}$ direction, and $u_i$ denotes displacement in the $i^{th}$ direction. In a static finite element analysis, the displacements $u_i$ are found by solving a system of equations

$$\boldsymbol{K}\boldsymbol{x} = \boldsymbol{F}$$

where $\boldsymbol{K}$ is a matrix of stuffiness coefficients that relate the nodal displacements contained in $\boldsymbol{x}$ to the external forces contained in $\boldsymbol{F}$. In dynamic finite element analysis with damping ignored, the ODE

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{F} \tag{1}$$

is advanced from an initial condition $\boldsymbol{x}_0$. In this equation, $\boldsymbol{K}$ and $\boldsymbol{F}$ are as described earlier, and $\boldsymbol{M}$ is a matrix that takes element mass into account. Given that we are interested in finding frequencies and shapes of vibrations, which are time-dependent in nature, our modeling will utilize the latter of these two approaches.

To see the value of the eigenvalues and eigvenvectors assume that solutions to this system (1) have the form $u_i(t) = \mathbf{U}_i e^{j\omega_i t}$ where $\mathbf{U}_i \in \mathbb{R}^{2n}$ and $\omega_i \in \mathbb{R}$. Substituting this assumed solution into (1) yields the generalized eigenvalue problem

$$\mathbf{K}\mathbf{\Lambda} = \mathbf{M}\mathbf{U}\mathbf{\Lambda},$$

where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues and $\mathbf{U}$ is a modal matrix containing the eigenvectors $\mathbf{U}_i$ of the linear deformation equation.

Using the transformation $\mathbf{x} = \mathbf{U}\mathbf{q}$, the linear deformation equation can be decoupled giving

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

The solutions of the homogenous system $\ddot{\mathbf{q}} + \mathbf{\Lambda}\mathbf{q} = \mathbf{0}$ are given by a set of modes

$$q_i = a_i \sin(2\pi f_i + \theta_i),$$

where $a_i$ is the amplitude of excitation, $f_i$ is its frequency, and $\theta_i$ is a phase shift, assumed to be zero. The $a_i$ depend on the object and the excitement location, and the $f_i$ are given by

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

for each eigenvalue $\lambda_i$. These outputs are used as inputs to the modal analysis, yielding a profile for a synthesized sound.

### 1.3.2 Physical Audio Modeling

As explained in the literature review, we find the set of modes,

$$q_i = a_i \sin(2\pi f_i + \theta_i),$$

where $a_i$ is the amplitude of excitation, $f_i$ is its frequency, and $\theta_i$ is a phase shift, assumed to be zero. The $a_i$ depend on the object and the excitement location, and the $f_i$ are given by

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

for each eigenvalue $\lambda_i$.

Each mode is implemented as an exponentially decaying sine wave, with its own frequency, gain, and resonance duration (T60). These sine waves are implemented as resonant bandpass filters, allowing any signal to excite them. Specifically, the mode filters are implemented as a parallel bank of biquad filters with transfer function

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

with

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

where $g$, $f$, and $t_{60}$ are the mode gain, frequency, and T60, respectively.

## 1.4 Development Platforms

- For reading the volumetric mesh and deriving its mass and stiffness matrices, we use Fortran.
- For the main application interface and user interaction handling, we use the ImGui[5] and SDL

3[12] C++ libraries.

- We use the Spectra[10] C++ library, which is built on Eigen [3], to find the eigenvalues and eigenvectors of the sparse mass and stiffness matrices.
- After deriving the frequencies for the resonating modes, we generate the parallel digital audio filter bank using the Faust audio programming language [4].
- Finally, we use the MiniAudio [11] C++ library to continuously render the audio from the modal physical model to the output audio device.

# 2 Literature Review

Physical audio modeling offers many benefits compared with other digital audio synthesis techniques. Physically accurate models map directly to the physical systems they approximate, making interpretation and control intuitive. They generate audio with a natural and realistic character, and they are highly general; a physical model for one instrument can be adapted to another with relative ease, and complex systems can be composed and connected in straightforward ways.

However, physical audio models are also computationally expensive compared to other audio synthesis techniques, requiring simplifications to trade off realism and physical plausibility for computational tractability.

Digital waveguides [14], for example, model 1D systems such as strings or wind instruments by iterating the 1D wave equation with an appropriately chosen wave variable (e.g., string displacement or air pressure), modeling boundaries like plectra (picks), guitar bridges, or flute tone holes as scattering junctions with digital audio filters designed to match their wave impedance. However, waveguides are limited to 1D systems with low dispersion and do not lend themselves easily to nonlinear interactions [2].

Finite difference schemes, finite volume methods, and other time-stepping methods are more computationally costly but are much more general, offering nonlinear interactions, time-varying external interaction, and straightforward extension to two and three dimensions [2].

The technique of modal synthesis uses a source-filter model to decompose physical audio systems into an excitation signal that drives a parallel filter bank implementing the transfer functions of resonant modes in the physical system [13]. The resonant modes can be computed directly from a 3D model, and the source-filter audio model can be implemented compactly and efficiently with classical digital audio filter techniques. This approach offers a compromise between the generality and physical accuracy of finite difference schemes while still allowing for real-time sound synthesis in computationally constrained settings.

Modal synthesis is the approach we will investigate in our project. Specifically, we aim to *convert a volumetric mesh into a physical audio model.* 3D volumetric meshes are ubiquitous in graphics. The ability to interpret existing volumetric meshes as physically plausible audio generators would be of

great use for digital audiovisual media production, such as in video games or movies. Additionally, the modal synthesis paradigm supports extracting audio responses from external impulse forces, enabling estimation of the audio response from object interactions, as well as real-time instrument control in the music production context.

We will model our initial implementation on the work of Michon, Martin & Smith in their *Mesh2Faust* project [8]. In this approach, a bell is modeled using the finite element method using a 3-D tetrahedral mesh. The physics of the system is modeled using the linear deformation equation with no damping,

$$\mathbf{M\ddot{x}(t) + Kx(t) = f(}\boldsymbol{t}\mathbf{)}$$

where $\mathbf{x}(\boldsymbol{t})$ is the vector of displacements of each node, $\mathbf{M}$ and $\mathbf{K}$ are the finite element mass and stiffness matrices respectively, and $\mathbf{f}(\boldsymbol{t})$ is the force vector. It is in this modeling step that we hope to greatly reduce the computational cost by modeling only a 2d cross-section of a bell using axisymmetric triangular elements [9]. Assume that solutions to this system have the form $u_i(t) = \mathbf{U}_i e^{j\omega_i t}$ where $\mathbf{U}_i \in \mathbb{R}^{2n}$ and $\omega_i \in \mathbb{R}$. Substituting this assumed solution into the linear deformation equation yields the generalized eigenvalue problem

$$\mathbf{K\Lambda = MU\Lambda},$$

where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues and $\mathbf{U}$ is a modal matrix containing the eigenvectors $\mathbf{U}_i$ of the linear deformation equation. For this step, we intend to implement the *FEAST*[15] algorithm to find eigenpairs quickly.

Using the transformation $\mathbf{x = Uq}$, the linear deformation equation can be decoupled giving

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

The solutions of the homogenous system $\mathbf{\ddot{q} + \Lambda q = 0}$ are given by a set of modes

$$q_i = a_i \sin(2\pi f_i + \theta_i),$$

where $a_i$ is the amplitude of excitation, $f_i$ is its frequency, and $\theta_i$ is a phase shift, assumed to be zero. The $a_i$ depend on the object and the excitement location, and the $f_i$ are given by

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

for each eigenvalue $\lambda_i$. These outputs are used as inputs to the modal analysis mentioned previously, yielding a profile for a synthesized sound.

We hope to extend and improve on the existing work of Mesh2Faust in the following ways:

- *Improve slow compile times from mesh to modal parameters.* Mesh2Faust takes about 15 minutes to convert a mesh with ∼30k faces on a "regular laptop" [7].
- Mesh2Faust is a command line application and provides no control over generating the 3D

mesh itself. We aim to create a visual interface with controls for mesh generation & material properties, as well as parameters of the modal audio generation, with a focus on bell modeling.

- Mesh2Faust utilizes full 3D finite element analysis. We will investigate using 2d axisymmetric elements to take advantage of the inherent radial symmetry in bells. This will dramatically reduce the number of vertices, the computational cost, and the size of the gains matrix required to post-process the finite element results. We expect this simplification will allow for near real-time workflows when combined with GPU acceleration. However, imposing radial symmetry may produce less realism in the final audio result, and we plan to investigate this tradeoff.
- In addition, Mesh2Faust relies on Intel's MKL library[1], which is closed-source and only compatible with Intel processors. As part of our contribution, we plan to support running on non-Intel processors, such as ARM.

# 3 Progress Update

## 3.1 Physical Audio Modeling & User Interface

As a baseline, we have reproduced the full pipeline of the Mesh2Faust project, including the following

- Read .obj file containing a volumetric mesh of a bell.
- Generate the mass and stiffness matrices using the Vega C++ FEM library [6].
- Find the eigenvalues and eigenvectors using Intel's MKL [1] library. (We have also submitted a pull request to the Faust project to use the Spectra C++ library [10] instead of MKL to support non-Intel processors such as ARM.)
- Find the frequencies of select resonating modes and create a Faust program implementing a corresponding audio DSP program.

In addition, we have started on the user-facing application, with the following implemented currently:

- Read in a 3D mesh .obj file and visualize it, providing simple camera and rendering controls.
- Edit source code for the Faust audio programming language with a simple embedded text editor.
- Render the audio from the resulting DSP program to the audio output device.

## 3.2 Finite Element Modeling

The current capabilities of the finite element code are as follows:

1. Access command line arguments to define the mesh file and the material properties (density, Young's Modulus, and Poisson's Ratio).
2. Read a .obj file and fill arrays containing information about the vertex locations, element connectivity, and element centroids.
3. Use the arrays containing information about vertex locations, element connectivity, and element centroids to compute element stiffness matrices.

The next steps in implementation are as follows:

1. Use the arrays containing information about vertex locations, element connectivity, and element centroids to compute element mass matrices.
2. Use the element stiffness and mass matrices to form the global mass and stiffness matrices.
3. Solve the eigen-problem $\boldsymbol{K\Lambda} = \boldsymbol{MU\Lambda}$ and output the eigenvalues and eigenvectors as inputs to the physical audio modeling.

# References


[1] *Accelerate Fast Math with Intel® oneAPI Math Kernel Library*. URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html (visited on 03/14/2023).

[2] Stefan Bilbao et al. "The NESS Project: Physical Modeling, Algorithms and Sound Synthesis". In: *Computer Music Journal* 43 (Nov. 2019). DOI: 10.1162/comj_a_00516.

[3] *Eigen - A C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms*. 2023. URL: https://eigen.tuxfamily.org/ (visited on 02/04/2023).

[4] *Faust - Functional programming language for signal processing and sound synthesis*. 2023. URL: https://github.com/grame-cncm/faust (visited on 02/04/2023).

[5] *ImGui*. URL: https://github.com/ocornut/imgui (visited on 02/04/2023).

[6] Daniel Schroeder: Jernej Barbič Fun Shing Sin. *Vega FEM Library*. URL: http://www.jernejbarbic.com/vega (visited on 02/04/2023).

[7] Romain Michon. *Romain Michon - Faust Tutorials*. URL: https://ccrma.stanford.edu/~rmichon/faustTutorials/#converting-a-mesh-to-a-faust-physical-model (visited on 03/10/2023).

[8] Romain Michon, Sara R Martin, and Julius O Smith. "MESH2FAUST: A Modal Physical Model Generator for the Faust Programming Language -Application to Bell Modeling". In: (Mar. 2021).

[9] Woolley Mitchell and Fisher. *Formulation and Experimental Verification of an Axisymmetric Finite-Element Structural Analysis*. 1971. URL: https://nvlpubs.nist.gov/nistpubs/jres/75C/jresv75Cn3-4p155_A1b.pdf.

[10] Yixuan Qiu. *Spectra - C++ Library For Large Scale Eigenvalue Problem*. 2023. URL: https://spectralib.org/ (visited on 02/04/2023).

[11] David Reid. *MiniAudio - A single file library for audio playback and capture*. 2023. URL: https://github.com/mackron/miniaudio (visited on 02/04/2023).

[12] *SDL 3*. URL: https://github.com/libsdl-org/SDL (visited on 02/04/2023).

[13] Julius O. Smith. "Physical Audio Signal Processing". In: (accessed March 2023). online book, 2010 edition. URL: https://ccrma.stanford.edu/~jos/pasp/Modal_Representation.html.

[14] Julius O. Smith. "Physical Modeling Using Digital Waveguides". In: *Computer Music Journal* 16.4 (1992), pp. 74–91. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3680470 (visited on 03/10/2023).

[15] Ping Tak Peter Tang and Eric Polizzi. *FEAST as a Subspace Iteration Eigensolver Accelerated by Approximate Spectral Projection*. 2013. URL: https://arxiv.org/abs/1302.0432.