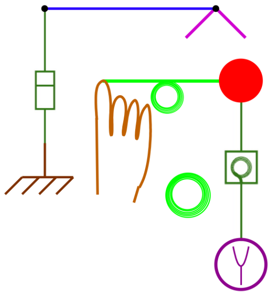
An Introduction to the Synth-A-Modeler Compiler

Modular and Open-Source Sound Synthesis using Physical Models

Edgar Berdahl and Julius O. Smith III



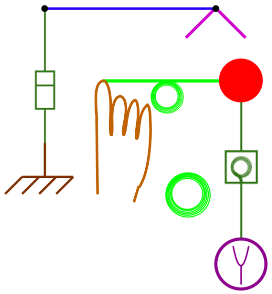
INTRODUCTION

FAUST PROGRAMMING LANGUAGE

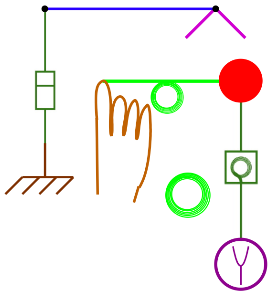
SYNTH-A-MODELER

MORE EXAMPLES

FINAL WORDS

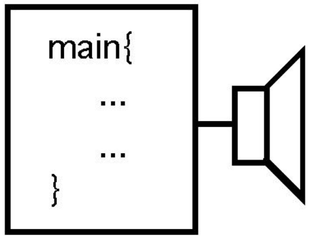


Digital Sound Synthesis

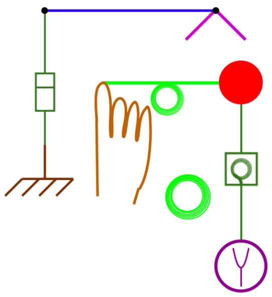


Digital Sound Synthesis

- We can create any perceivable sound using digital sound synthesis.

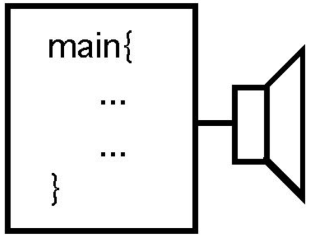


```
main{
    ...
    ...
}
```

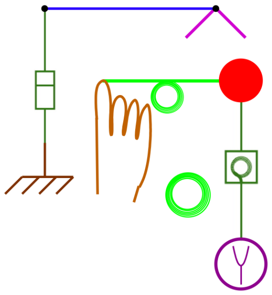


Digital Sound Synthesis

- We can create any perceivable sound using digital sound synthesis.
- Almost all sounds created by computers are either not interesting, ugly, unpleasant, painful or dangerous.



```
main{
     ...
     ...
}
```

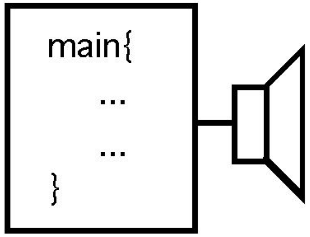


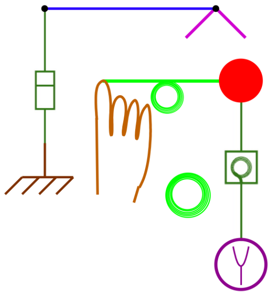
Digital Sound Synthesis

- We can create any perceivable sound using digital sound synthesis.

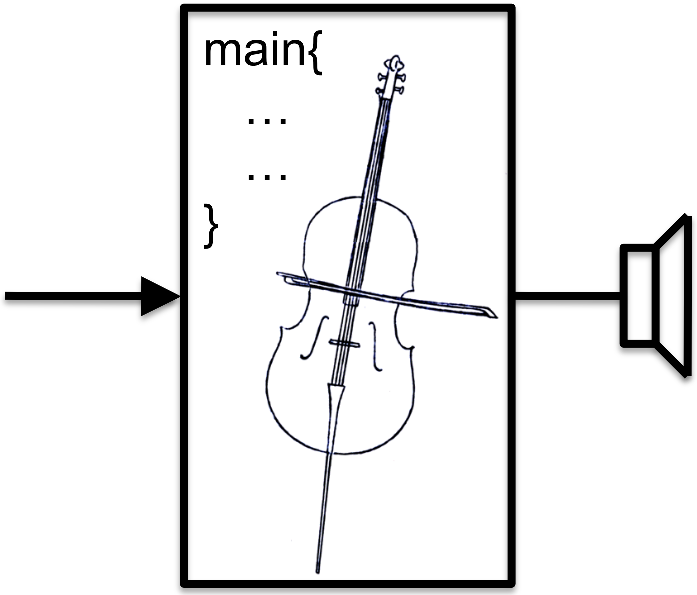
- Almost all sounds created by computers are either not interesting, ugly, unpleasant, painful or dangerous

- How do we obtain the specific sounds that we want?

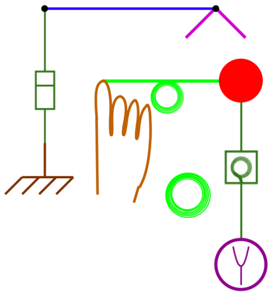




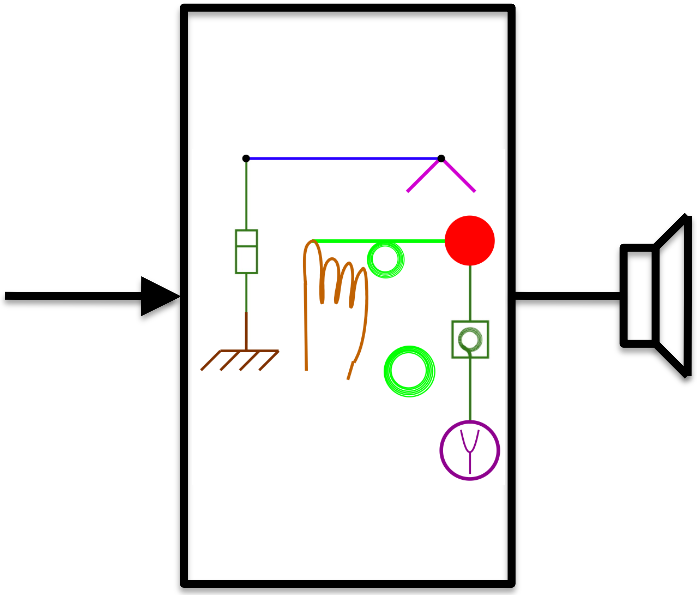
Physical Modeling

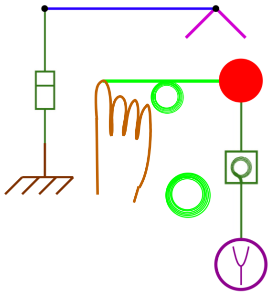


```
main{
    ...
    ...
}
```



Physical Modeling

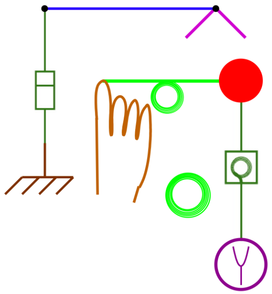




- thinking of physical modeling as a creative activity or a design process

Requirements

- produce efficient and modern DSP code for real-time applications
- free and open-source
- combine digital waveguide, mass-interaction, and modal synthesis
- easy to quickly design a large number of models
- easy to extend and modify framework
- platform for pedagogical exploration of mechanics and dynamics
- accessible to artists who may have limited technical experience
- enable the development of MIDI-based synthesizers
- compatible with programming haptic force-feedback systems
- accessible from as many sound synthesis host environments as possible



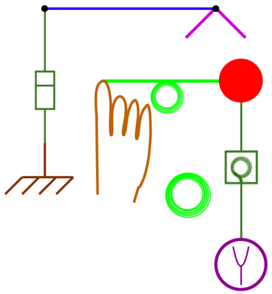
INTRODUCTION

FAUST PROGRAMMING LANGUAGE

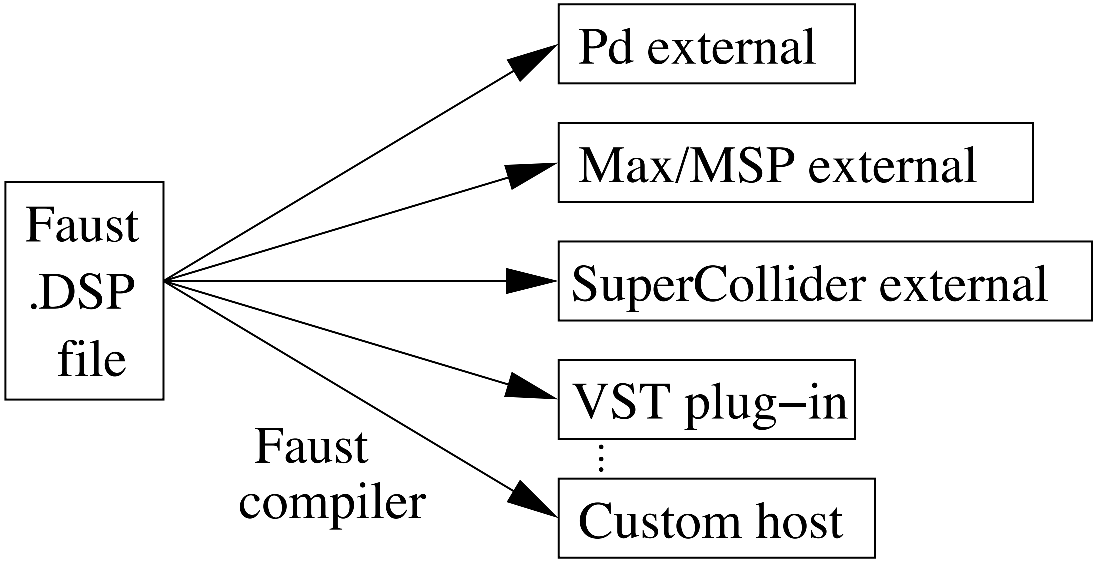
SYNTH-A-MODELER

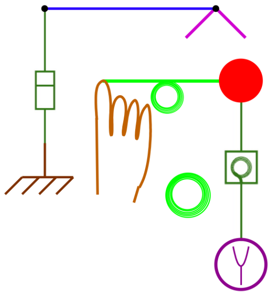
MORE EXAMPLES

FINAL WORDS



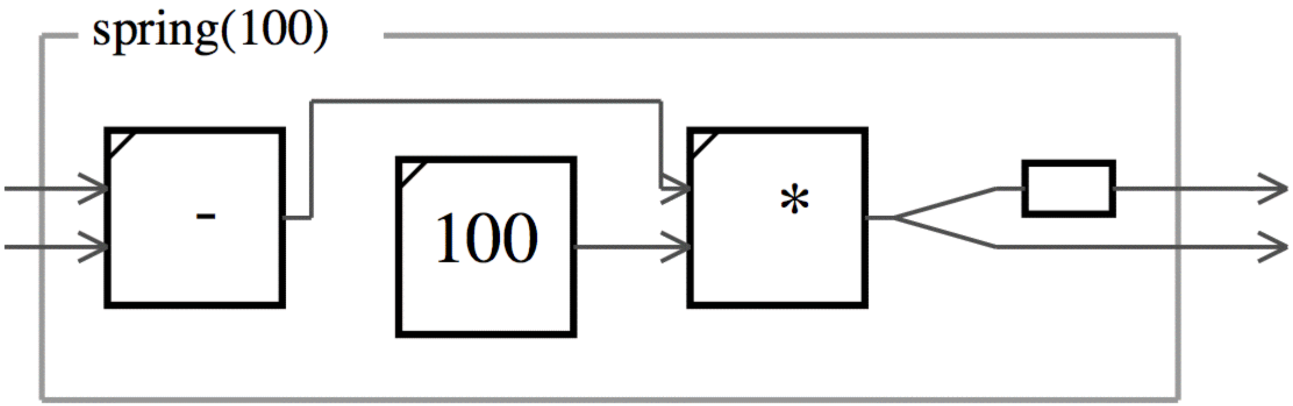
Faust Dataflow Summary

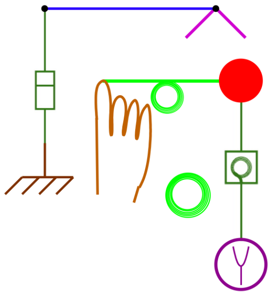




Functional **AU**dio **ST**ream

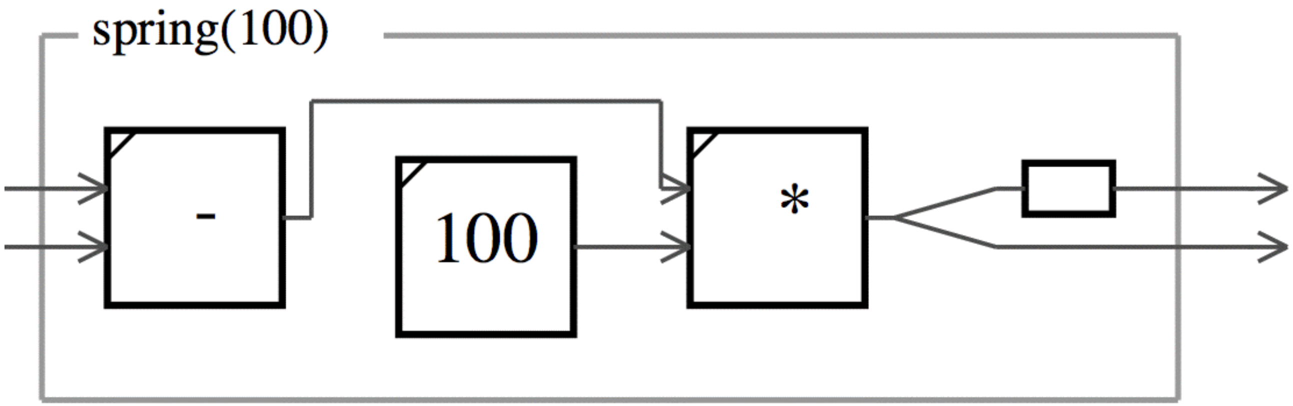
```
spring(k) = (_,_) : - : *(k) : _ <: (*(-1.0),_);

process = spring(100.0);
```



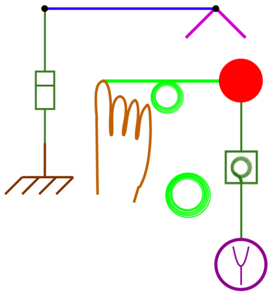


Functional **AU**dio **ST**ream

```
spring(k) = (_,_) : - : *(k) : _ <: (*(-1.0),_);

process = spring(100.0);
```



But FAUST signal flow is primarily left to right



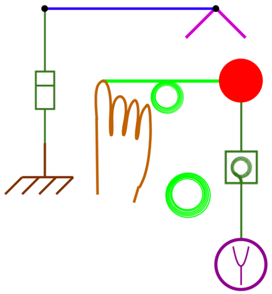
INTRODUCTION

FAUST PROGRAMMING LANGUAGE

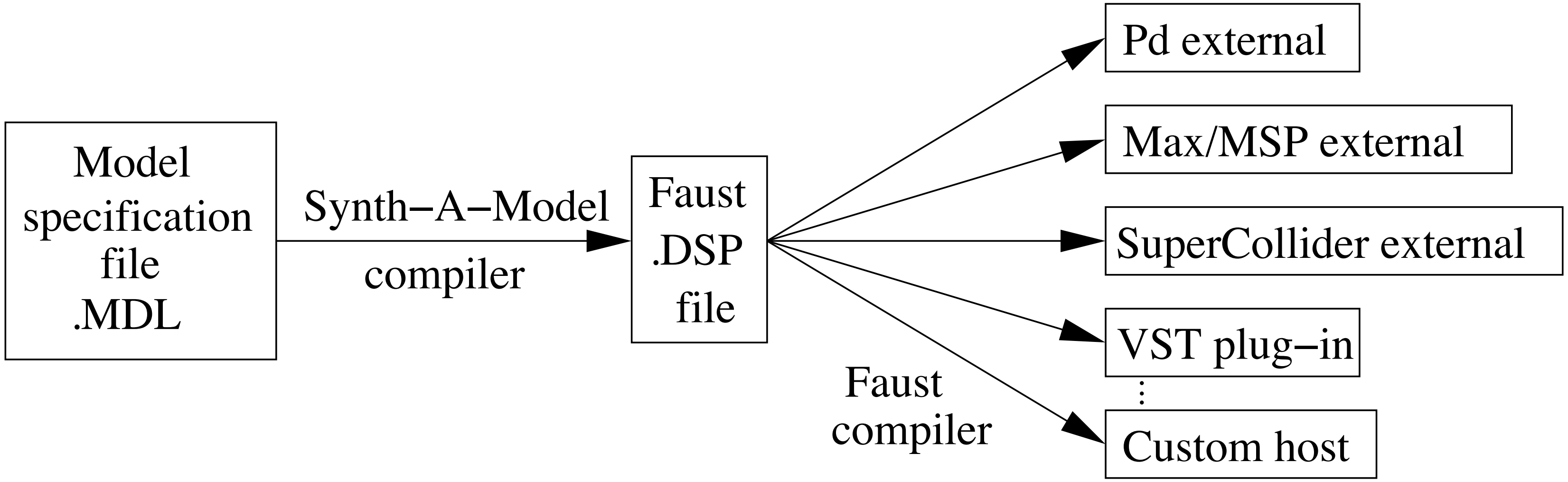
SYNTH-A-MODELER

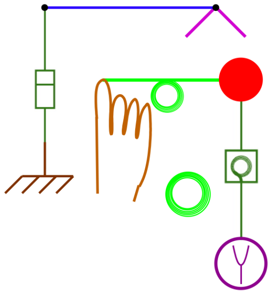
MORE EXAMPLES

FINAL WORDS



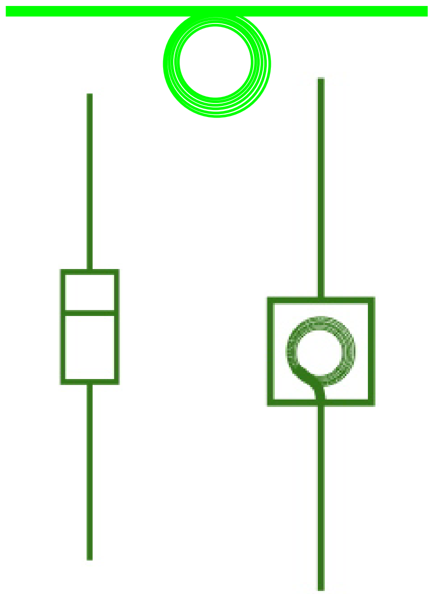
Synthesizing A *Model*



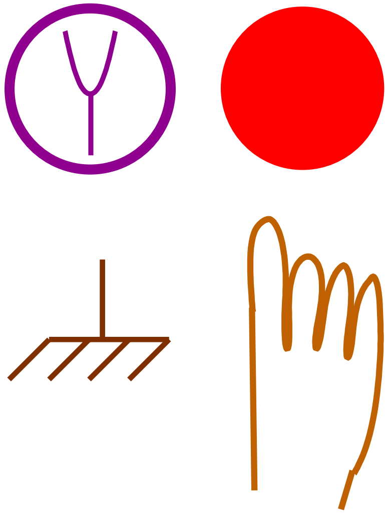


Elements

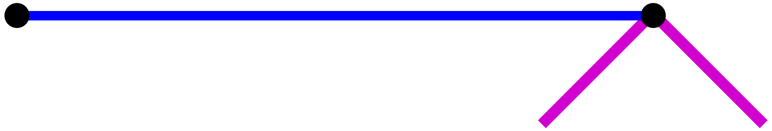
Link-like elements

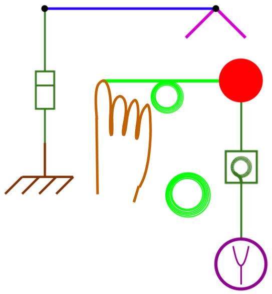


Mass-like elements

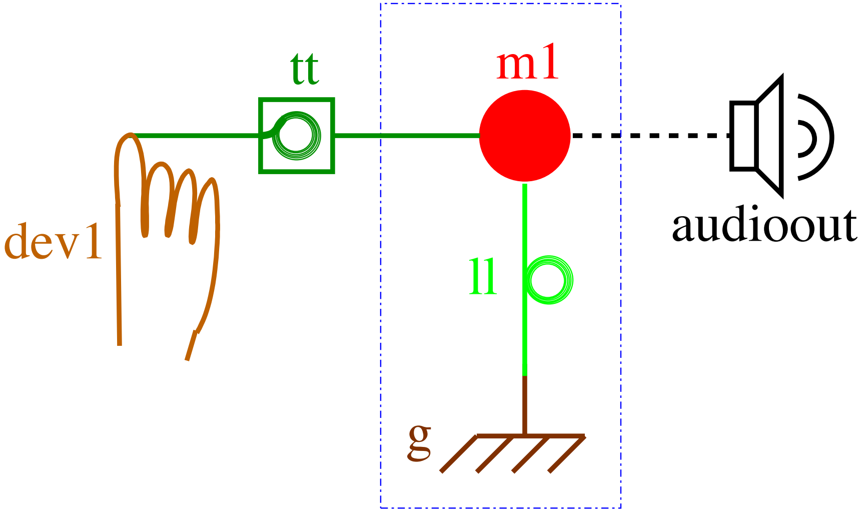


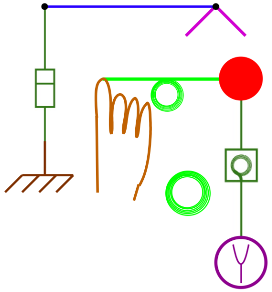
Wave variable elements



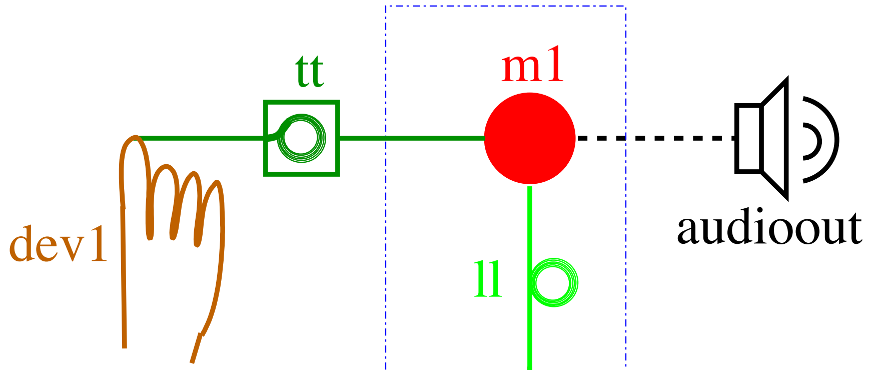


Example Model: Play A Resonator

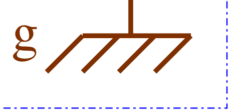




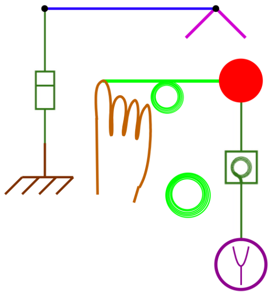
Example Model: Play A Resonator



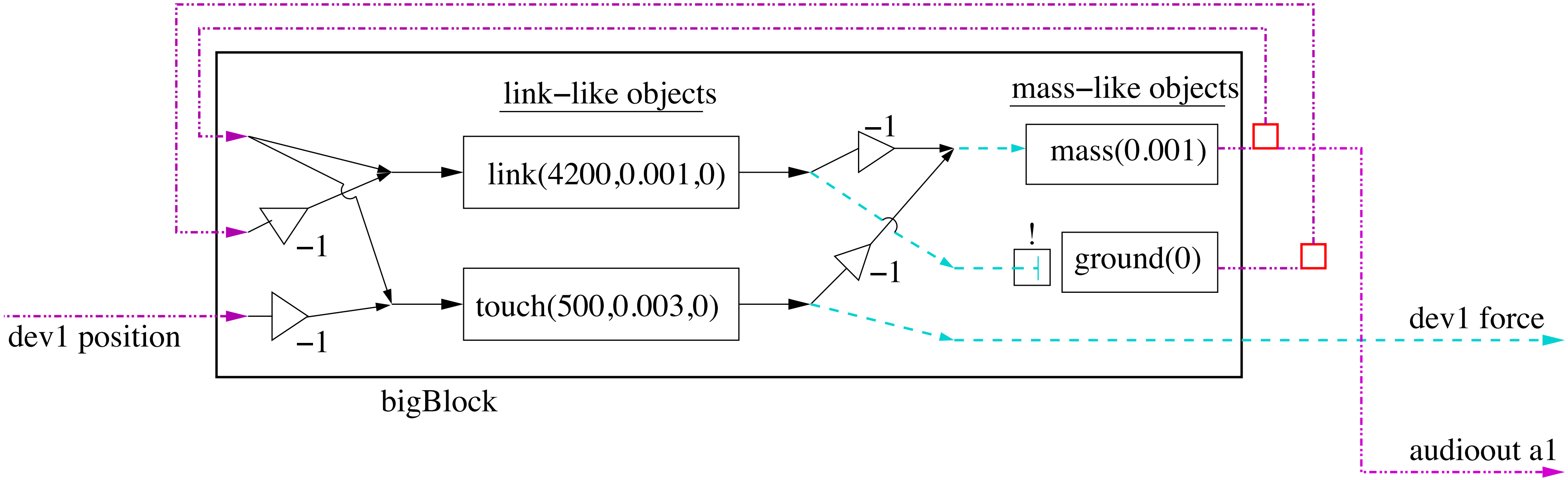
```
mass(0.001),m1,();
ground(0.0),g,();
port( ),dev1,();
```

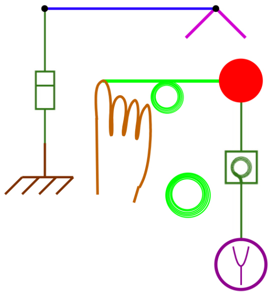


```
link(4200.0,0.001),ll,m1,g,();
touch(1000.0,0.03,0.0),tt,m1,dev1,();

audioout,a1,m1,1000.0;
```

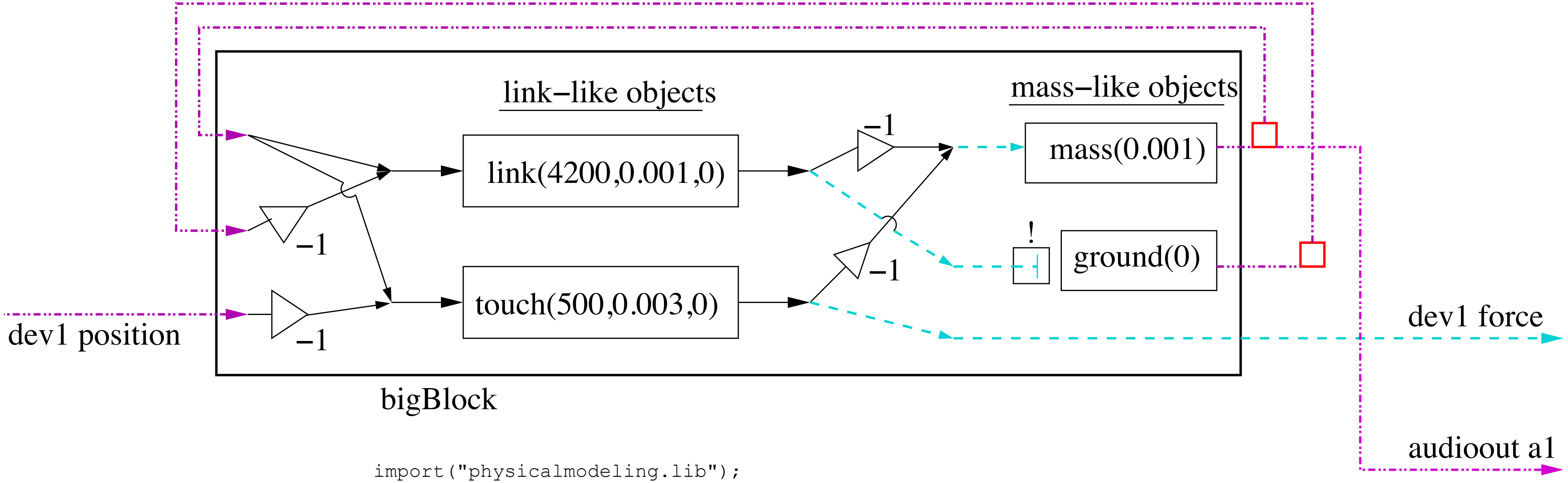


Example Model: Play A Resonator

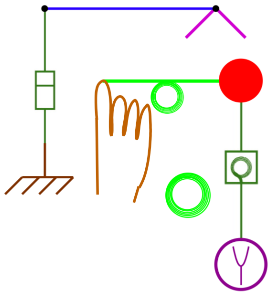




Example Model: Play A Resonator

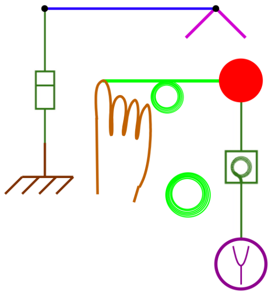


```
import("physicalmodeling.lib");

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)~(_,_):(!,!,_,_);
```

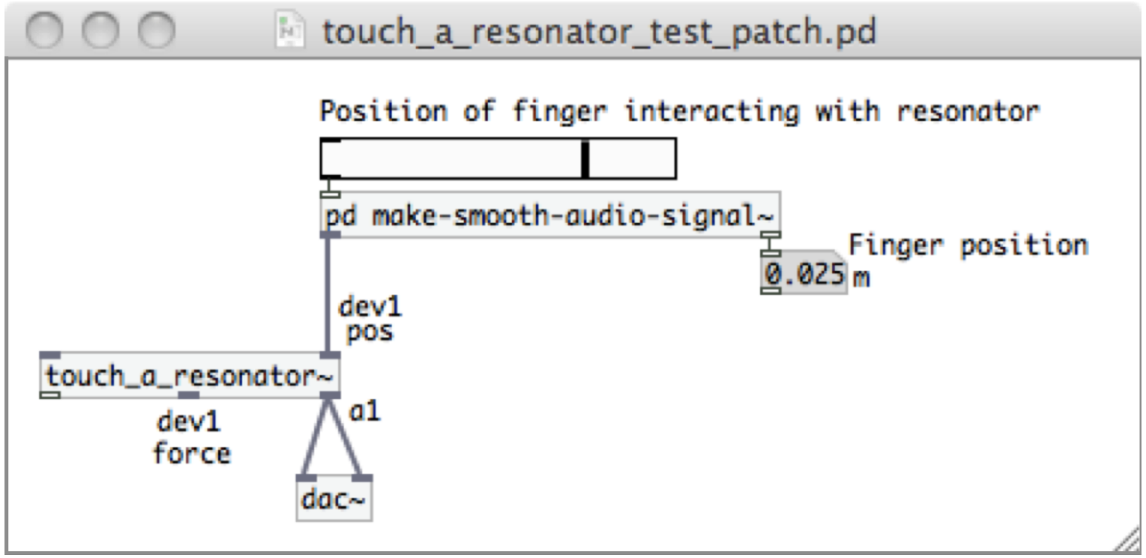


Example Model: Play A Resonator

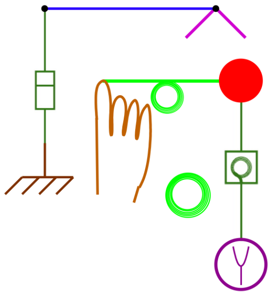
```
import("physicalmodeling.lib");

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)~(_,_):(!,!,_,_);
```



Example Model: Play A Resonator



touch_a_resonator_test_patch.pd
Position of finger interacting with resonator
pd make-smooth-audio-signal~
Finger position
0.025 m
dev1
pos
touch_a_resonator~
a1
dev1
force
dac~



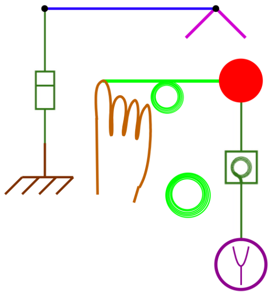
INTRODUCTION

FAUST PROGRAMMING LANGUAGE

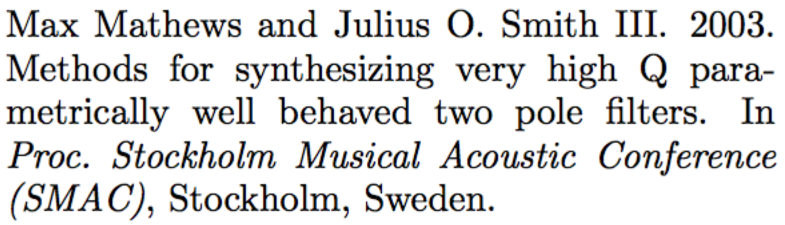
SYNTH-A-MODELER

MORE EXAMPLES

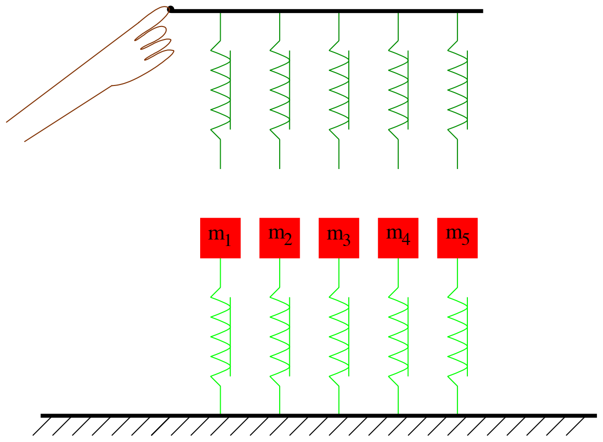
FINAL WORDS

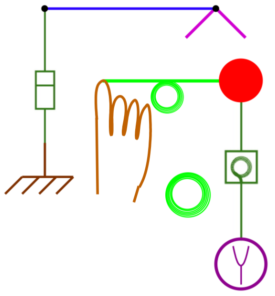


Touching Interpolatable Resonators

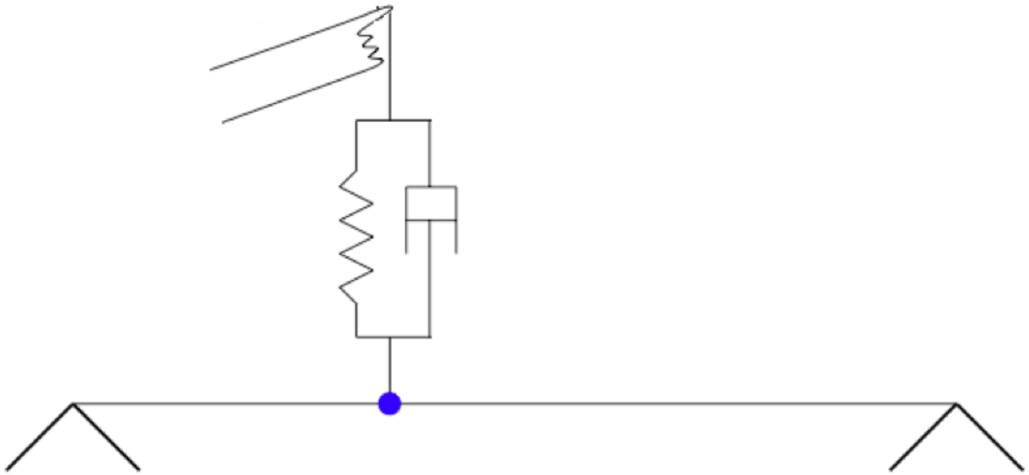


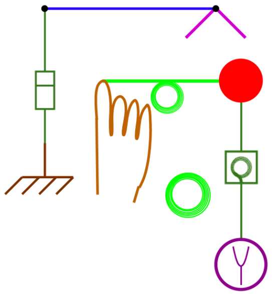
Max Mathews and Julius O. Smith III. 2003. Methods for synthesizing very high Q parametrically well behaved two pole filters. In *Proc. Stockholm Musical Acoustic Conference (SMAC)*, Stockholm, Sweden.





Pluck A String





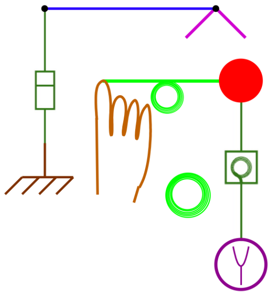
INTRODUCTION

FAUST PROGRAMMING LANGUAGE

SYNTH-A-MODELER

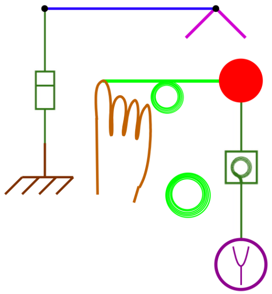
MORE EXAMPLES

FINAL WORDS



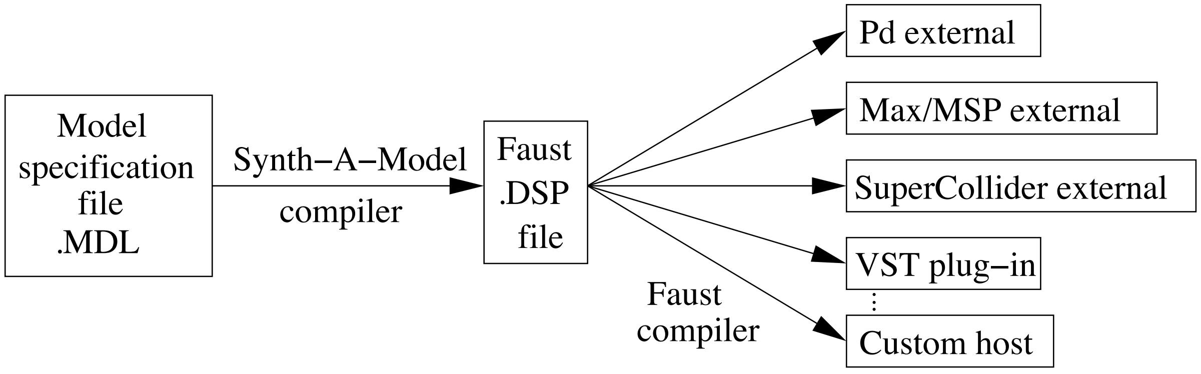
Thanks!

- To Alexandros Kontogeorgakopoulos, Yann Orlarey, and to other researchers in physical modeling who have inspired us very much.



Thanks!

- It's not a synthesizer, it's a *Synth-A-Modeler*


Model specification file .MDL
Synth–A–Model compiler
Faust .DSP file
Faust compiler
Pd external
Max/MSP external
SuperCollider external
VST plug–in
⋮
Custom host