An Introduction to the Synth-A-Modeler Compiler: Modular and Open-Source Sound Synthesis using Physical Models

Edgar Berdahl, Julius O Smith

▶ To cite this version:

Edgar Berdahl, Julius O Smith. An Introduction to the Synth-A-Modeler Compiler: Modular and Open-Source Sound Synthesis using Physical Models. Proceedings of the 10th International Linux Audio Conference (LAC-12), Apr 2012, Stanford, United States. hal-03162970

HAL Id: hal-03162970
https://hal.science/hal-03162970

Submitted on 8 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An Introduction to the *Synth-A-Modeler* Compiler: Modular and Open-Source Sound Synthesis using Physical Models

Edgar BERDAHL
Audio Communication Group, TU Berlin
Einsteinufer 17c
10587 Berlin
Germany
eberdahl@mail.tu-berlin.de

Julius O. SMITH III
CCRMA, Stanford University
Stanford, CA
94305
USA
jos@ccrma.stanford.edu

Abstract

The tool is not a synthesizer—it is a Synth-A-Modeler! This paper introduces the Synth-A-Modeler compiler, which enables artists to synthesize binary DSP modules according to mechanical analog model specifications. This open-source tool promotes modular design and ease of use. By leveraging the Faust DSP programming environment, an output Pd, Max/MSP, SuperCollider, VST, LADSPA, or other external module is created, allowing the artist to hear the sound of the physical model in real time using an audio host application. To show how the compiler works, the example model "touch a resonator" is presented.

Keywords

physical modeling, virtual acoustics, Faust DSP, haptic force-feedback, open source

1 Introduction

Simulating the equations of motion of acoustic musical instruments, also known as physical modeling, has been employed for decades to synthesize sound digitally [Smith, 2010; Smith III, 1982; Cadoz et al., 1981]. It is an intriguing paradigm for synthesizing sound in new media applications because it enables sound synthesis for fictional acoustic-like instruments that would be impractical or very labor-intensive to construct in real life.

Physical modeling DSP algorithms incorporate internal feedback loops, which means that an error or inaccuracy in the modeling algorithm can cause the algorithm to become unstable. This results in a loud sound, which can be very unpleasant for an artist working with a physical model.

Several tools have been packaged for implementing modular physical modeling with algorithms that are designed to be stable. The basic concept is that the artist specifies an interconnection of passive mechanical, acoustical, and/or electrical elements, rather than programming any equations. This makes it much easier for artists to employ physical modeling to make new sounds without needing to understand all of the details under the hood.

However, most of these tools have not been immediately available to artists, partly due to the cost of purchasing the tools. For example, the Modalys modal synthesis environment requires both the purchase of Max/MSP and the IRCAM Forum Recherche [Ellis et al., 2005]. GENESIS is a complete modeling package and can be purchased from the Association pour la Création et la Recherche sur les Outils d'Expression [Castagne and Cadoz, 2002]. GENESIS includes a graphical user interface, which makes it easier for artists to specify the interconnection of the mechanical elements.

In contrast, the BlockCompiler by Matti Karjalainen is open-source; however, it is a complex package that has been rewritten at least three times and requires the artist to write Lisp code to create models [Karjalainen, 2003]. Stefan Bilbao has written a modular environment for synthesizing percussion sounds, but it runs in MATLAB and so is not immediately applicable to real-time synthesis [Bilbao, 2009]. Finally, the Synthesis ToolKit (STK) has become popular due to its MIDI capability and ability to run within Max/MSP, Pd, and other sound synthesis environments [Cook and Scavone, 1999].[1] However, new models can only be created in the STK by artists who can manually write stable difference equations in C++ for physical modeling.

For the above reasons, we decided to create a new, free and open-source tool for modular sound

[1] See `http://ccrma.stanford.edu/software/stk`

synthesis using physical models.

2 Synth-A-Modeler

2.1 Requirements

The following requirements of the Synth-A-Modeler project have guided the design. The Synth-A-Modeler compiler should

- enable efficient real-time physical modeling sound synthesis for new media applications,
- be free and open-source,
- be as modular as possible,
- be easy to extend and modify,
- serve as a platform for pedagogical exploration of the physics of mechanically vibrating systems,
- be accessible to artists who may have little or no experience in programming, digital signal processing (DSP), or physics,
- be accessible from as many sound synthesis host environments as possible,
- enable the development of MIDI-based synthesizers, and
- be compatible with programming haptic force-feedback systems.

2.2 Faust

To enable efficient real-time synthesis while targeting as many host environments as possible, we decided to use the Functional AUdio STreaming (Faust) programming language [Orlarey et al., 2009; Barkati et al., 2011; Orlarey et al., 2002]. In fact, some physical models had already been written directly in Faust code [Michon and Smith III, 2011]; however, most artists would not be ready to put in the detailed effort required to program physical models in Faust. Hence, we planned to extend Faust with the development of Synth-A-Modeler.

2.3 Dataflow

Consequently, we adopted the dataflow shown in Figure 1. The Synth-A-Modeler compiler receives a netlist-like model specification in an MDL file and compiles it into Faust DSP code [Vladimirescu, 1994]. Then the Faust compiler together with `g++` can transform the code into a target binary format as suitable for Pd, SuperCollider, Max/MSP, CSound, LADSPA plug-in, VST plug-in, a generic audio application, generic C++ code, or other host target as desired by the artist.

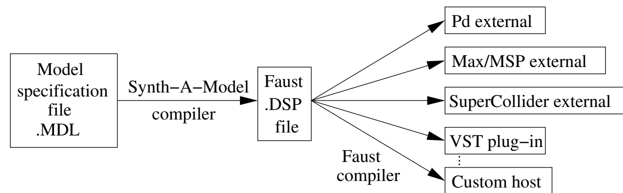

Figure 1: Dataflow for synthesizing a model with Synth-A-Modeler

2.4 Specifying MDL Files

For synthesizing sound with traditional signal flow-based approaches such as Pure Data (pd), Max/MSP, Simulink, or others, a user specifies a directed graph of sound sources, processing elements, and sound outputs. The signal flow is considered to be unidirectional: from the sources to the sinks.

However, in the case of physical modeling, the signal flow is *bidirectional* among the elements. One reason for this is Newton's third law: "For every action, there is an equal and opposite reaction." Hence, in physical modeling, a graph with bidirectional edges describes the signal flow. An engineer must be keenly aware of the bidirectional signal flow between each pair of elements; however, an artist designing a physical model using a modular approach needs only to understand which elements are connected to which. For this reason, the artist can simply specify an *un*directed graph of virtual physical elements [Castagne and Cadoz, 2002]. For the Synth-A-Modeler compiler, the artist specifies the model in an MDL file by specifying connections between elements, such as digital waveguides, masses, springs, dampers, ports, etc., using a netlist-like format with some extensions. Of course the artist may also specify the physical parameters for each element.

3 Modeling Paradigm

3.1 Example Model

Consider the model shown in Figure 2, which implements a very simple synthesizer with only a

single resonance frequency. The model describes a series of mechanical elements that connect to a user's finger, allowing the user to "touch" a virtual mechanical resonator. This model features objects as in GENESIS and CORDIS-ANIMA, except that the units are SI units and English names are employed [Castagne and Cadoz, 2002][Cadoz et al., 1993][Kontogeorgakopoulos and Cadoz, 2007]. The following text specifies the same model using an MDL model specification file:

```
link(4200.0,0.001),ll,m1,g,();
touch(1000.0,0.03,0.0),tt,m1,dev1,();

mass(0.001),m1,();
ground(0.0),g,();
port( ),dev1,();

audioout,a1,m1,1000.0;
```

The external `port` named *dev1* is connected by a `touch` link *tt* to a `mass` *m1* of 0.001 kg. The mass resonates because the linear `link` *ll* connects it to mechanical `ground` *g*, which always remains at the position 0 m. The linear `link` *ll* consists of the parallel combination of a spring with stiffness 4200 N/m and damper with parameter 0.001 N/(m/s). The touch link `tt` (analogous to the *BUT* element in GENESIS and CORDIS-ANIMA) is similar to a linear link, except it only results in a force when one of the objects is pushing "inside" the other one [Kontogeorgakopoulos and Cadoz, 2007].

3.2 Inputs And Outputs

A port models a mechanical connection from within a model to the outside. By default each port brings an external position signal into the model and sends a collocated force signal outside the model. This is why a port can easily be used to control a haptic force-feedback device [Berdahl, 2009].

In addition, the `audioout` object in Synth-A-Modeler provides a simple audio output. When applied to a mass-like object, it outputs position (see Figure 2, right), and when applied to a link-like object, it outputs force.

3.3 Resonator Abstraction

The generic `resonator` object (related to CEL in GENESIS) could have been employed instead of

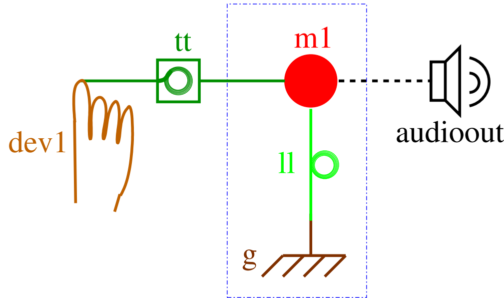

Figure 2: Model for "touch a resonator"

the content within the dash-dotted box in Figure 2. Our resonator implementation in Synth-A-Modeler allows for the frequency and damping time of the resonator to be adjusted in real-time while minimizing transients. Modalys might operate in a similar fashion as it also allows interpolation of resonance frequencies in real time. In Synth-A-Modeler, we use a state-space implementation employing a well-conditioned rotation matrix [Mathews and Smith III, 2003]. Max Mathews employed banks of this style of resonator in his piece *Angel Phasered*, which was performed at the CCRMA Transitions Concert on Sept. 16, 2010.

4 The Synth-A-Modeler Compiler

4.1 Strategy

Although it is possible to represent any explicitly computable linear block diagram in Faust [Orlarey et al., 2002], Faust's block diagram algebra is oriented toward signals that flow from the left to the right. To obtain a signal flowing back from the right to the left, it is necessary to employ Faust's recursive composition operator, which automatically incorporates a single sample of delay. In physical modeling, signals are bidirectional, which means that roughly half of the signals must flow from the right to the left. However, it is challenging to organize Faust networks in this fashion without inserting a plethora of single-sample delays, which can detract from the stability of the models.

For example, consider the Faust code for a mass of m [kg] with zero initial conditions and a linear link but with stiffness k [N/m], damping factor R [N/(m/s)], and spring centering position offset o [m], as given in Figure 3. These equations are similar to those in CORDIS-ANIMA and GENE-

```
mass(m) = (/(m*fs*fs) : ((_,_ : +) ~ _) : ((_,_ : +) ~ _));

link(k,R,o) = _ : (_-o) <: (_,_) : (*(k), (_<:(_,_):(_,mem):-:*(R*fs))) : (_,_):+:_;
```

Figure 3: Faust code for a mass and a linear link from `physicalmodeling.lib`

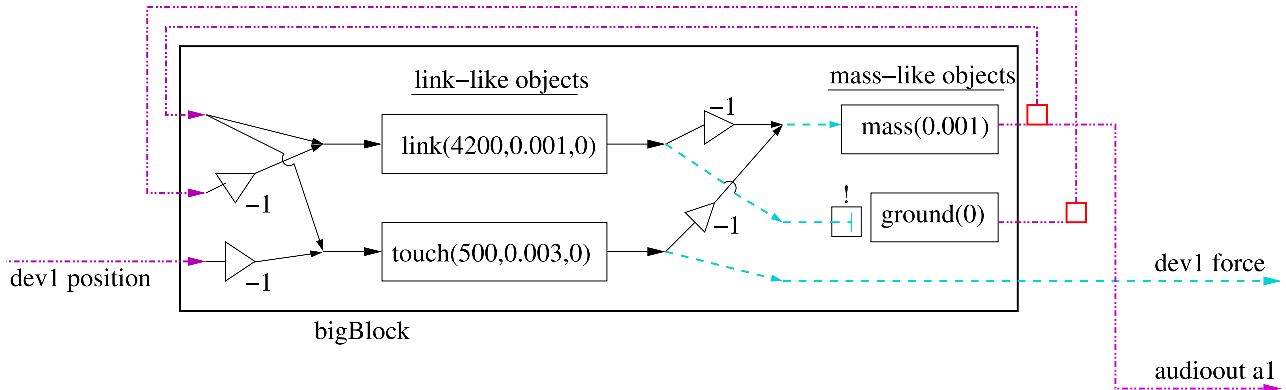

Figure 4: Representation of Faust DSP code for "touch a resonator"

SIS, except that the units are SI units.

Neither the mass nor the link incorporates a single sample of delay; however, in order to wrap these specific elements in feedback about one another, it is necessary to insert samples of delay to make the network explicitly computable. In the CORDIS-ANIMA equations, the delay is included in the masses [Kontogeorgakopoulos and Cadoz, 2007], so we follow suit by having Faust's right to left signals emanate from the masses.

For example, Figure 4 shows the representation of the Faust DSP code for the model depicted in Figure 2. The mass-like objects are placed on the right, and the link-like objects are placed further to the left. The signals representing displacements are shown in magenta, and the signals representing net forces are shown dashed in cyan (see Figure 4—for color see the online version of this paper). Linear combinations are formed in order to properly interconnect the mass-like objects and link-like objects. Finally, the single-sample of delay per feedback loop is represented on the far right using the small red boxes as in the Faust diagram notation. According to this interconnection strategy, there is no sample of delay associated with the link-like objects.

The single port in the model (ref. "dev1" in Figure 2) is responsible for the position signal input (see Figure 4, bottom left dash-dot-dotted in magenta) and the force signal output (see Figure 4, bottom dashed in cyan) could be connected to an impedance controlled haptic force-feedback device in order to control the sound synthesis. Alternatively, using a more conventional kind of new media controller, the position input could be used independently of any force output to control the sound synthesis. Finally, the additional audio output `a1`, which corresponds to the position of the virtual mass, is also provided (see Figure 4, right dash-dot-dotted in magenta) as a more sensible audio output.

4.2 Compiled Code

Applying the Synth-A-Modeler compiler to the model specification MDL file given in Section 3.1 results in the Faust code presented in Figure 5. The code begins by first importing the physical modeling library, which contains Faust code describing how each of the elements works. Then `bigBlock` is defined about which the feedback paths will be wrapped. In this case, `m1` is fed back, which represents the position of mass *m1*, and the ground position `g` is also fed back. The letter "p" is appended to each variable fed back, denoting *previous*, since the variable is delayed by a single sample (see the small red squares on the

```
import("physicalmodeling.lib");

bigBlock(m1p,gp,dev1p) = (m1,g,dev1,a1) with {
     // Link-like objects:
     ll = (m1p - gp) : link(4200.0,0.001,0.0);
     tt = (m1p - dev1p) : touch(1000.0,0.03,0.0);

     // Mass-like objects:
     m1 = (0.0-ll-tt) : mass(0.001);
     g = (0.0+ll) : ground(0.0);
     dev1 = (0.0+tt);

     // Additional audio output
     a1 = 0.0+m1*(1000.0);
};

process = (bigBlock)~(_,_):(!,!,_,_);
```

Figure 5: Compiled Faust DSP code for implementing "touch a resonator"

right-hand side of Figure 4).

The identifier for each element (e.g. `ll`, `tt`, `m1`, `g`, `dev1`, and `a1`) is then employed as an output variable from the element, which can only be accessed from within `bigBlock`. The inputs to the elements are formed as linear combinations of the other variables (see Figure 5).

4.3 Example Pure Data Patch

Next the Faust compiler together with `g++` can compile the Faust code into a target format as suitable for an audio host application. We briefly present an example of compiling our example into a Pure Data (pd) external object called `touch_a_resonator~`. The pd patch in Figure 6 shows how the external object can be employed for real-time sound synthesis without requiring a haptic force-feedback user input device. The leftmost inlet and outlet are automatically generated by the faust2pd script,[2] and we do not use them in this example. The remaining audio inlets and outlets (see Figure 6) are ordered in correspondence with the input and outputs in Figure 4 and in the third line of Figure 5.

The rightmost inlet corresponds to the input position `dev1`. A horizontal slider GUI object from pd is employed as a user interface. The slider's output is converted into an "audio" signal, smoothed, and fed into the `dev1` position input.

The force outlet from the `dev1` output (see Figure 6) is not needed in this case since there is no mechanism to provide force feedback. The audio output comes from the additional `a1` audio output that was specified in the model. Despite having only a single resonance frequency, we find that the quality of the user interaction with the model is intriguing, due to our process of physically modeling as many elements as possible in the interaction loop being simulated.

5 Conclusion

Due to its modular character, the Synth-A-Modeler compiler enables the creation of complex models for artistic applications using simple building blocks. In some sense, it is an extension

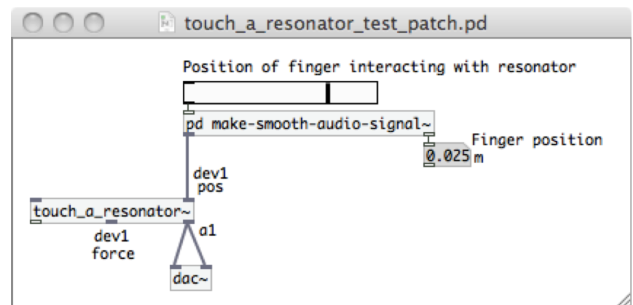

Figure 6: Example pd patch for "touch a resonator"

[2]More information is available on this inlet and outlet [Graef, 2007].

of our own prior work, rewritten to employ Faust for efficient DSP and to target multiple host applications [Berdahl et al., 2010].

We are currently working to add support for digital-waveguide objects to the Synth-A-Modeler compiler. We look forward to an open-source platform for prototyping, developing, and releasing physical modeling binaries that incorporate the popular modeling techniques of mass and link (i.e. mass-interaction) modeling, modal synthesis, and digital waveguide synthesis. The implementation is simple enough that we hope other developers can easily add support for further modeling formalisms.

Due to the open-format licensing of Synth-A-Modeler via the GPL version 2, we believe that Synth-A-Modeler could be especially attractive for commercial applications for compiling optimized and fine-tuned models into portable binary modules. We hope that this way we can create a large user base including industrial, artistic, and scientific users.

6 Acknowledgments

We graciously thank Alexandros Kontogeorgakopoulos, Claude Cadoz, Stefan Weinzierl, Annie Luciani, Andre Bartetzki, Chris Chafe, and the Alexander von Humboldt foundation.

References

K. Barkati, D. Fober, S. Letz, and Y. Orlarey. 2011. Two recent extensions to the FAUST compiler. In *Proc. of the Linux Audio Conference*, Maynooth, Ireland.

Edgar Berdahl, Alexandros Kontogeorgakopoulos, and Dan Overholt. 2010. HSP v2: Haptic signal processing with extensions for physical modeling. In *Proceedings of the Haptic Audio Interaction Design Conference*, pages 61–62, Copenhagen, Denmark.

Edgar Berdahl. 2009. *Applications of Feedback Control to Musical Instrument Design.* Ph.D. thesis, Stanford University, Stanford, CA, USA, December.

Stefan Bilbao. 2009. A modular percussion synthesis environment. In *Proc. 12th Int. Conference on Digital Audio Effects (DAFx-09)*, Como, Italy.

Claude Cadoz, Annie Luciani, and Jean-Loup Florens. 1981. Synthèse musicale par simulation des mécanismes instrumentaux. *Revue d'acouqistique*, 59:279–292.

Claude Cadoz, Annie Luciani, and Jean-Loup Florens. 1993. CORDIS-ANIMA: A modeling and simulation system for sound and image synthesis—The general formalism. *Computer Music Journal*, 17(1):19–29.

Nicolas Castagne and Claude Cadoz. 2002. Creating music by means of 'physical thinking': The musician oriented Genesis environment. In *Proc. 5th Internat'l Conference on Digital Audio Effects*, pages 169–174, Hamburg, Germany.

Perry Cook and Gary Scavone. 1999. The synthesis toolkit (STK), version 2.1. In *Proc. Internat'l Computer Music Conf.*, Beijing, China.

Nicholas Ellis, Joël Bensoam, and René Caussé. 2005. Modalys demonstration. In *Proc. Int. Comp. Music Conf. (ICMC'05)*, pages 101–102, Barcelona, Spain.

Albert Graef. 2007. Interfacing Pure Data with Faust. In *Proc. of the Linux Audio Conference*, Technical University of Berlin, Berlin, Germany.

Matti Karjalainen. 2003. BlockCompiler: Efficient simulation of acoustic and audio systems. In *Proc. 114th Convention of the Audio Engineering Society*, Preprint #5756, Amsterdam, The Netherlands.

Alexandros Kontogeorgakopoulos and Claude Cadoz. 2007. Cordis Anima physical modeling and simulation system analysis. In *Proc. 4th Sound and Music Computing Conference*, pages 275–282, Lefkada, Greece.

Max Mathews and Julius O. Smith III. 2003. Methods for synthesizing very high Q parametrically well behaved two pole filters. In *Proc. Stockholm Musical Acoustic Conference (SMAC)*, Stockholm, Sweden.

Romain Michon and Julius O. Smith III. 2011. Faust-STK: A set of linear and nonlinear physical models for the Faust programming language. In *Proc. 14th Int. Conference on Digital Audio Effects (DAFx-11)*, Paris, France.

Yann Orlarey, Dominique Fober, and Stéphane Letz. 2002. An algebra for block diagram languages. In *Proceedings of International Computer Music Conference*, Göteborg, Sweden.

Yann Orlarey, Dominique Fober, and Stéphane Letz. 2009. *Faust: an Efficient Functional Approach to DSP Programming*. Edition Delatour, France.

J.O. Smith III. 1982. Synthesis of bowed strings. In *Proceedings of the International Computer Music Conference*, Venice, Italy.

Julius O. Smith. 2010. *Physical Audio Signal Processing: For Virtual Musical Instruments and Audio Effects*. W3K Publishing, `http://ccrma.stanford.edu/~jos/pasp/`, December, ISBN 978-0-9745607-2-4.

Andrei Vladimirescu. 1994. *The SPICE Book*. Wiley, Hoboken, NJ.