

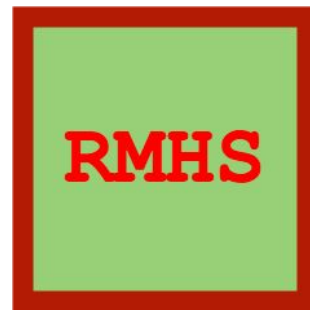
Computer Music Practice Examples

RMHS

Step-by-step process

Joo Won Park

www.joowonpark.net/cmpe



Index

[Step 1. Make S\(iSimple\)](#)

[Step 2. Make L\(overtones\)](#)

[Step 3. Modify S\(iSimple\)](#)

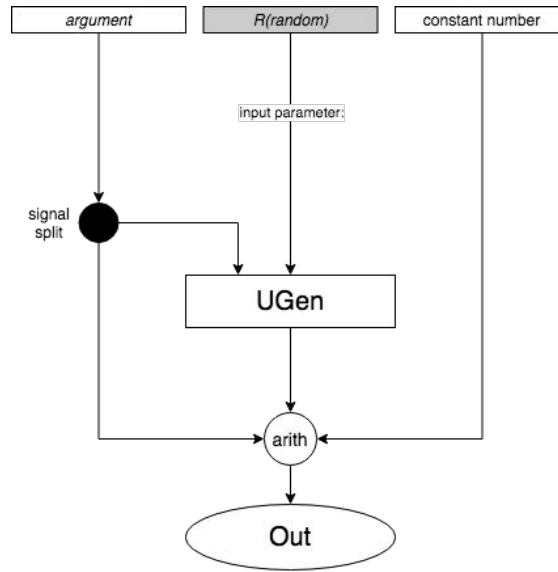
[Step 4. Make F\(~rmhs\)](#)

[Step 5. Make a GUI](#)

[Link to Music and App](#)

[Link to .scd files](#)

Block Diagram Legend



UGen:

Unite Generator.
Processes audio or data

arrow:

Shows the direction of signal
The text in the line (input parameter:) shows the name of the input parameter of an input in the connected UGen

argument:

Controllable arguments
Written in *italics*
Can be a list in [arg,arg,arg] format

constant number:

Discrete numeric value

signal splitter:

Used when one signal is connected to multiple inputs

Arith:

Arithmetics.
Incoming signals are added(+), subtracted(-), multiplied(x), or divided(/).

Out:

Audible audio output

S(synthdef)

S(synthdef) : SynthDef. Includes OSCFunc

L(loop)

L(loop): loops including do{} and Routine

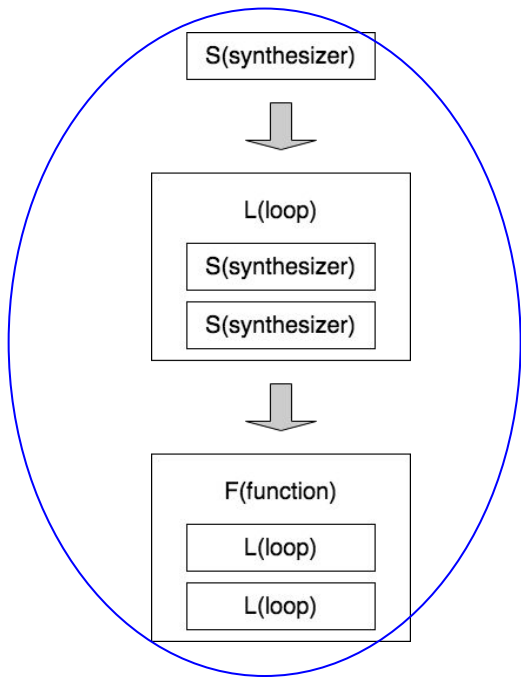
F(function)

F(function): custom function.

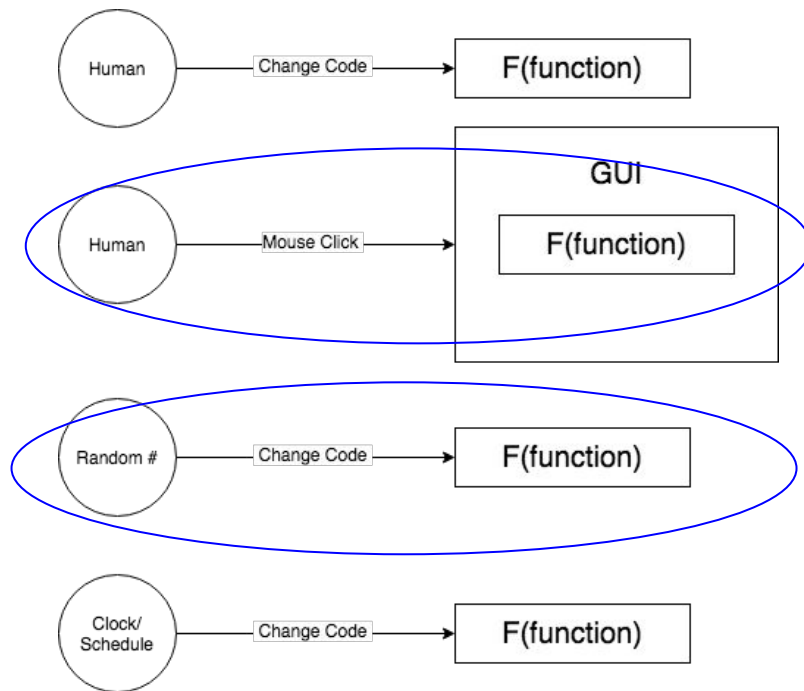
[CMPE - Introduction](#)

Design and Creative Process Overview

Instrument Design Process



Presentation/Performance Methods

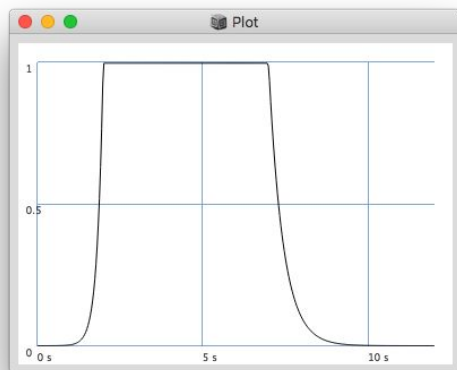


Step 1. Make S(iSimple)

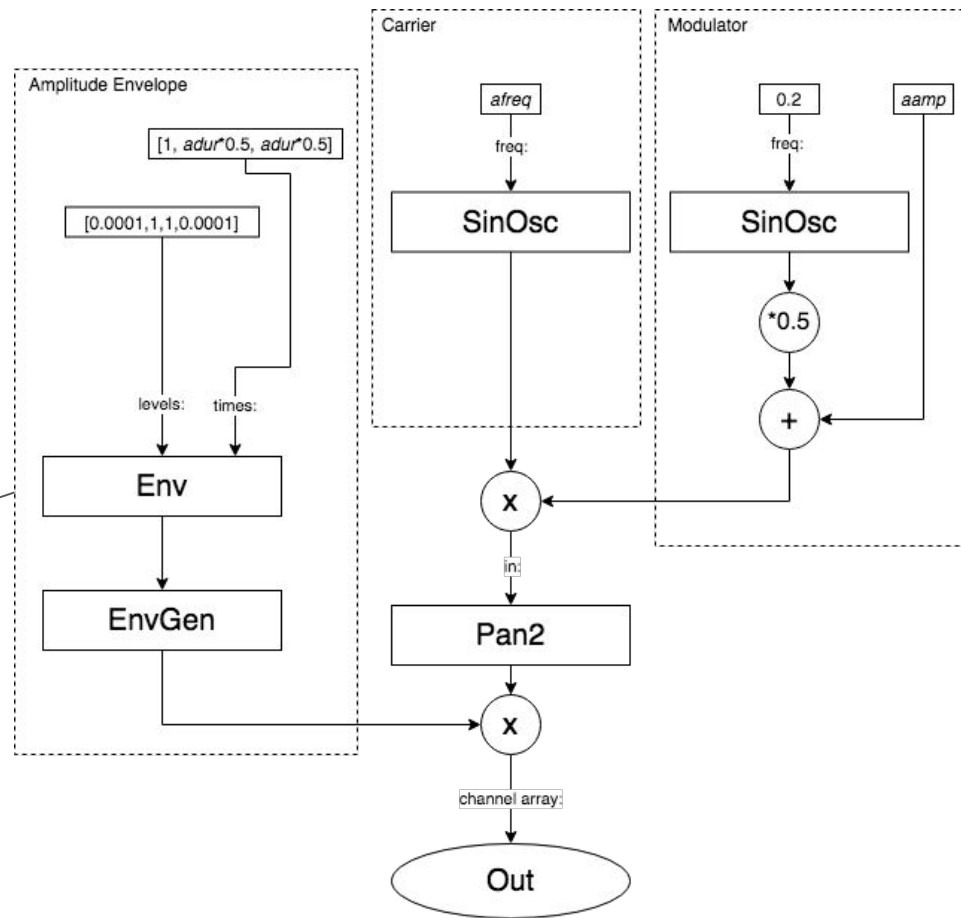
Make a sine wave Synth with a slow amplitude modulation



audio example



Block Diagram



Step 1. Make S(iSimple)

RMHS_Step1.scd

Make a sine wave Synth with
a slow amplitude modulation



audio example

```
SynthDef("iSimple",{  
  arg aamp,afreq,apan,adur;  
  var sound,alter,am,env,mix;  
  sound=SinOsc.ar(afreq);  
  am=SinOsc.ar(0.2)*(aamp*0.5);  
  env=Env.new([0.000001,1,1,0.000001],[1,adur*0.5,0.5*adur],'exp');  
  env=EnvGen.ar(env,doneAction:2);  
  mix=sound*(aamp+am);  
  mix=Pan2.ar(mix,apan)*env;  
  Out.ar(0,mix);  
}).add;  
  
Synth("iSimple",[aamp,0.5,afreq,150,apan,0,adur,10]);
```

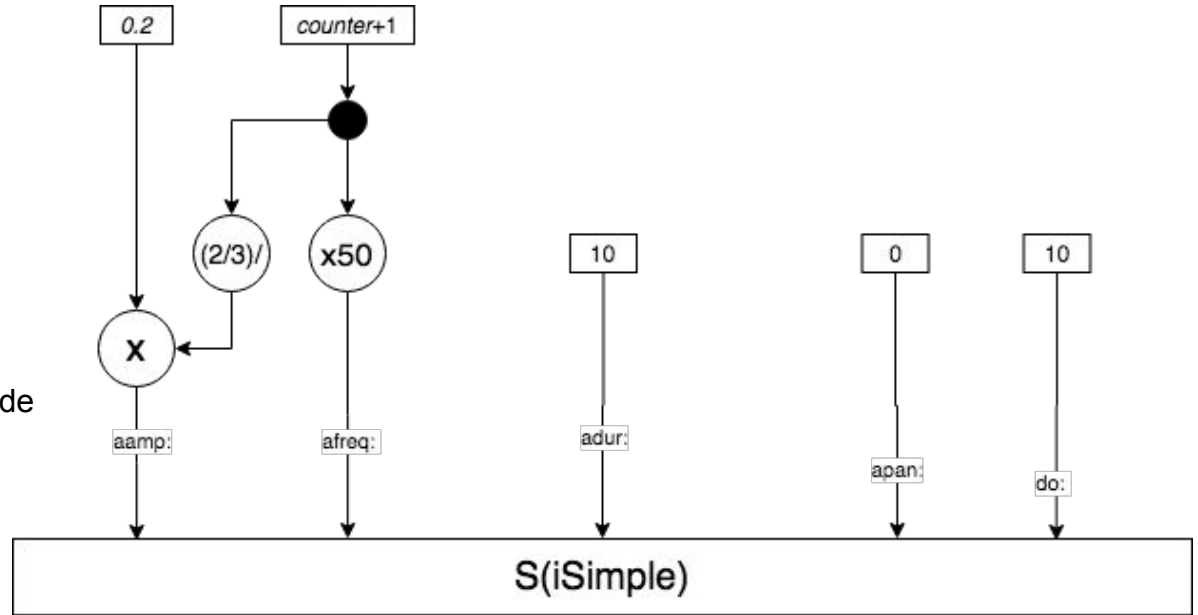
L(overtones)

Step 2. Make L(overtones)

Using S(iSimple), make a loop that creates overtones with decreasing amplitudes



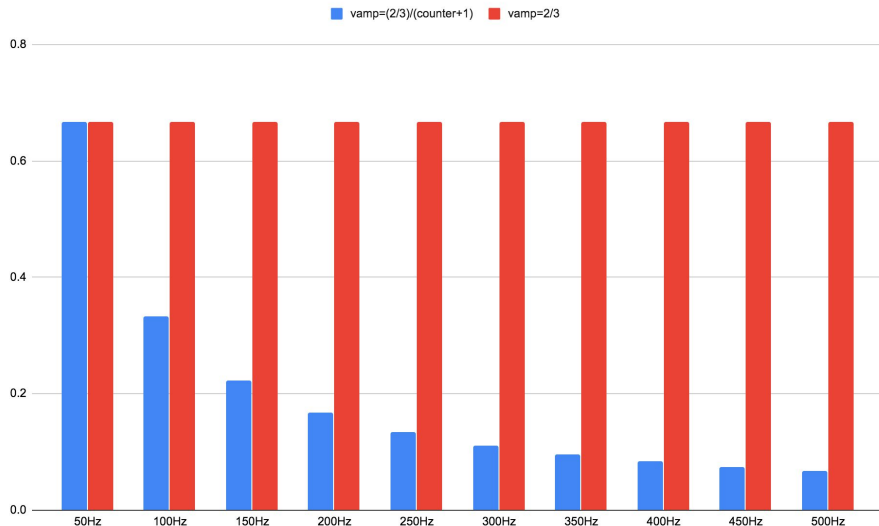
10 partials with decreasing amplitude



Step 2. Make L(overtones)

Using S(iSimple), make a loop that creates overtones with decreasing amplitudes

```
~center=10.do{  
  arg counter;  
  var vamp;  
  vamp=(2/3)/(counter+1);  
  Synth("iSimple",[  
    \aamp,0.2*vamp,  
    \afreq,50*(counter+1),  
    \apan,0,  
    \adur,10  
  ]);  
};
```



partials with decreasing amplitude



partials with constant amplitude

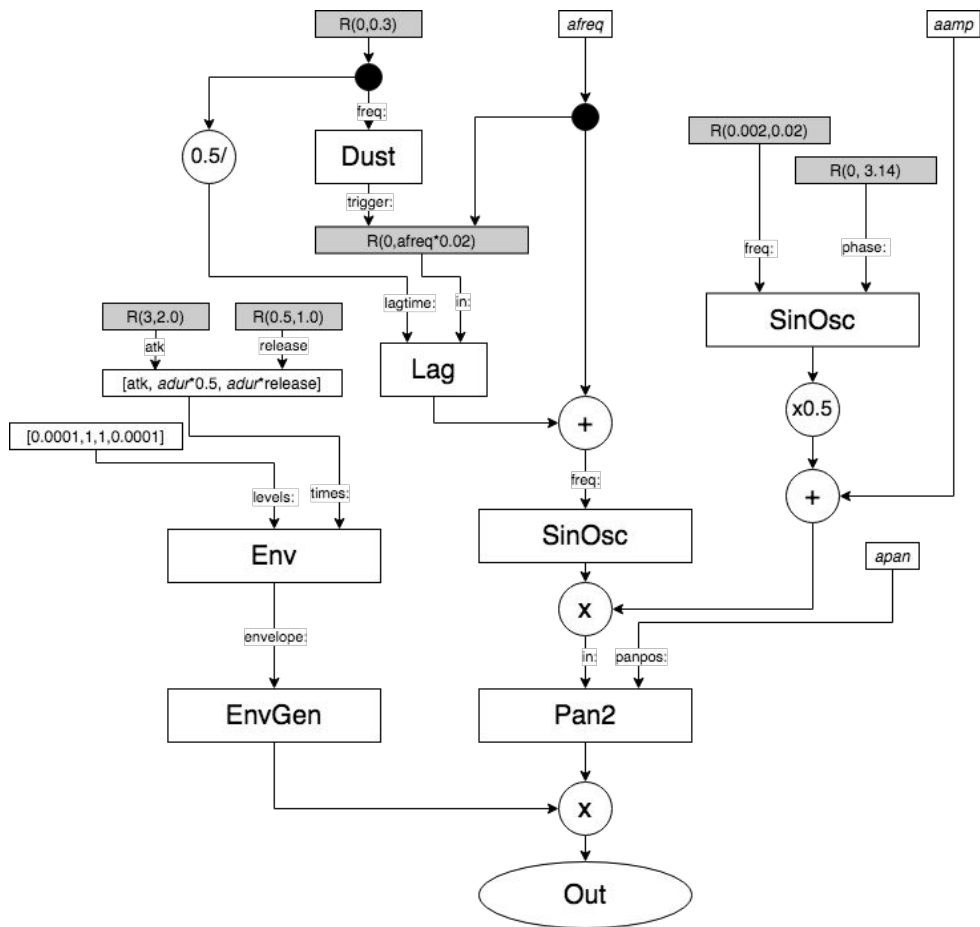
Step 3. Modify S(iSimple)

Randomize parameters in
envelope, carrier, and
modulator



Two S(iSimple) notes

S(iSimple) ver 2



RMHS_Step3.scd

Step 3. Modify S(iSimple)

Randomize parameters in
envelope, carrier, and
modulator

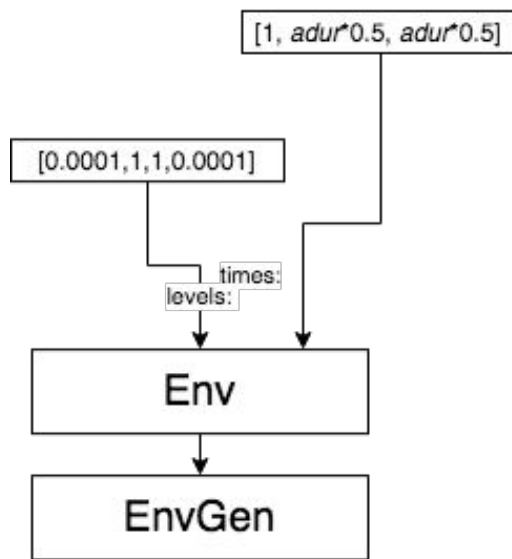
```
SynthDef("iSimple",{  
  arg aamp,afreq,apan,adur;  
  var sound,randtrigg,fm,trigfreq,env,mix;  
  
  trigfreq=Rand(0.0,0.3);  
  randtrigg=TRand.kr(0,afreq*0.02,Dust.kr(trigfreq));  
  randtrigg=Lag.kr(randtrigg,0.5/trigfreq);  
  
  env=Env.new([0.000001,1,1,0.000001],[Rand(3.0,20.0),adur*0.5,Rand(0.5,1.0)*adur],'exp');  
  env=EnvGen.ar(env,doneAction:2);  
  
  sound=SinOsc.ar(afreq+randtrigg);  
  fm=SinOsc.ar(Rand(0.002,0.02),Rand(0,pi))*(aamp*0.5);  
  
  mix=sound*(aamp+fm);  
  mix=Pan2.ar(mix,apan)*env;  
  Out.ar(0,mix);  
}).add;
```



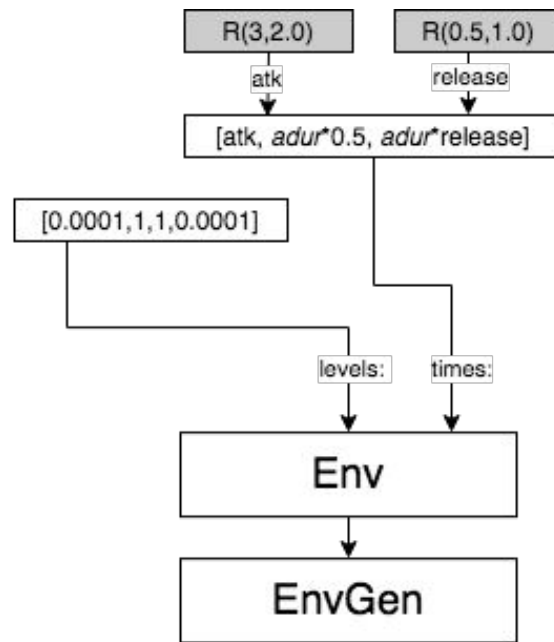
Two S(iSimple) notes

Amplitude Envelope

Step 3. Modify S(iSimple)



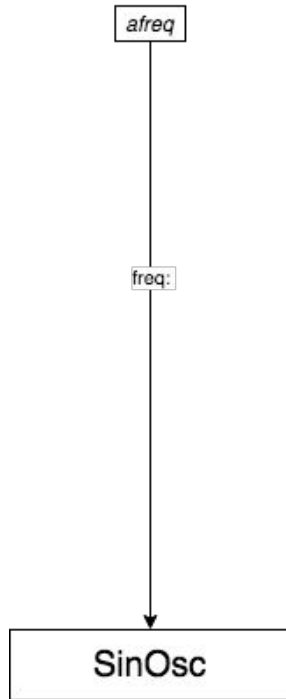
Step 1



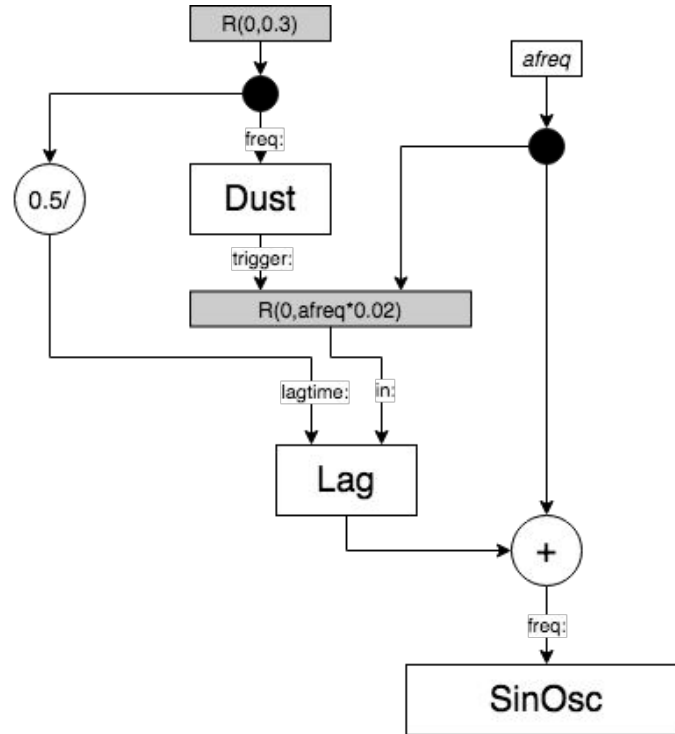
Step 3

Carrier

Step 3. Modify S(iSimple)



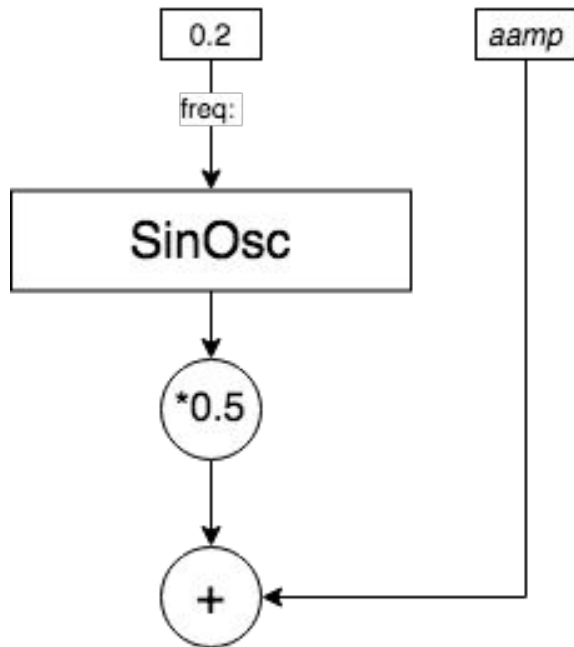
Step 1



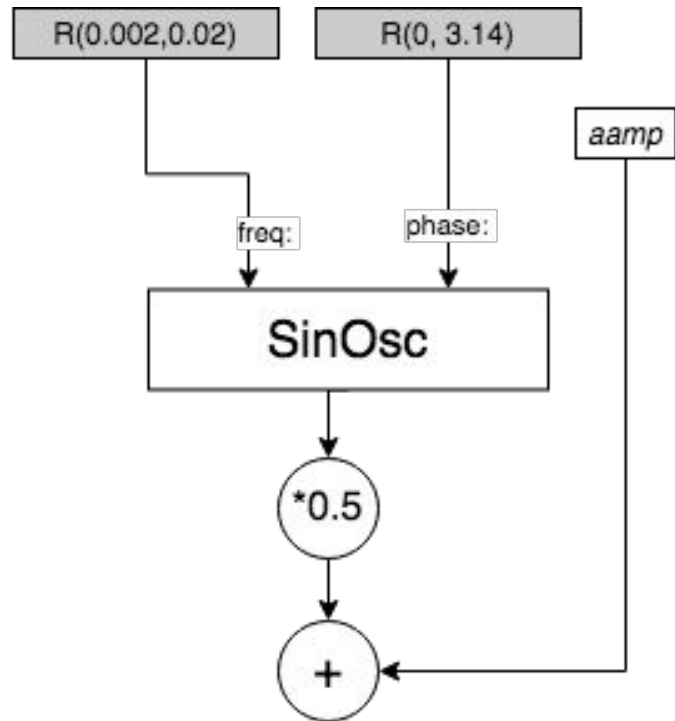
Step 3

Modulator

Step 3. Modify S(iSimple)



Step 1



Step 3

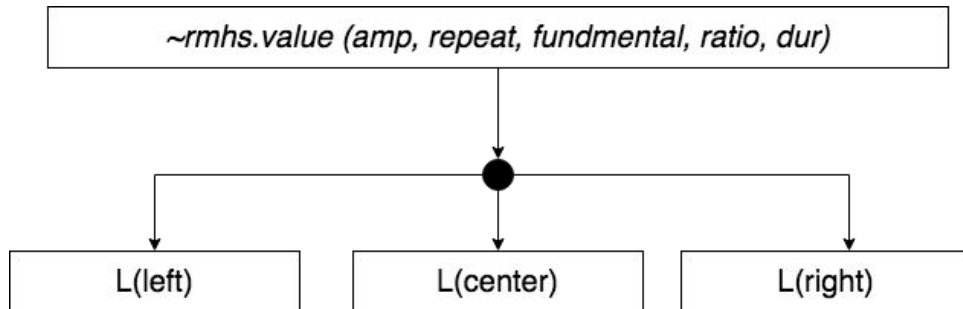
Step 4. Make F(~rmhs)

Make an executable function that creates three variations of L(overtone)

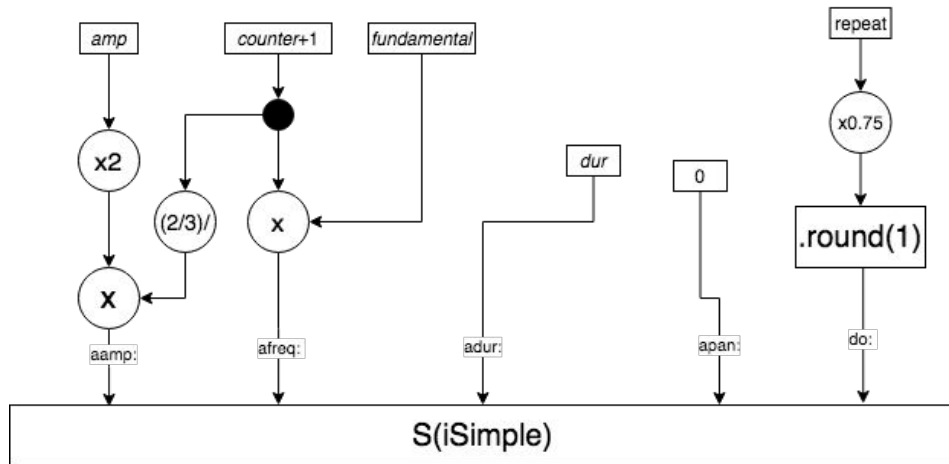


audio example

~rmhs



L(center)



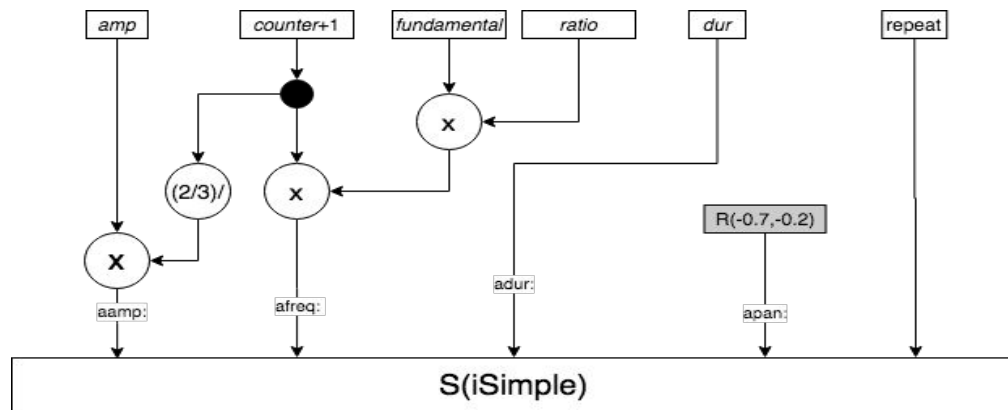
Step 4. Make F(~rmhs)

Make an executable function that creates three variations of L(overtone)

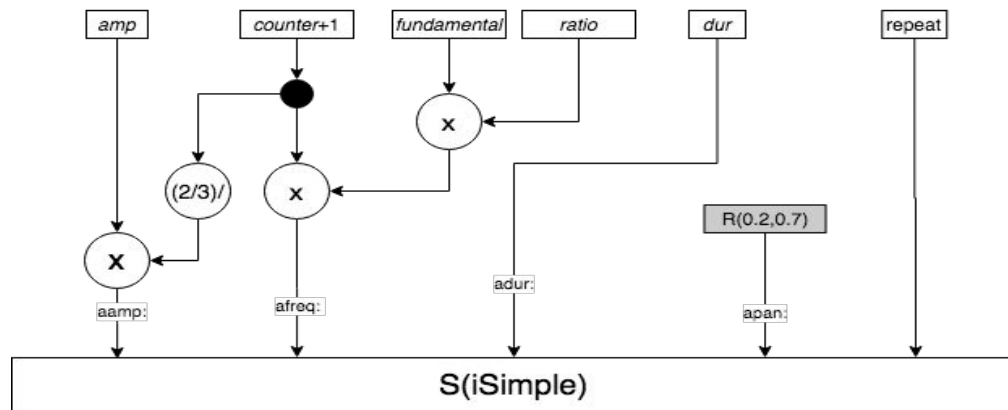


audio example

L(left)



L(right)



Step 4. Make F(rmhs)

Make an executable function that creates three variations of L(overtone), creating a detuned interval



audio example

```
~rmhs={
  arg amp=0.1,repeat=14,fundamental=200,ratio=3/2,dur=100;
  //root
  ((repeat*3/4).round(1)).do{
    arg counter;
    var vamp;
    vamp=(2/3)/(counter+1);
    Synth("iSimple",[\aamp,amp*2*vamp,\afreq,fundamental*(counter+1),\apan,0,\adur,dur]);
  };
  //right
  repeat.do{
    arg counter;
    var vamp;
    vamp=(2/3)/(counter+1);
    Synth("iSimple",[\aamp,amp*vamp,\afreq,fundamental*(counter+1)*ratio,
      \apan,rrand(0.2,0.7),\adur,dur]);
  };
  //left
  repeat.do{
    arg counter;
    var vamp;
    vamp=(2/3)/(counter+1);
    Synth("iSimple",[\aamp,amp*vamp,\afreq,fundamental*(counter+1)*ratio,
      \apan,rrand(-0.7,-0.2),\adur,dur]);
  };
};
//~rmhs.value( amp,repeat,fundamental,ratio,dur);
~rmhs.value(0.45,10,50,4/3,20)
```

Step 4. Make F(rmhs)



audio example

RMHS Frequencies Playing 4:3 Ratio (Hz)

Count+1	Pitch 1	Center	Left	Right	Pitch 2
1	50	50.117	66.713	67.668	66.667
2	100	100.300	134.972	135.381	133.333
3	150	151.286	200.601	200.441	200.000
4	200	202.740	271.814	268.520	266.667
5	250	253.760	336.288	339.604	333.333
6	300	304.726	405.955	401.749	400.000
7	350	352.208	467.678	473.150	466.667
8	400	400.016	542.544	541.526	533.333
9	450	453.432	604.497	602.179	600.000
10	500	504.454	669.586	676.797	666.667

Step 5. Make a GUI

The screenshot shows a macOS-style window titled "RMHS" containing a GUI for a sound synthesis application. The GUI has a title bar with red, yellow, and green window control buttons. The main content area includes a "Start (with fade-in)" button at the top. Below it is a checkbox labeled "record at start", which is linked by an arrow to the text "s.record" on the left. Further down are several numeric input fields: "loudness (range: 0.0-1.0)" with a value of "0.5", "number of harmonics (range: 1-20)" with a value of "12", "fundamental frequency (Hz)" with a value of "100", "interval (half steps, range: 0-24)" with a value of "5", and "duration (seconds, excluding fade-in time)" with a value of "100". An arrow points from the "randomize parameters" text on the left to the "randomize" checkbox at the bottom. On the right side, a list of code snippets is shown, with arrows pointing from the GUI elements to them: "~rmhs.value(" from the "Start" button, "amp," from the "loudness" field, "repeat," from the "number of harmonics" field, "fundamental," from the "fundamental frequency" field, "ratio," from the "interval" field, "dur," from the "duration" field, and ")" from the "randomize" checkbox. At the bottom of the window, it says "by Joo Won Park (www.joowonpark.net)".

s.record

randomize parameters

Start (with fade-in)

☐ record at start

loudness (range: 0.0-1.0)

0.5

number of harmonics (range: 1-20)

12

fundamental frequency (Hz)

100

interval (half steps, range: 0-24)

5

duration (seconds, excluding fade-in time)

100

☐ randomize

by Joo Won Park (www.joowonpark.net)

~rmhs.value(
amp,
repeat,
fundamental,
ratio,
dur,
)

Inspirations

Music on a Long Thin Wire

By Alvin Lucier

What's Next & More Music Examples

ISJS: Granular Processor

Overundertone

Contact joowon@joowonpark.net if you have questions or see errors.

