Definition of Done: Fun.

Code you can play!

Krista Campbell

Architect, Coach, Advocate

May 5th 2023

Insight | Digital Innovation

What did we just experience?

Code You Can Play

Let’s talk about software that’s not about delivering value but a sensory experience.

Transcend language.

Build. Tinker. Play. Repeat.

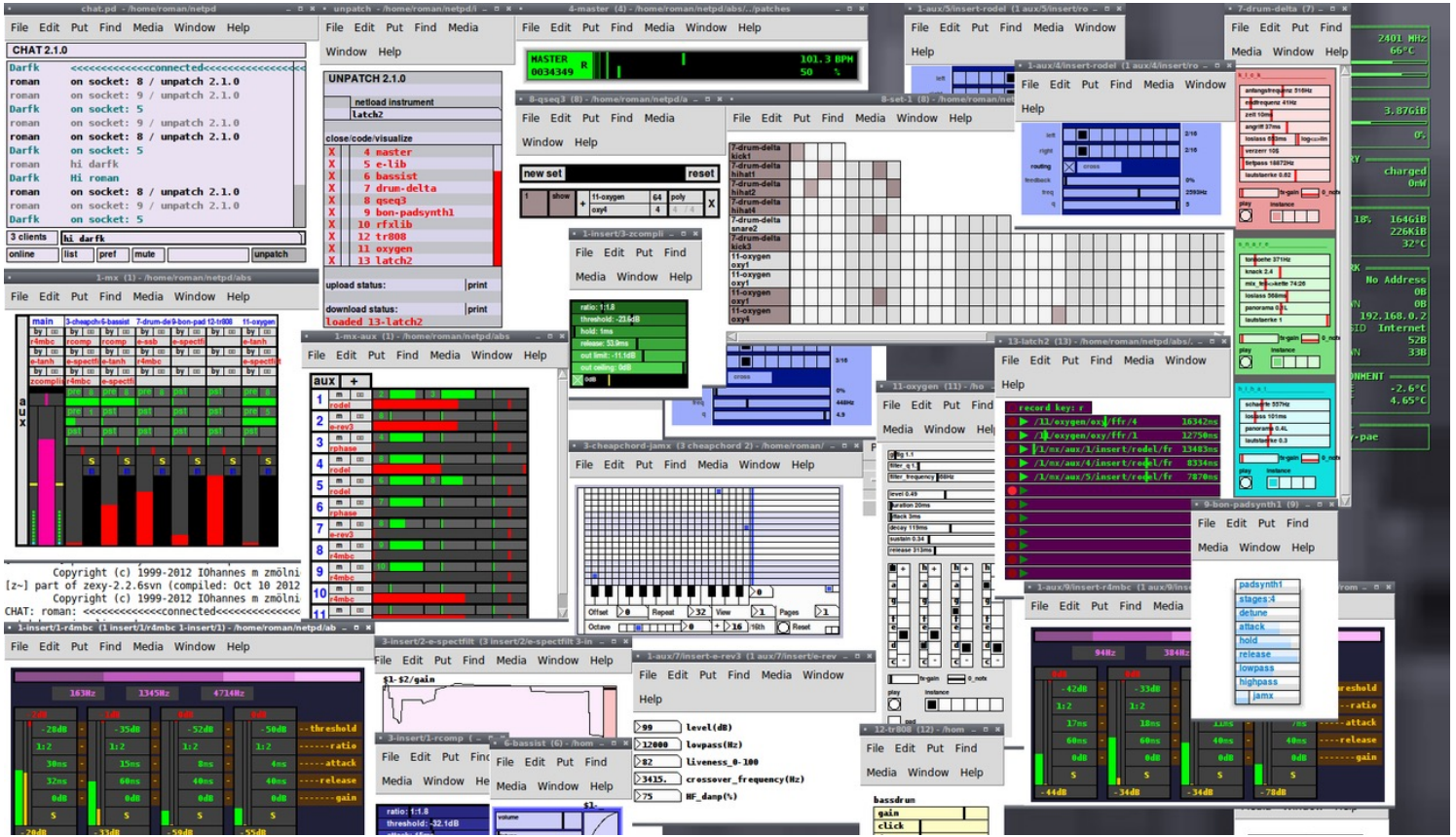
Code You Can Play

The development process itself is intended to be about playing!

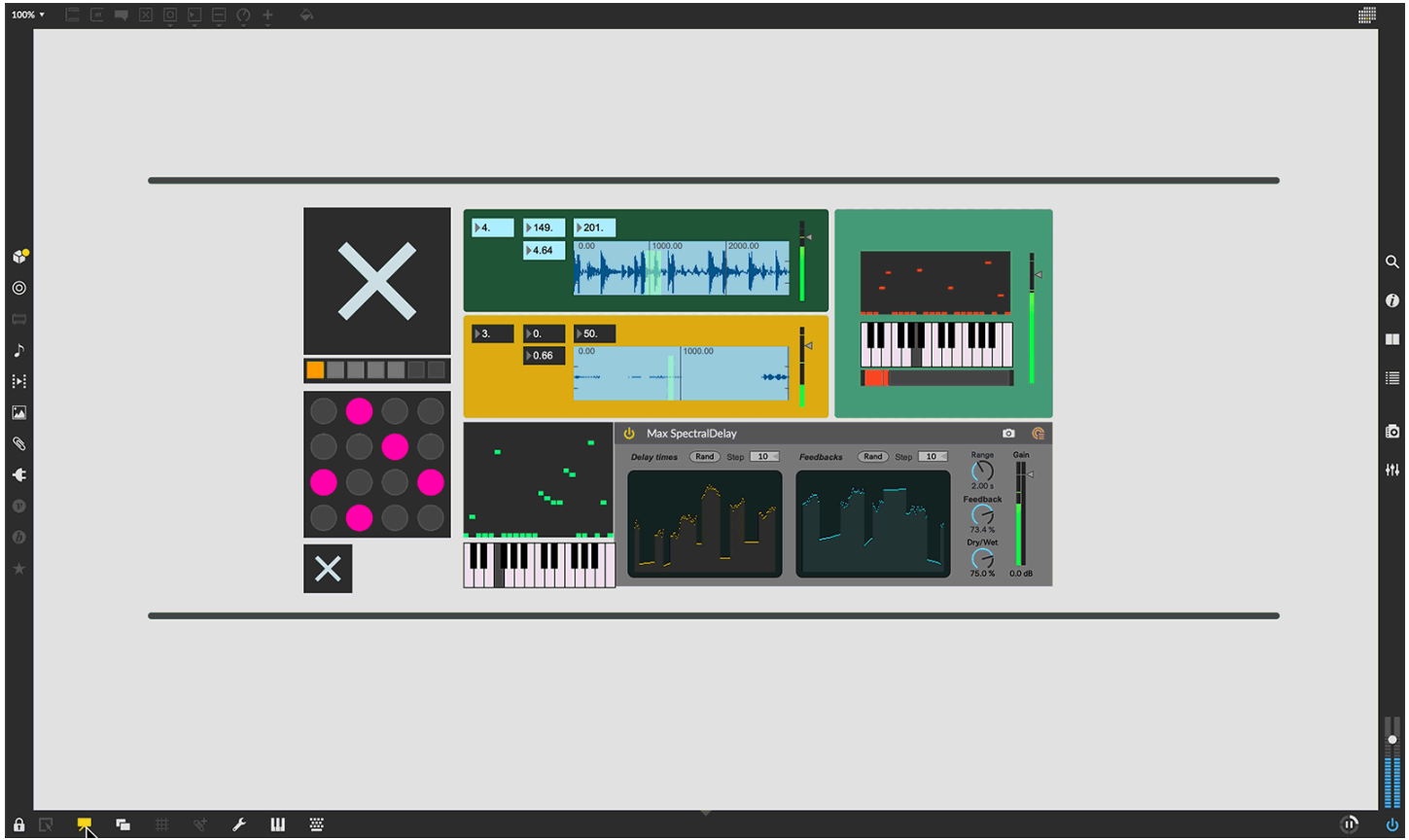
Using two patcher environments…..

Build. Tinker. Play. Repeat.

The Open Source PureData "PD"



Its shinier commercial cousin: Cycling 74's Max



Code You Can Play

In the process we will cover some basic audio synthesis and signal processing.

And look under the hood a bit too.
(This is a developer conference after all)

It can actually give you insight into your day job too.

Build. Tinker. Play. Repeat.

Agenda

1. PD & Max
2. How it works
3. Tinker!
4. Play!

Krista Campbell

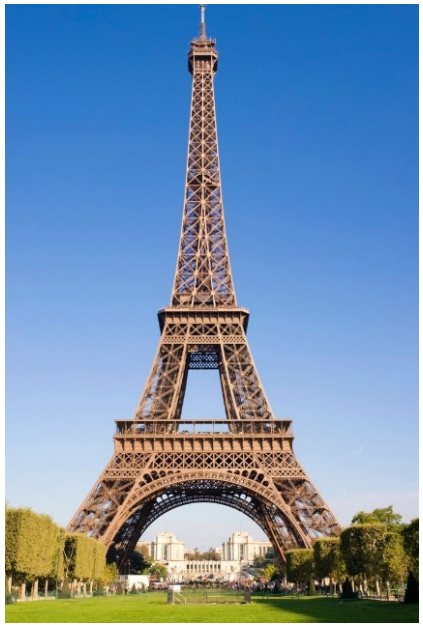
Architect. Technical Coach. Advocate.
Musician. Mom. Transwoman. Human.

1st Program: BASIC on C64 to make sound effects. (4th grade)
Studied composition and computer science
Been doing this way too long!
Software Engineer, Tech Lead, Architect, Coach.

PD & Max

Real time signal processing you can play

PD & Max



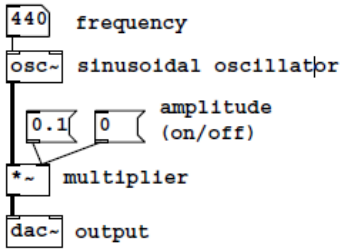
- PureData and Max both originally written by Miller Puckette starting in 1985 at IRCAM in Paris
- Max was initially MIDI. Then added signal processing: MSP.
- Max was bought by fellow developer David Zicarelli who commercialized it and started the company Cycling 74.
- PureData has always remained open source.
- Idea behind PureData is that it allows for abstract data structures to interface with control data (and people) and real time signal processing (sound and visual). In practice it's mostly used for audio processing.
- Both also have real time video processing components too: PD's GEM and Max's Jitter.
- Share the same audio engine and other source code.

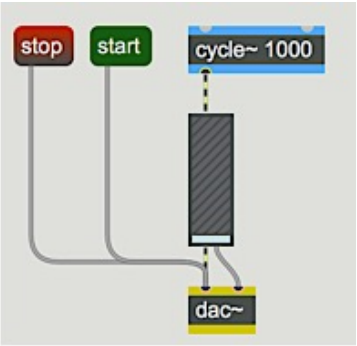
PD

- PureData is free
- And it is VAST
- PureData Download and InfoHub: http://puredata.info/
- But it is extremely well documented and a great way to learn is to read MSP's book
- Miller Puckette's Companion Book (best source to get started):http://msp.ucsd.edu/techniques.htm
- Written in C/C++ and is extremely portable

Max

- Max is equally vast and polished as well
- Basis for a LOT of software, including Ableton Live, who eventually bought Cycling 74 and ported a version of Max directly into Live.
- Used to create VST/AU Plugins & Virtual Synths/Samplers
- Much prettier :)
- Used by devs and musicians
- Newest components are Gen which is incredibly optimized and can be used to generate high-performing C/C++ & RNBO which can export plugins, hardware plugins, and C/C++ source
- Runs on Windows and Mac
- Month trial, then $10 month subscription.
- Available for download here: https://cycling74.com/downloads





PD & Max

Okay but what is a “patcher”

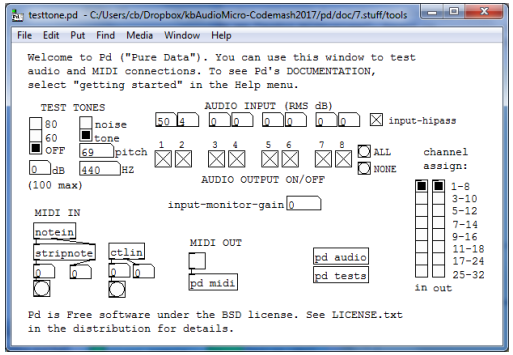
- It’s a type of dataflow language – a directed graph where nodes are computations and data flows along the edges. (aka stream processing or reactive programming)
- Is essentially a visual programming language of objects represented by boxes connected by lines
- The boxes represent variables or code
- The lines represent variable assignment or function calls or audio signals
- Some event triggers downstream objects (bang, control signal, timer)

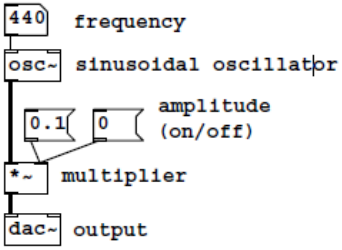
PD & Max

Okay but what is a “patcher”

Originally it was conceptually based around the idea of old school modular synthesizers like the Moog but grew far beyond that.

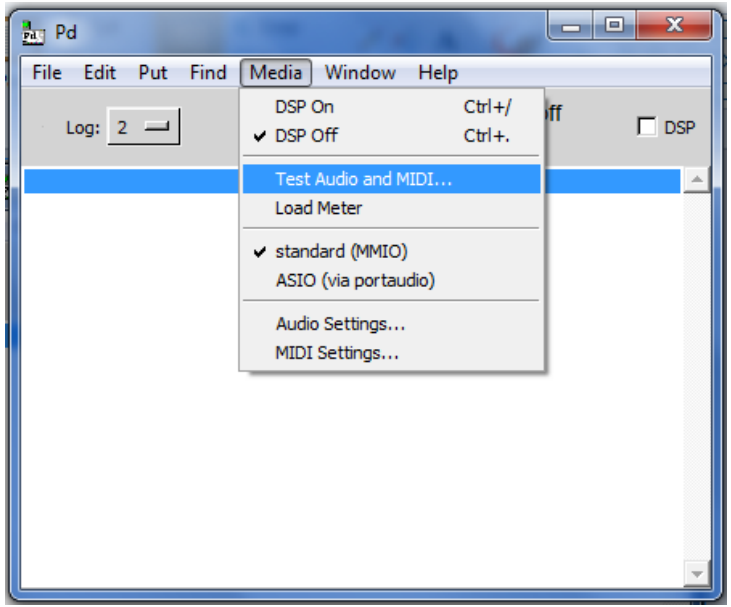
Tinker.

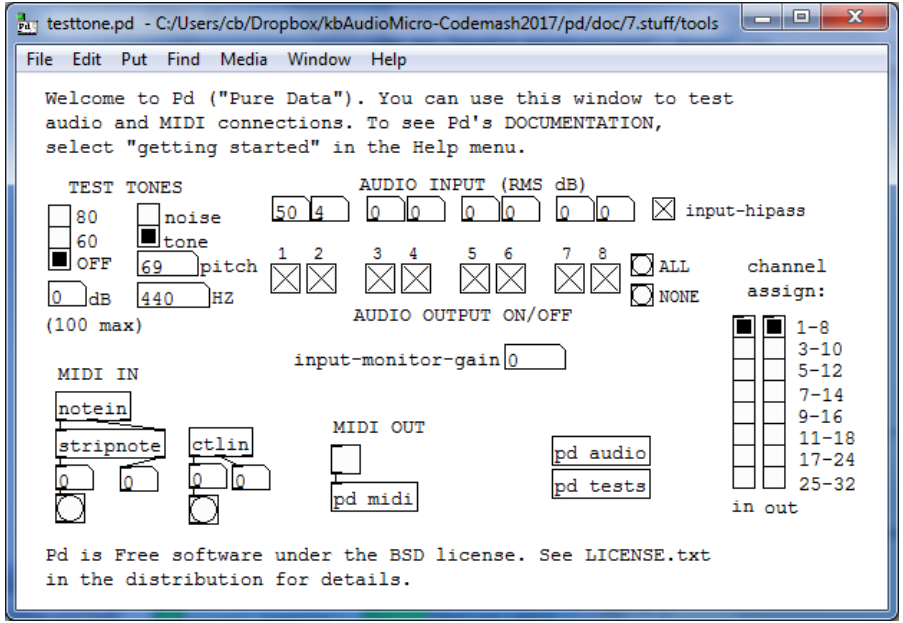

testtone.pd - C:/Users/cb/Dropbox/kbAudioMicro-Codemash2017/pd/doc/7.stuff/tools
File Edit Put Find Media Window Help
Welcome to Pd ("Pure Data"). You can use this window to test
audio and MIDI connections. To see Pd's DOCUMENTATION,
select "getting started" in the Help menu.
TEST TONES
80
60
OFF
noise
tone
69
pitch
0
dB
440
HZ
(100 max)
AUDIO INPUT (RMS dB)
50
input-hipass
1 2
3 4
5 6
7 8
ALL
NONE
AUDIO OUTPUT ON/OFF
channel
assign:
1-8
3-10
5-12
7-14
9-16
11-18
17-24
25-32
in out
input-monitor-gain
0
MIDI IN
notein
stripnote
ctlin
MIDI OUT
pd audio
pd tests
pd midi
Pd is Free software under the BSD license. See LICENSE.txt
in the distribution for details.


440
frequency
osc~
sinusoidal oscillator
0.1
0
amplitude
(on/off)
*~
multiplier
dac~
output

Tinker

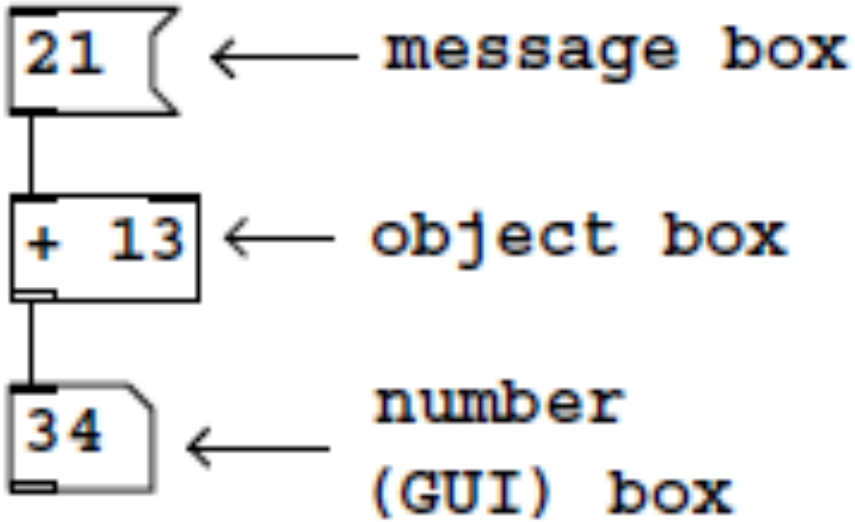
PD Message Window, Config, and Test





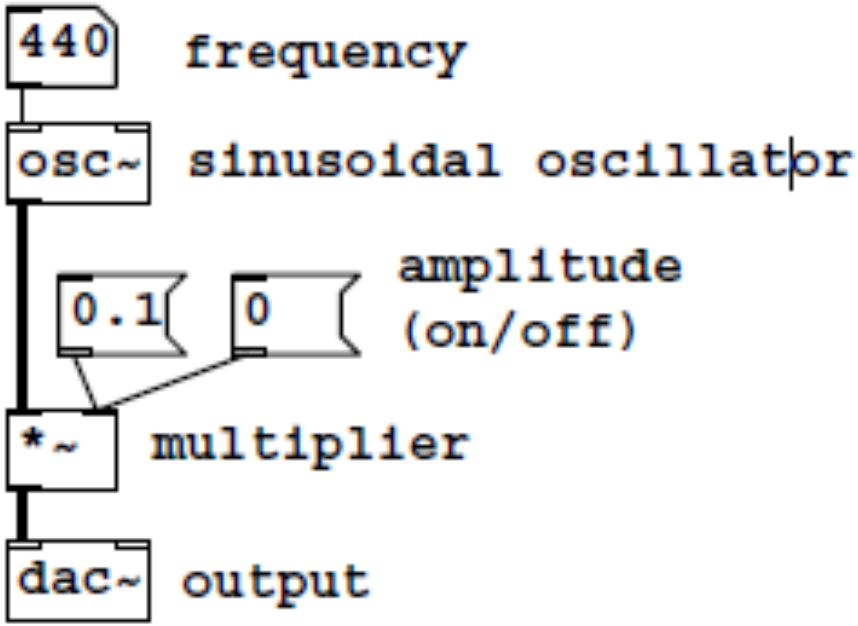
Test and Start Your Engines!

PD Essentials: Messages, Objects, Numbers

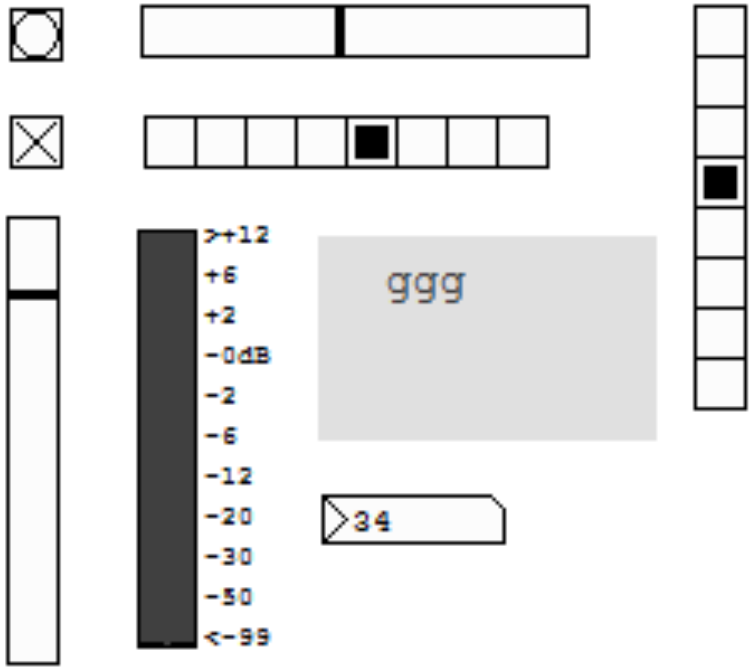


pd~ Tinker

PD Essentials: Audio Processing

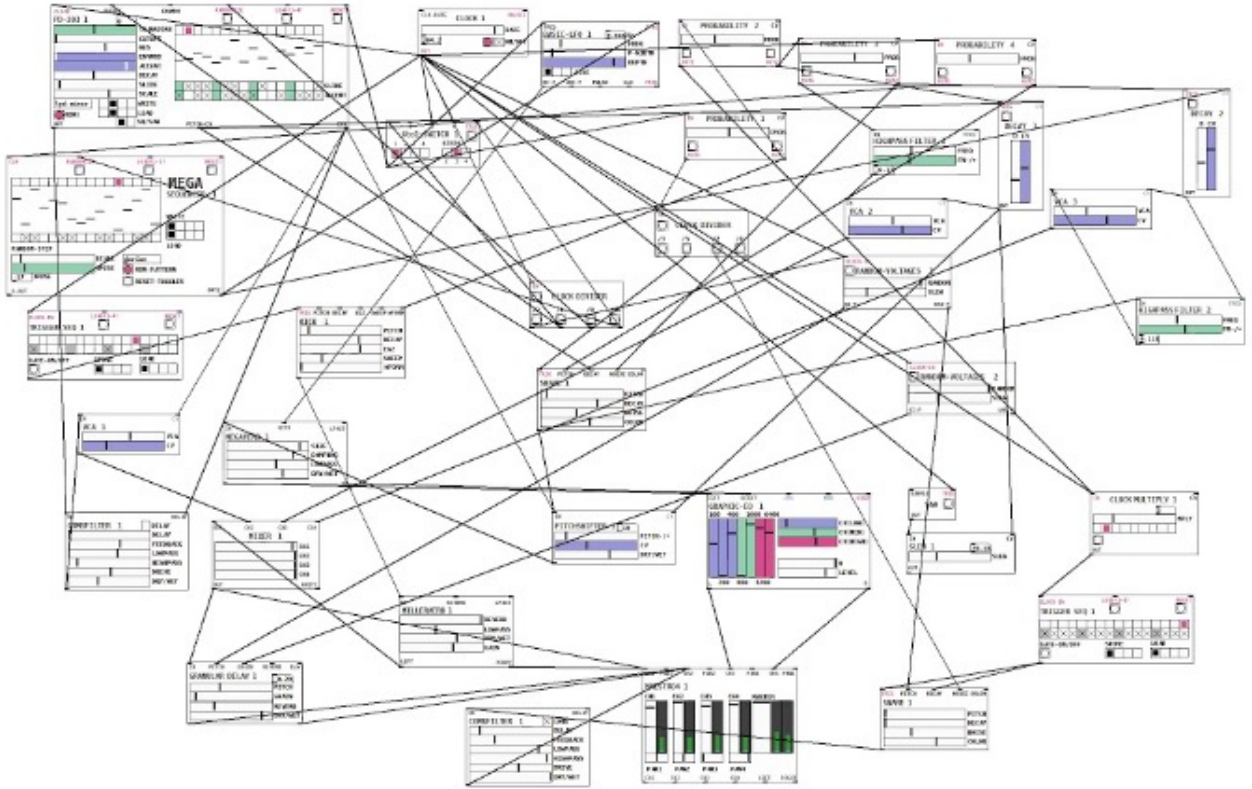


PD Essentials: GUI Objects



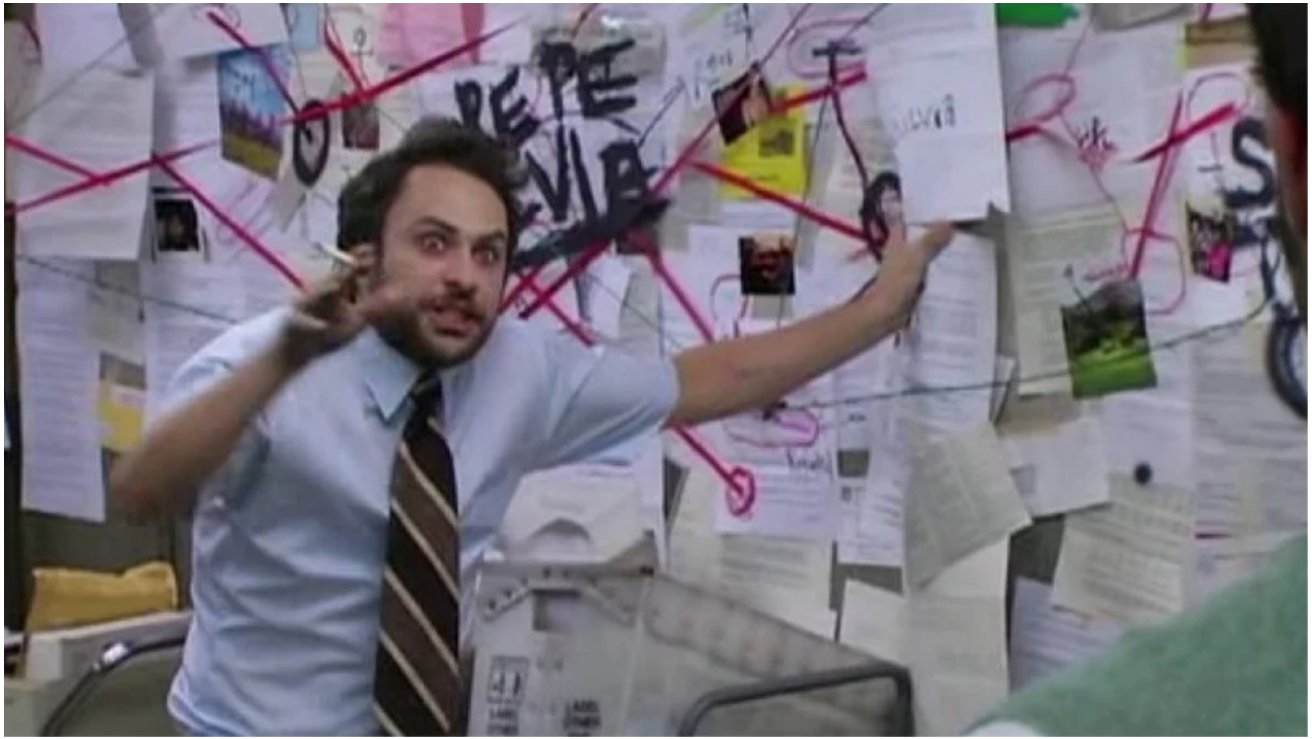
Tinker

Patchers can easily start to look a little crazy….

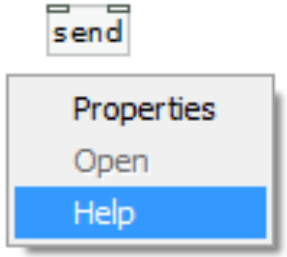


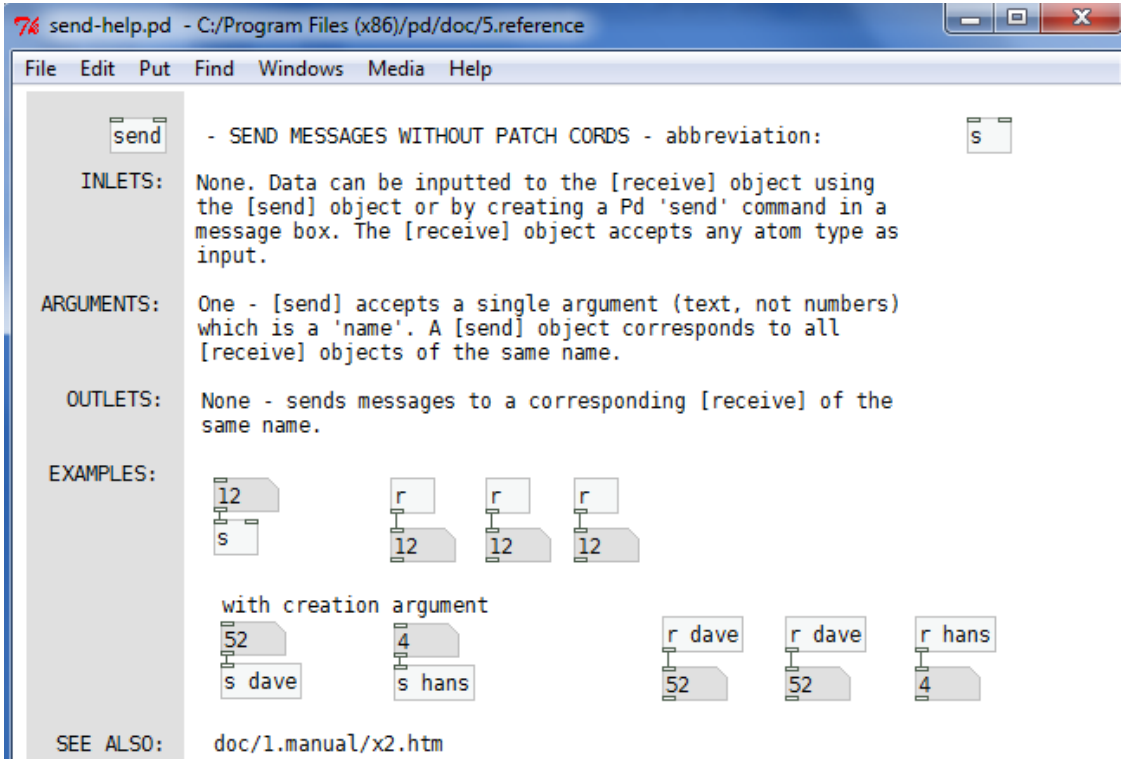
Tinker

Patchers can easily start to look a little crazy….



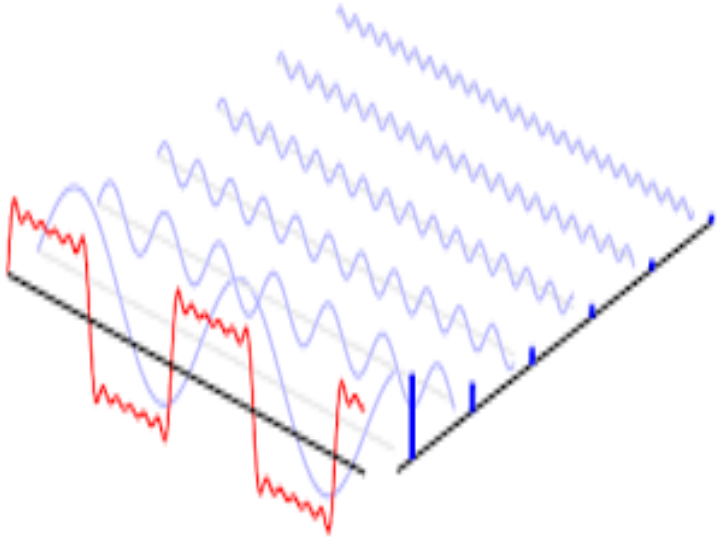
PD Essentials: Help & Send/Receive





How it Works

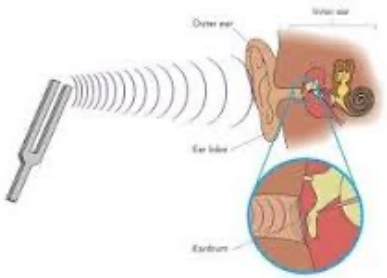
Looking under the hood

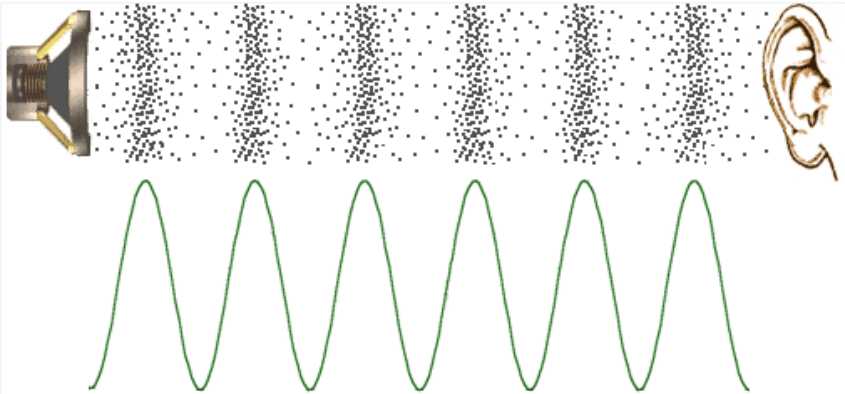


How it Works

Some essential digital audio theory

- Sound is caused by the movement of air molecules, the vibration of sound waves, moving our eardrum



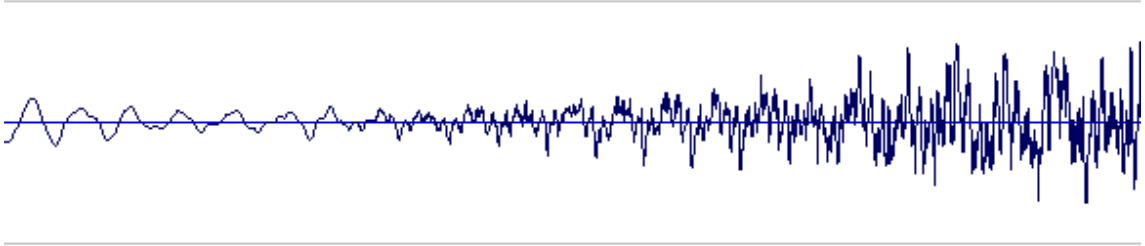


- How about from a speaker?
- Voltage charges magnet, moving the speaker surface, moving air, moving our eardrum

How it Works

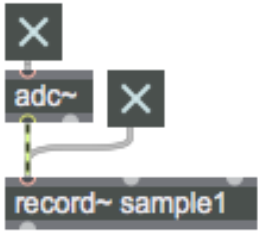
Some more essential digital audio theory

- Sound waves we are used to seeing is that voltage differential, causing that speaker head to move forward or backward
- Distance from the center = loudness
- How close waves are = frequency

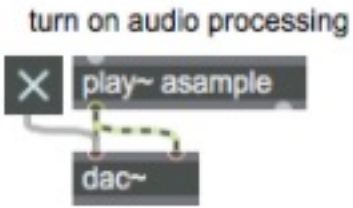


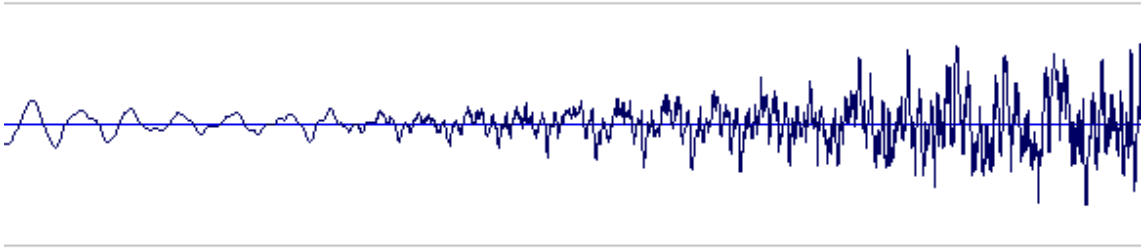
How it Works

In PD and Max



- Input signals (e.g. microphone input) are translated from voltages into numbers by a device called an Analog to Digital Converter, the adc~ object.
- Output signals (e.g. speaker or headphone out) are translated from numbers into voltages by a device called a Digital to Analog converter, the dac~ object.





How it Works

Simple (non-signal) Objects

- PD & Max simple (non-signal) objects are implemented in C/C++ by implementing functions which respond to messages and registering those functions with the PD/Max engine.
- {name}_setup registers the constructor (canonically {name}_new) and message handlers
- {name}_new instantiates the object, creates additional inputs and outlets
- The engine then calls the appropriate message handler when data is sent to the object

How it Works

Simple (non-signal) Object: Counter

```
#include "m_pd.h"

static t_class *counter_class;

typedef struct _counter {
  t_object  x_obj;
  int i_count;
} t_counter;

void counter_bang(t_counter *x)
{
  t_float f=x->i_count;
  x->i_count++;
  outlet_float(x->x_obj.ob_outlet, f);
}
```

```
void *counter_new(t_floatarg f)
{
  t_counter *x = (t_counter *)pd_new(counter_class);

  x->i_count=f;
  outlet_new(&x->x_obj, &s_float);

  return (void *)x;
}

void counter_setup(void) {
  counter_class = class_new(gensym("counter"),
        (t_newmethod)counter_new,
        0, sizeof(t_counter),
        CLASS_DEFAULT,
        A_DEFFLOAT, 0);

  class_addbang(counter_class, counter_bang);
}
```

How it Works

Audio (signal~) Objects

- PD & Max signal objects (suffix of ~) are implemented just like non-signal objects, but includes 2 additional functions:
- {name}_dsp which adds the object's perform method to the DSP tree by registering the object's {name}_perform method using dsp_add. {name}_dsp is called everytime the DSP tree is built, which happens every time the dsp engine is turned on.
- {name}_perform which is called repeatedly and VERY fast while DSP is on. Remember it has to process 44,100 samples every second to maintain quality sound!
- It handles this by being passed an array of arrays, each representing audio of the audio inputs and audio outputs. All of which must be the same size (the signal buffer size).
- This is how it is able to process very quickly – perform is not called for every sample, only once for the size of the buffer size. It is called sequentially through the tree of audio objects, allowing each object to do whatever it needs to do to each of the signal vectors.

How it Works

Signal Object: xfade~.c

```
#include "m_pd.h"

static t_class *xfade_tilde_class;

typedef struct _xfade_tilde {
  t_object x_obj;
  t_float x_pan;
  t_float f;

  t_inlet *x_in2;
  t_inlet *x_in3;
  t_outlet*x_out;
} t_xfade_tilde;

t_int *xfade_tilde_perform(t_int *w)
{
  t_xfade_tilde *x = (t_xfade_tilde *)(w[1]);
  t_sample    *in1 =      (t_sample *)(w[2]);
  t_sample    *in2 =      (t_sample *)(w[3]);
  t_sample    *out =      (t_sample *)(w[4]);
  int             n =             (int)(w[5]);
  t_sample pan = (x->x_pan<0)?0.0:(x-
>x_pan>1)?1.0:x->x_pan;

  while (n--) *out++ = (*in1++)*(1-
pan)+(*in2++)*pan;
  return (w+6);
}

void xfade_tilde_dsp(t_xfade_tilde *x, t_signal
**sp)
{
  dsp_add(xfade_tilde_perform, 5, x,
          sp[0]->s_vec, sp[1]->s_vec, sp[2]-
>s_vec, sp[0]->s_n);
}

void xfade_tilde_free(t_xfade_tilde *x)
{
  inlet_free(x->x_in2);
  inlet_free(x->x_in3);
  outlet_free(x->x_out);
}

void *xfade_tilde_new(t_floatarg f)
{
  t_xfade_tilde *x = (t_xfade_tilde
*)pd_new(xfade_tilde_class);

  x->x_pan = f;
```

How it Works

Signal Object: xfade~.c (continued)

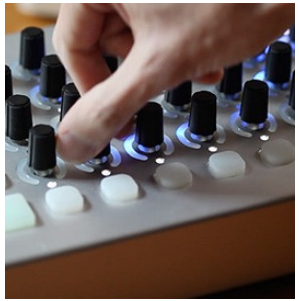
```
x->x_in2=inlet_new(&x->x_obj, &x->x_obj.ob_pd,
&s_signal, &s_signal);
  x->x_in3=floatinlet_new (&x->x_obj, &x-
>x_pan);
  x->x_out=outlet_new(&x->x_obj, &s_signal);

  return (void *)x;
}

void xfade_tilde_setup(void) {
  xfade_tilde_class =
class_new(gensym("xfade~"),
        (t_newmethod)xfade_tilde_new,
        0, sizeof(t_xfade_tilde),
        CLASS_DEFAULT,
        A_DEFFLOAT, 0);

  class_addmethod(xfade_tilde_class,
        (t_method)xfade_tilde_dsp,
gensym("dsp"), A_CANT, 0);
  CLASS_MAINSIGNALIN(xfade_tilde_class,
t_xfade_tilde, f);
}
```

Play!



Play!

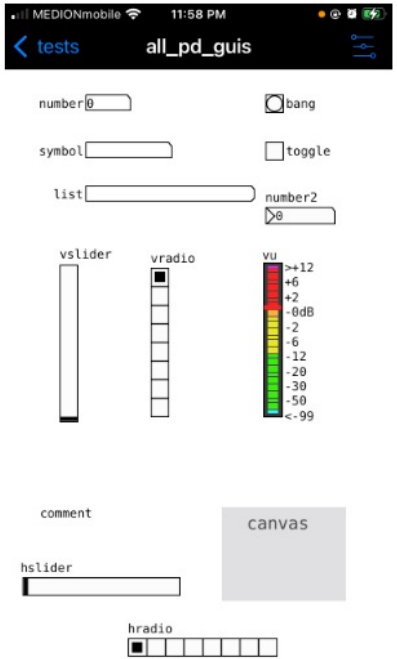
So many ways to play!

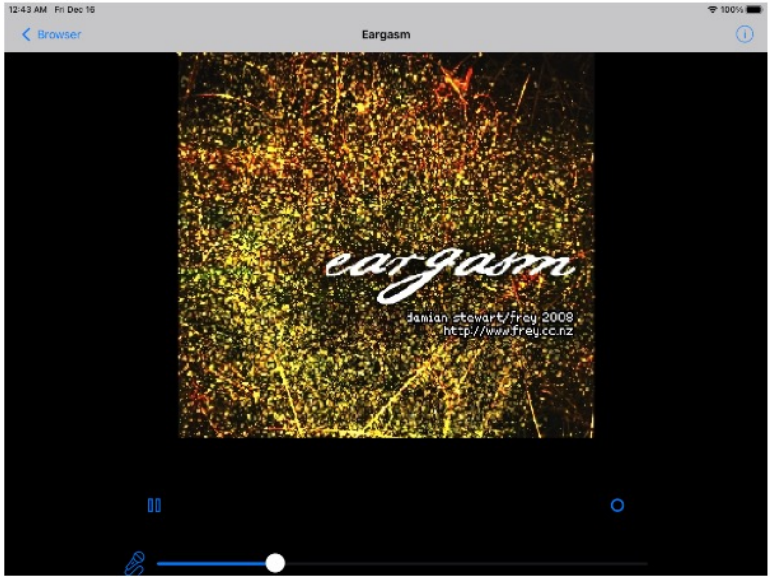
- PD & Max can support nearly every possible type of controller because the initial design of Max only supported MIDI – i.e controllers. So it's been a part of the environments since the beginning
- A few examples:
- Instruments / Midi
- Control surfaces / Android and iOS tablets
- OSC (Open Sound Control)
- iOS Device inputs
- Game controllers
- Cameras

Play

Running PD on your Phone!

RJDJ / PD Party / PD Droid Party





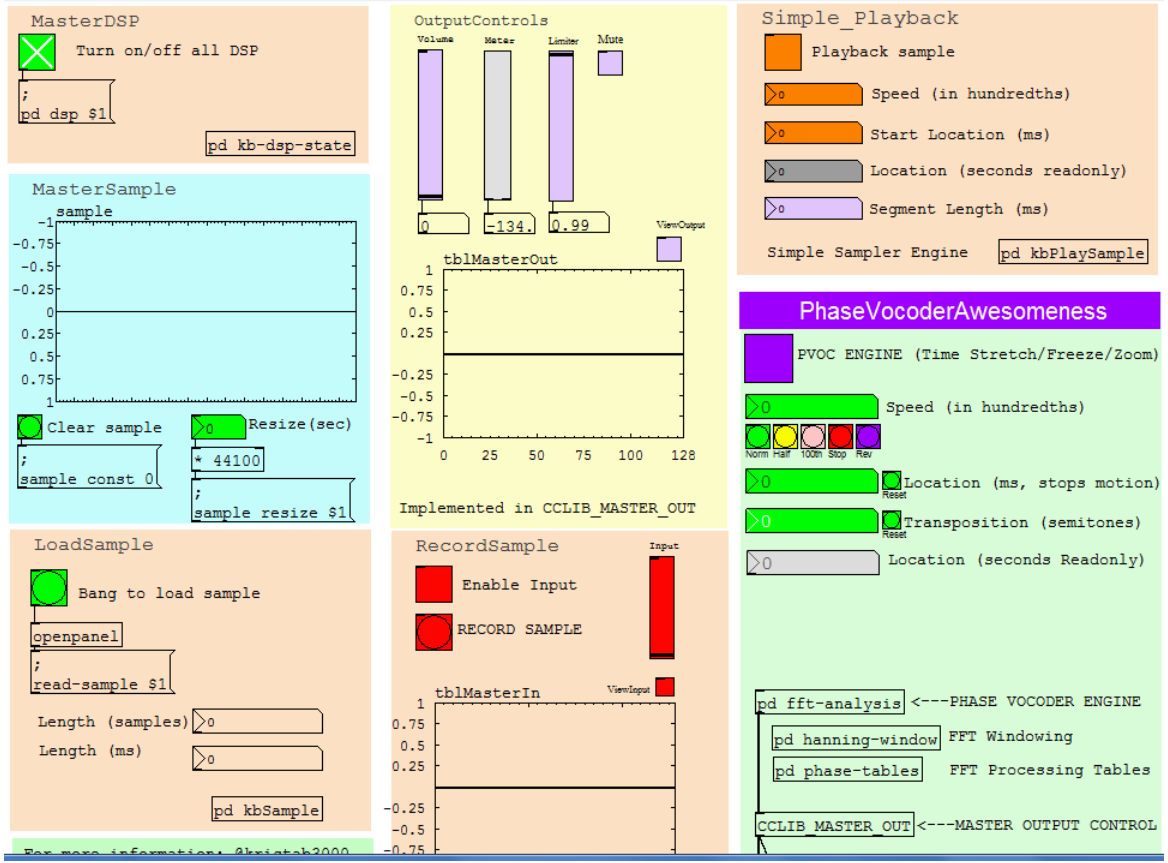
Play

kbPVoc.pd – Phase Vocoder Based Audio Microscope

1. DSP / Output

2. Master Sample

3. Load / Record



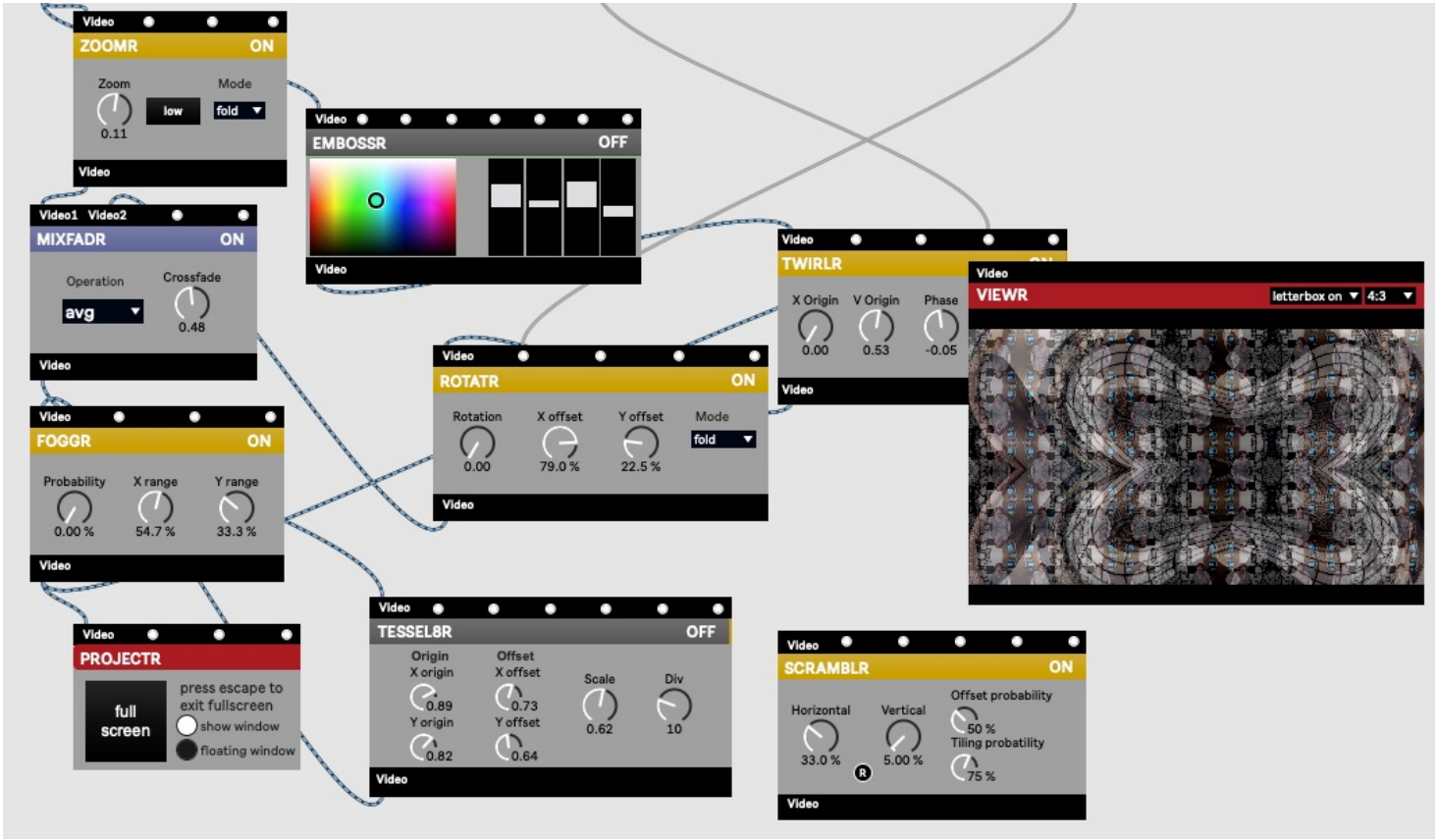
4. Simple Playback Controls

5. Phase Vocoder (Micrcoscope) Controls

6. Engine

Play

Vizzie – Realtime Video Processing



Links and Info

- PureData Download and InfoHub: http://puredata.info/
- Miller Puckette's Companion Book (best source to get started): http://msp.ucsd.edu/techniques.htm
- Max (Commercial Version): https://cycling74.com/
- IRCAM (one of the coolest places on the planet and where most of this originated from) : https://www.ircam.fr/
- Email me: krista.campbell@insight.com

Thank You!

Krista Campbell
Architect, Insight
krista.campbell@insight.com