Notating musical movement

Richard Hoadley

CoDE 2012: 1st Annual Conference

March 2012

http://rhoadley.net/presentations/code2012.pdf

//
//

“...the mystifyingly exquisite variation that can match the instruments on which

music has been made for centuries.”

The One-Handed Musical Instrument Trust
http://www.ohmi.org.uk/

//
//

Jamie Cullum, Desert Island Discs, Sunday 25th March

Combining Research Streams

- Real-time algorithmic generation of music
- Real-time generation and presentation of a notated version of this music and

thence, performance and improvisation

- Real-time physical interaction

['algorithmic' here means 'computer-generated algorithmic music which happens to be realtime in production' (Collins 2008)]

//
//

- Algorithmic music has often been analysed as being either 'top down' (macro-structure-based) or 'bottom up' (micro-structure-based)
- One tendency is for algorithms to work on a micro-structure as vignettes (cf. James McCartney, SC), or as

inordinately long installations (minimalist)

//
//

Algorithmic Example

```
(
~synPianoDo = ({ arg myPitch;
var n;
n = 12; // number of keys playing
n.postln;
~synPiano = ({
    Mix.ar(Array.fill(n, {   // mix an array of notes
        var delayTime, pitch, detune, strike, hammerEnv, hammer;
        pitch = myPitch.choose; // calculate delay based on a random note
        strike = Impulse.ar(0.1+0.4.rand, 1.0.rand, 0.1); // random period
for each key
        hammerEnv = Decay2.ar(strike, 0.008, 0.04); // excitation envelope
        Pan2.ar(
```

```
                Mix.ar(Array.fill(3, { arg i; // array of 3 strings per note
                detune = #[-0.05, 0, 0.04].at(i); // detune strings,
calculate delay time :
                delayTime = 1 / (pitch + detune).midicps; // each string gets
own exciter :
                hammer = LFNoise2.ar(3000, hammerEnv); // 3000 Hz was chosen
by ear..
                CombL.ar(hammer,     // used as a string resonator
                    delayTime,       // max delay time
                    delayTime,       // actual delay time
                    6)            // decay time of string
            })),
            (pitch - 36)/27 - 1 // pan position: lo notes left, hi notes
right
        )
    })
}).play
})
)

~synPiano.free; ~synPianoDo.value([ 36, 48, 55, 60, 64, 67, 70, 74, 76 ]);

~synPiano.free; ~synPianoDo.value([ 36, 48, 56, 60, 63, 68, 72, 75, 79 ]);

b = Array.fill(12, {(36 + (54.rand))});
b.class
~synPiano.free; ~synPianoDo.value(b);

~synPiano.free;

////////////////////////////////////////////////
////////////////////////////////////////////////
```

Mark Franz 1997 'Mathematics and Art'

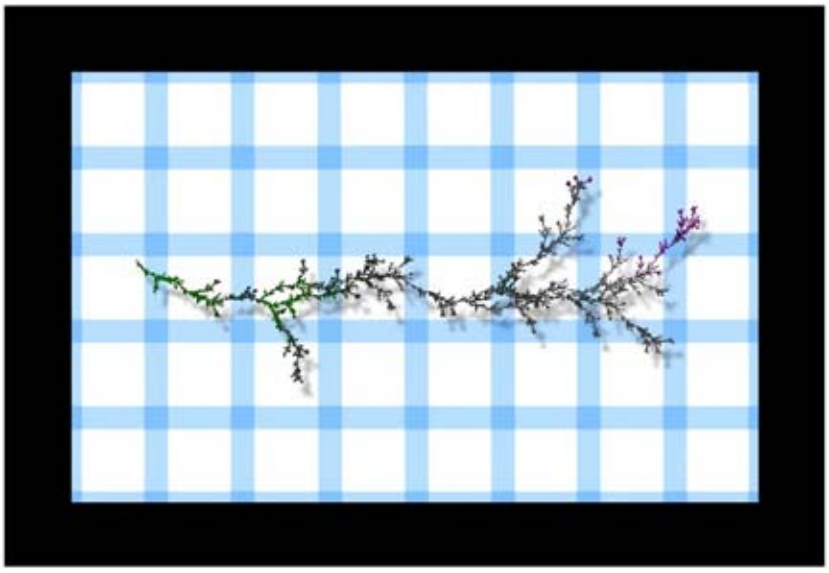


Figure 1. *Vine and Tablecloth*

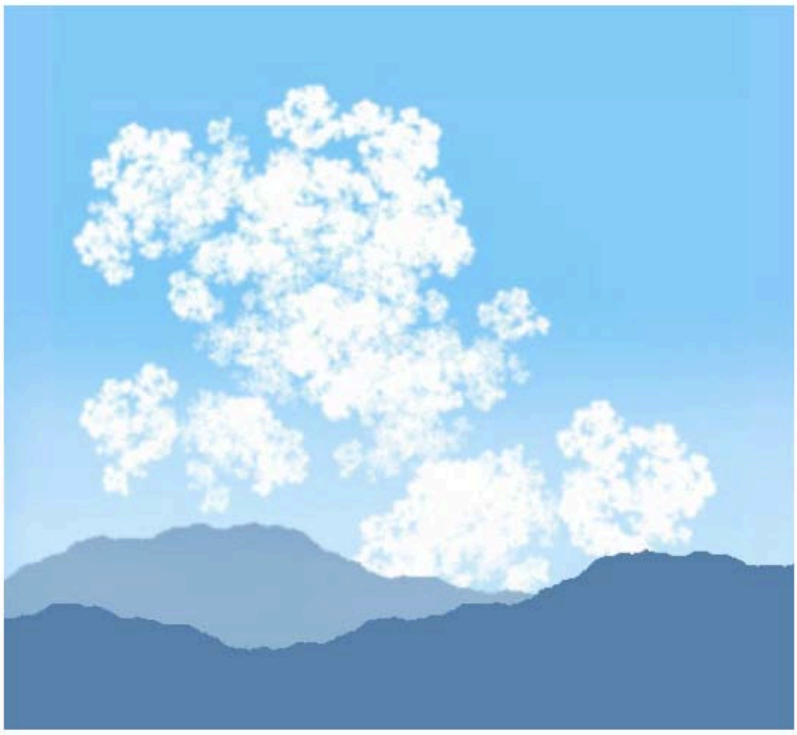


Figure 11. A landscape made from IFS attractors.

- The majority of western music is of intermediate length (say, between 3 minutes and 3 hours) and use structures where there is a ‘top down’ component.

- Usually composers work somewhere along the continuum between these two extremes.

- A useful metaphor compares composing to mapping a new territory.

Some examples of previous work using algorithms and physical computing...

//
//

Human Computer Interaction 2009, Cambridge

external video link: http://vimeo.com/7710584

//

MIST 2010

external video link: http://vimeo.com/13801015

//
//

Triggered: Metapiano, June 2011

external video link: http://vimeo.com/25465581

//
//

Calder’s Violin

combining research streams:

- algorithmic composition
- music composition, performance, improvisation and notation
- physical computing, potentially (hence title)

involving:

- SuperCollider for audio and programming...
- ...linked via Open Sound Control (OSC) to INScore (http://inscore.sourceforge.net/)...
- ...also potentially uses microprocessor (Arduino or mBed) for physical computing (although not in current version of piece)

some excerpts...

//
//

Calder's Violin (excerpts), Mifune Tsuji, October 2011

Video directed and edited by Antonis Papavasiliou
external video link: http://vimeo.com/32520438

//
//

Why do it?

- comprehension: writing algorithms allows a deeper understanding of basic compositional processes - it's both 'different' from but similar to 'normal' composition (and it certainly doesn't save time!)

- imagination: it allows insight into and exploitation of areas of thought not normally considered consciously: differently balanced phrases and coloured harmonies, lots of notes, playability...

- predicability: it allows immediate audition but maintains an element of unpredictability, maybe mirroring performance. Balance with control and micro/macro structuring

- performance practice: giving (classically trained?) performers notation tends to mean they perform with more confidence more quickly

//

//////////////////////////////////////

Demos

- score demo, then add physical computing
- why? creative reasons (dancer causes playable notation), educational/ therapeutic (fool-proof sight-reading!), analytical: analysis of algorithmic output (although how useful is this if you know the algorithm?)...
- notation is another form of data representation
- it’s also just an interesting thing to do.

```
//=================================================================
//=================================================================
// digiphone
// a device for investigation of finger movements
// also controlled by a minimonome
```

```
// v003 dmrn demo
// v004 speec demo
// original richbook display: 1680x1050
//=====================================================================
//=====================================================================

~digiPort.close
SerialPort.closeAll

s.boot;

(
~panSetup = 2;
// common library
"~/Documents/Music/devs/common/func/*".loadPaths;
"~/Documents/Music/devs/common/synth/*".loadPaths;

// melodia library
"~/Documents/Music/devs/melodia/func/*".loadPaths;
"~/Documents/Music/devs/melodia/synth/*".loadPaths;

// metapiano library
"~/Documents/Music/devs/metapiano/func/*".loadPaths;
"~/Documents/Music/devs/metapiano/synth/*".loadPaths;

// calder library
"~/Documents/Music/devs/calder/func/*".loadPaths;

// conway library
"~/Documents/Music/devs/conway/func/*".loadPaths;
"~/Documents/Music/devs/conway/synth/*".loadPaths;

// local
a = "func/*".resolveRelative;
a.loadPaths;
// a = "synth/*".resolveRelative;
// a.loadPaths;
)

~rev = Synth.new("reverb", [\revTime, 1.5, \inBus, ~effect]);
~rev.free;
~rev.set(\revTime, 1.4);

~calderMIDI.value; // if you like
MIDIIn.disconnect; // if you don't like


// variables, not score
(
~inRange = false;
~buttState = 0;
~doingChord06cs == false;
~doingChord06conNum = 0;
~myTupletCoin = 0.00; // the chance that we'll hear a melisma (1.0 = certainty)
~myAmp = 1.0;
// chordStream variables
~myTempo = 160;
~myRel = 1.1;
// a spec:
// if ~myTempo = 120, ~myRel = 1.0, if ~myTempo = 20, ~myRel = 8.0
~toDoChord06csNum = 1; // for the notation of the chords
~doingPedalArpDigiNum = 0; // for speec demo...
~doingPedalArpTrillDigiNum = 0;
~doingTrill01DigiNum = 0;
// for digiChord
~digiChordQuantiseValue = 0.33;
~genDelay = 0.0; // for ~chord03digi below....
)

//=====================================================================
// score
//=====================================================================

(
```

```
// "/Applications/INScore-0.81/INScoreViewer.app/Contents/MacOS/INScoreViewer".unixCmd;
"open '/Users/rich/Documents/Music/devs/digiphone/scores/digiPhone90.inscore'".unixCmd; // maybe
need to check inscore version
// open the viewer, then
NetAddr.disconnectAll; ~digiScore = INScore.new("127.0.0.1", 7000);
~doScore = NetAddr.new("127.0.0.1", 7000);
// NetAddr.disconnectAll; ~digiScore = NetAddr.new("RichDPLBook.local", 7000);
// NetAddr.disconnectAll; ~digiScore = NetAddr.new("169.254.236.40", 7000); // ethernet
)
(
// ~digiScore.gui(~digiScore);
~digiScore.guiSmall(~digiScore);
~myNoteArray = "\\clef<\"g\"> \\meter<\"2/4\"> { a }  ";
~digiScore.note(1, ~myNoteArray);
~digiScore.scale("/myScore", 1, 4.0);
)

~digiScore.mouse("hide");
~digiScore.mouse("show");

//=====================================================================
// sound test
Synth.new("metaPiano", [ \myFreq, 440, \myVel, 80, \myDecay, 1.008, \myAttack, 0.0, \myRelease, 2.6,
\myTune, 0.0, \random, 0.1, \pan, ((2.0.rand)-1), \outBus, 0, \effectBus, ~effect  ]);


//=====================================================================
// score demo
//=====================================================================
// slow sequence of triad quavers
// hear and see all
~digiScore.scale("/myScore", 1, 12.0);
~chord03digi.value(4, 52+(12.rand), 1.5, 6, 0.002, 48, 1.2, 0.0, true, true, 1, "chord" );
// seventh chords
~chord06digi.value(52+(12.rand), [ 0, 3, 4, 3 ], 0.0, 0.002, 48, 1.2, 0.0, true, true, 1, "chord" );
// series of seventh chords
(
~phraseArray = "";
Task({
     8.do({
          ~chord06digi.value(52+(12.rand), [ 0, 3, 4, 3 ], 0.0, 0.002, 48, 1.2, 0.0, true, true, 1,
"chord" );
          0.5.wait;
     });
}).play;
)


// fast sequence of diads
~chord03digi.value(12, 58+(12.rand), 0.15, 2, 0.002, 40, 2.2, 0.0, true, true, 1 );

// just hear
~myNoteArray = " "; ~digiScore.note(1, ~myNoteArray);
~chord03digi.value(5, 52+(12.rand), 0.5, 4, 0.002, 52, 1.2, 0.0, false, true, 1, "chord" );

// just see
~chord03digi.value(5, 52+(12.rand), 0.5, 4, 0.002, 45, 1.2, 0.0, true, false, 1, "chord" );

// just see only one line (for instrumental)
~chord03digi.value(6, 58+(12.rand), 1.1, 4, 0.002, 50, 2.2, 0.0, true, true, 1, "line" );


// the calder final melody.  The higher the value of the argument, the more the chance of rests
~digiScore.scale("/myScore", 1, 5.0);
~s5MelodyScoreArray=""; ~doMelBitDigi.value(0.5);

// different scenes...
~myNoteArray = " "; ~digiScore.note(1, ~myNoteArray); // clear, or
~digiScore.delete;
~chord03digi.value(12, 48+(12.rand), [ 0.15, 0.25, 0.5 ].choose, 4, 0.002, 40, 1.2, 0.0, true, true,
1 ); // scene 1
~digiScore.scale("/myScore", 1, 6.0);

~chord03digi.value(8, 58+(12.rand), [ 1.1, 0.55 ].choose, 4, 0.002, 40, 2.2, 0.0, true, true,
```

```
2 ); // scene 2
~digiScore.scale("/myScore", 2, 6.0);

// image:
~digiScore.file(1, "/Users/rich/Documents/Dropbox/devs/class/rooster.png")
~digiScore.file(1, "/Users/rich/Documents/Music/devs/digiphone/media/moran.png")
~digiScore.file(1, "/Users/rich/Documents/Music/devs/digiphone/media/zyklus.png")
~digiScore.file(1, "/Users/rich/Documents/Music/devs/digiphone/media/bell.png")

~digiScore.happybirthday

~digiScore.text(1, "hello, world!")
~digiScore.scale("/myText", 1, 6.0)




//=====================================================================
// reset window contents
//=====================================================================

~digiScore.delete;

(
~digiScore.note(1, " ");
~digiScore.scale("/myScore", 1, 6.0);
~digiScore.move("/myScore", 1, 0.0, -0.8);

~digiScore.note(2, " ");
~digiScore.scale("/myScore", 2, 6.0);
~digiScore.move("/myScore", 2, 0.0, -0.2);

~digiScore.note(3, " ");
~digiScore.scale("/myScore", 3, 6.0);
~digiScore.move("/myScore", 3, 0.0, 0.4);
)

//=====================================================================
// algorithms demo
//=====================================================================

// set 'play' to 'false' for immediate score appearance
~pedalArpDigiTask.stop;
~pedalArpDigi.value([5, 7], false, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)
+0.3, 0.1, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, false, 1);
~pedalArpDigi.value([5, 7], false, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)
+0.3, 0.1, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, true, 2);
~pedalArpDigi.value([5, 7], false, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)
+0.3, 0.1, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, true, 3);

// a bit more detail
// concentrate on one stave

(
~digiScore.note(1, " ");
~digiScore.scale("/myScore", 1, 1.0);
~digiScore.move("/myScore", 1, 0.0, -0.8);

~digiScore.note(2, " ");
~digiScore.scale("/myScore", 2, 6.0);
~digiScore.move("/myScore", 2, 0.0, -0.2);

~digiScore.note(3, " ");
~digiScore.scale("/myScore", 3, 1.0);
~digiScore.move("/myScore", 3, 0.0, 0.4);
)

~panSetup

// arguments
arpNum=[3, 4], arpNumRand=false, arpSubRand=false, arpInts=[ 5, 7 ], arpSubInts=[ 3, 6 ], arpKey=
[1,1], subWait=0.12, subWaitRand=0.0, arpWait=0.001, arpWaitRand=0.0, subAccel=0.0,
subAccelRand=false, myVel=60, myRel=1.0, delay=0.0, display=true, play=true, itemNum=1;
```

```
// always one 'arpeggio', always two notes, notes will always be a tone apart
// NB durations and notation are very tricky, as they are fairly intuitively dealt with by people. From a computing point of view, very confusing!
~pedalArpDigi.value([1, 4], false, false, [ 2 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)+0.3, 0.3, 0.001, 0.0, 0.018, true, 70, 1.2, 0.0, true, true, 2);

// as above, but always two arpeggios, each of two notes.  The first notes of each arpeggio will always be a minor third apart
~pedalArpDigi.value([2, 2], false, false, [ 2 ], [ 3 ], [ 4.rand, 5 ], (0.1.rand)+0.3, 0.3, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, true, 2);

// as above, but the first arpeggio will always start on 'note 1' (c#) of the fifth octave
~pedalArpDigi.value([2, 2], false, false, [ 2 ], [ 3 ], [ 1, 5 ], (0.1.rand)+0.3, 0.3, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, true, 2);

// as above, but four arpeggios of four notes each and now with a fixed duration
~pedalArpDigi.value([4, 4], false, false, [ 2 ], [ 3 ], [ 1, 5 ], 0.1, 0.0, 0.001, 0.0, 0.0, true, 60, 1.2, 0.0, true, true, 2);

// as above, but four arpeggios of four notes each, fixed duration slightly randomised
~pedalArpDigi.value([4, 4], false, false, [ 2 ], [ 3 ], [ 1, 5 ], 0.1, 0.3, 0.001, 0.0, 0.0, true, 60, 1.2, 0.0, true, true, 2);

// as above, including accelerando.  Note if this is too high a value, the notation will not work correctly
~pedalArpDigi.value([4, 4], false, false, [ 2 ], [ 3 ], [ 1, 5 ], 0.2, 0.3, 0.001, 0.0, 0.01, false, 60, 1.2, 0.0, true, true, 2);

// you can also randomise the accelerando, meaning you need to increase the general rate
~pedalArpDigi.value([4, 4], false, false, [ 2 ], [ 3 ], [ 1, 5 ], 0.2, 0.3, 0.001, 0.0, 0.02, true, 70, 1.2, 0.0, true, true, 2);

// finished example with various randomisers and choices
~pedalArpDigi.value([6, 9], false, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)+0.3, 0.1, 0.001, 0.0, 0.025, true, 60, 1.2, 0.0, true, true, 2);

// finished example with various randomisers and choices
~pedalArpDigi.value([4, 8], false, false, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], 0.3, 0.1, 0.001, 0.0, 0.015, true, 60, 1.2, 0.0, true, true, 2);

// now as chords, although remember the actual wait time also depends on the acceleration
// notation as chords has been provisionally implemented: the ease with which one can move between the horizontal and the vertical is one of the more interesting features...
~pedalArpDigi.value([8, 4], false, false, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], 0.00001, 0.0, 0.2, 0.0, 0.0, true, 50, 1.2, 0.0, true, true, 2);
~pedalArpDigi.value([4, 4], false, false, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], 0.1, 0.0, 0.0, 0.0, 0.0, true, 40, 1.2, 0.0, true, true, 2);

// lots of short, quiet four note chords.  Impractical.
~pedalArpDigi.value([32, 4], false, false, [ 1, 2, 3, 4, 5 ], [ 2, -2 ], [ 4.rand, 5 ], 0.00001, 0.0, 0.1, 0.4, 0.0, true, 30, 2.2, 0.0, true, true, 2);

// one with a trill at the end (as in Calder's Violin)
~pedalArpTrillDigi.value([6, 7], false, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, 5 ], (0.1.rand)+0.3, 0.3, 0.001, 0.0, 0.018, true, 60, 1.2, 0.0, true, false, 2);


//=====================================================================
// digiphone
//=====================================================================
~digiGUI.value("usbmodemfa131");
~digiGUI.value("FireFly-0580-SPP");
~digiGUI.value("usbserial-A9007Lvk"); // mega
~digiGUISmall.value("usbserial-A9007Lvk"); // mega

// scale
~digiScore.delete
~digiScore.scale("/myScore", 1, 6.0);

~digiPort.close
SerialPort.closeAll

// nothing without something in range
```

```
// defaults: pPatt = [[ 9, 12, 16 ], [ 9, 12, 17 ], [ 9, 12, 19 ]], dPatt=[ 0.25 ], trans=60;


// simple arpeggios, only when something's there; the further the distance, the higher the pitches.
The more movement detected, the more events there are...
~digiMP.value
~digiMPTask.stop;
~digiMPTask.stop; ~digiMP.value;
~digiScore.reset

/*
summary of movements:
//=========================================
button 1, ~buttState = 1

first just arpeggio:
~digiScore.delete; // if necessary
(
~digiScore.scale("/myScore", 1, 6.0);
~digiScore.text(1, "more movement, more and faster notes; higher hand position, higher notes");
~digiScore.scale("/myText", 1, 2.0);
~digiScore.move("/myText", 1, 0.01, 0.94);
)
more movement: more and faster notes;
higher hand position, higher notes, arranged in octave tiers
distance and octave - currently distances are not scaled so the greater the distance, the greater
the range of speeds...
notates

//=========================================
button 2, ~buttState = 4
nice chords

(
~digiScore.scale("/myScore", 1, 12.0);
~digiScore.text(1, "plays an arpeggio when you move in range, the greater the distance, the higher
         the pitch.");
~digiScore.scale("/myText", 1, 2.0);
~digiScore.move("/myText", 1, 0.01, 0.94);
)
plays an arpeggio when you move in range.
the greater the distance, the higher the pitch.
can be made to play more arpeggios with more movement
could possibly be a 'harp' with virtual 'chords'
notates (but only last 'played' chord)
~genDelay = 0.0;

//=========================================
button 3, ~buttState = 2

(
~digiScore.scale("/myScore", 1, 5.0);
~digiScore.text(1, "the more movement, the higher the frequency and the more notes");
~digiScore.scale("/myText", 1, 2.0);
~digiScore.move("/myText", 1, 0.01, 0.94);
)

frequency responds to movement: the more movement, the higher the frequency and the more notes;
notates

//=========================================
button 4, ~buttState = 8
frequency responds to a higher hand
more hand movement equals more notes
notates

//=========================================
buttons 1 & 2, ~buttState = 5
rhythmic chords (in development)

(
~digiScore.scale("/myScore", 1, 12.0);
~digiScore.text(1, "hand movement in range will turn on sequence, another will stop it.
```

```
each movement produces a different chord");
~digiScore.scale("/myText", 1, 2.0);
~digiScore.move("/myText", 1, 0.01, 0.90);
)

hand movement in range will turn on sequence, another will stop it.
each movement produces a different chord

// experiment with these
~myTempo = 40; ~myRel = 12.0;
~myTempo = 160; ~myRel = 1.0;
~myTempo = 580; ~myRel = 0.05;

//=========================================
buttons 3 & 4, ~buttState = 10
~digichord (in development)
produces a stream of chords in time, when in range and there is movement.
~digiChordQuantiseValue = 0.75; // experiment with this
~digiChordQuantiseValue = 1.25;
~digiChordQuantiseValue = [ 1.25, 1.0, 0.75, 0.5, 0.25 ].choose;

//=========================================
to do
use three pings separately

*/
```

//
//

Calder’s Violin

Tuesday April 17th 2012, City University, London, part of the SuperCollider Symposium

Metapiano installation

Conway Hall, London, April 14th

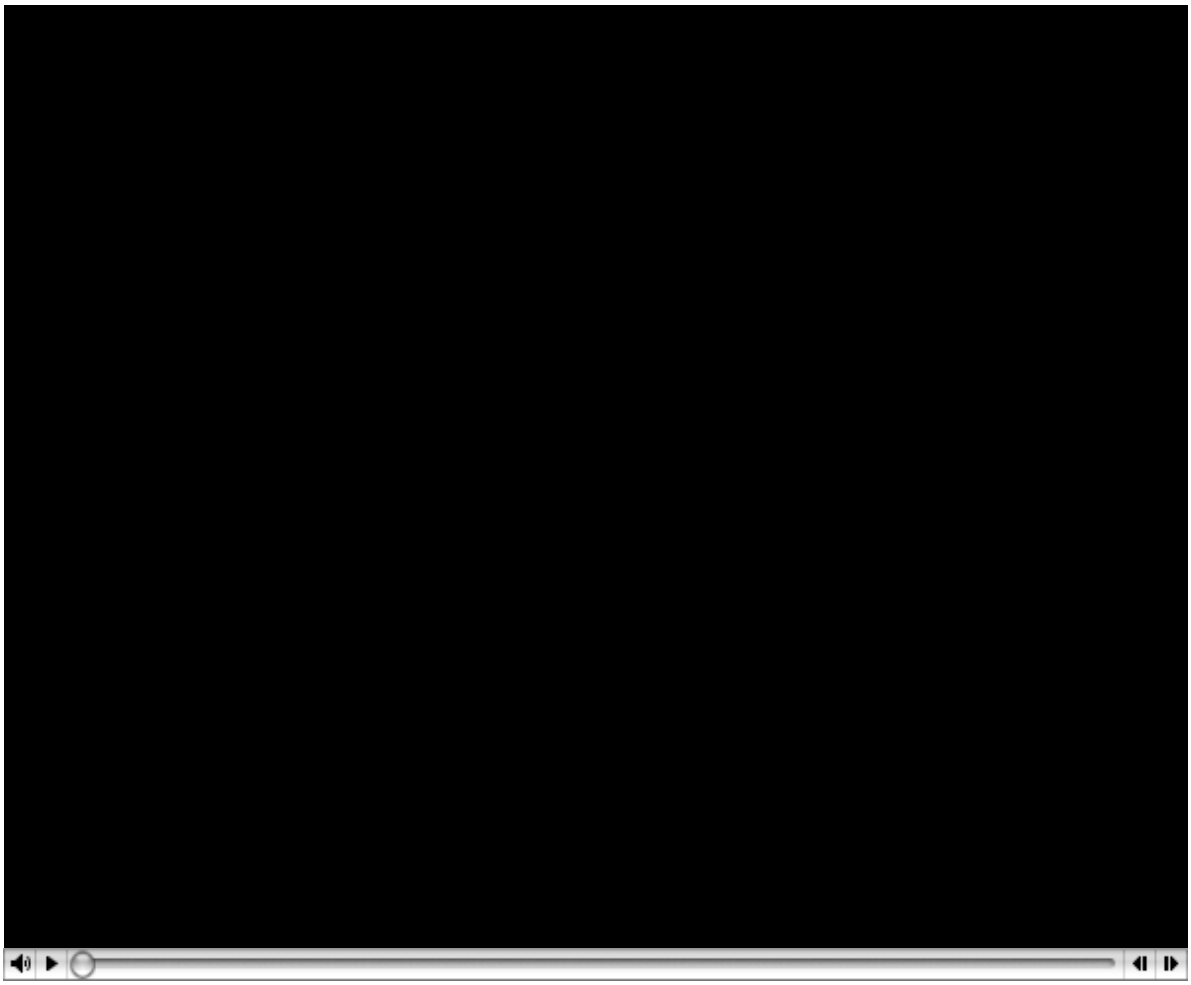
Touching Sound

Phil Barnard (MRC Cognition and Brain Sciences Unit), Ian Cross (West Road Centre for Music and Science), Satinder Gill (West Road Centre for Music and Science), Tommi Himberg (University of Jyvaskyla, Finland), Richard Hoadley (CoDE/DPL/MPA), Helen Odell-Miller (ARU

Music and Health Research Group), Gill Westland (Cambridge Body Psychotherapy Centre), Sam Aaron (Cambridge University Computer Laboratory) and Rob Toulson (ARU CoDE).

//
//

Calder’s Violin (excerpts), Mifune Tsuji, October 2011



external video link: http://vimeo.com/32520438

//
//

Thankyou...

richard.hoadley@anglia.ac.uk
research@rhoadley.net

```
////////////////////////////////////////////////////////////
///
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//////
////////////////////////////////////////////////////////////
///
```

```
// a mini algorithmic task
// prepare the score

~digiScore.delete
(
~digiScore.note(1, " ");
```

```
~digiScore.scale("/myScore", 1, 4.0);
~digiScore.move("/myScore", 1, 0.0, -0.8);

~digiScore.note(2, " ");
~digiScore.scale("/myScore", 2, 4.0);
~digiScore.move("/myScore", 2, 0.0, -0.5);

~digiScore.note(3, " ");
~digiScore.scale("/myScore", 3, 4.0);
~digiScore.move("/myScore", 3, 0.0, -0.2);

~digiScore.note(4, " ");
~digiScore.scale("/myScore", 4, 4.0);
~digiScore.move("/myScore", 4, 0.0, 0.1);

~digiScore.note(5, " ");
~digiScore.scale("/myScore", 5, 4.0);
~digiScore.move("/myScore", 5, 0.0, 0.4);

~digiScore.note(6, " ");
~digiScore.scale("/myScore", 6, 4.0);
~digiScore.move("/myScore", 6, 0.0, 0.7);
)

(
Task({
     30.do({ arg i;
          ~pedalArpDigi.value([6, 8], true, true, [ 1, 2, 3, 4, 5 ], [ 1, 3, 5 ], [ 4.rand, (3.rand +
4) ], 0.2, 0.1, 0.0, 0.0, 0.016, true, 40, 2.2, 0.0, true, true, ((i%6)+1));
          0.5.wait;
     });
}).play;
)

// a bit more minimal?
(
Task({
     30.do({ arg i;
          ~pedalArpDigi.value([6, 8], true, true, [ 5, 7 ], [ 0 ], [ 0, 4 ], 0.2, 0.1, 0.0, 0.0,
0.016, true, 40, 2.2, 0.0, true, true, ((i%6)+1));
          0.5.wait;
     });
}).play;
)

(
Task({
     60.do({ arg i;
          ~pedalArpDigi.value([2, 8], true, true, [ 5, 7 ], [ 1, 3, 5 ], [ 4.rand, (3.rand + 4) ],
0.0001, 0.0, 0.0, 0.0, 0.0, true, 40, 2.2, 0.0, true, true, ((i%6)+1));
          [0.1, 0.2, 0.4].choose.wait;
     });
"done".postln;
}).play;
)




~digiScore.delete
(
~digiScore.note(1, " a ");
~digiScore.scale("/myScore", 1, 4.0);
~digiScore.move("/myScore", 1, 0.0, -0.4);

~digiScore.note(2, " a ");
~digiScore.scale("/myScore", 2, 4.0);
~digiScore.move("/myScore", 2, 0.0, 0.4);
)


// multipatt, the origin of lines
~multiPattMP.value(1, 1, false, 120, true, loopWait:0.025, loopWaitRand:false, pitchInc: 3,
```

```
pitchIncRand: true, loNote: 60, hiNote:100, loVel:10, hiVel:60);

// lines, chromatic, winding
// now with 'displayWhen' parameter, 0 (def) = display after each loop, 1: display as generated, 2: display after all loops

~multiPattDigiTask.stop; ~multiPattDigiTask.reset; ~multiPattDigiLines.value(1, 1, false, 50, false, loopWait:0.05, loopWaitRand:false, pattWait: 1.4, pitchInc: [3], pitchIncRand: true, loNote: 60, hiNote:100, loVel:10, hiVel:60, displayWhen:1, delay: 0.0, display: true, play: true, itemNum: 1);


~multiPattDigiTask.stop; ~multiPattDigiTask.reset; ~multiPattDigiLines.value(1, 1, false, 150, false, loopWait:0.01, loopWaitRand:false, pattWait: 1.4, pitchInc: [1, 11, 23], pitchIncRand: false, loNote: 60, hiNote:100, loVel:10, hiVel:60, displayWhen:2, delay: 0.0, display: true, play: true, itemNum: 1);

// lines, jumpy
~multiPattDigiTask.stop; ~multiPattDigiTask.reset; ~multiPattDigiLines.value(1, 1, false, 30, false, loopWait:0.05, loopWaitRand:false, pitchInc: [1, 11], pitchIncRand: false, loNote: 60, hiNote:100, loVel:10, hiVel:60, displayWhen:1, delay: 0.0, display: true, play: true, itemNum: 2);

// just in case
~multiPattCalderDo.stop; ~multiPattCalderDo.reset; ~bigArray=""; ~multiPattMPTask.stop;
~multiPattMPTask.reset; ~multiPattCalderLines.value(1, 1, false, 6, false, loopWait:0.5, loopWaitRand:false, pitchInc: 3, pitchIncRand: false, loNote: 70, hiNote:80, loVel:10, hiVel:60, delay: 0.0, display: true, play: false, itemNum: 1);
~multiPattCalderDo.stop; ~multiPattCalderDo.reset; ~bigArray=""; ~multiPattMPTask.stop;
~multiPattMPTask.reset; ~multiPattCalderLines.value(1, 1, false, 32, false, loopWait:0.05, loopWaitRand:false, pitchInc: 3, pitchIncRand: false, loNote: 70, hiNote:80, loVel:10, hiVel:60, delay: 0.0, display: true, play: true, itemNum: 1);









~digiMPTask.stop; ~digiMP.value(dPatt: [ 0.1 ], trans: 50 + (12.rand))



~digiMPTask.stop; a = 12.rand; ~digiMP.value(pPatt: [ [ a, a+7 ] ], dPatt: [ 0.25 ], trans: 50 + (12.rand))

~digiMPTask.stop; ~digiMP.value( [[ 10, 13, 17 ], [ 10, 13, 18 ]], [ 0.1 ], 30 );

// the below plays a melody.  The less distance, the lower the pitch of the melody.  The pitch will only fall at the end of a phrase to maintain melodic integrity.  More hand movement, faster melodic movement...
~digiMPTask.stop; ~digiMP.value( [[ 67, 69, 65, 67, 69, 62, 70, 69, 67, 65, 67, 69, 65, 67, 69, 65 ]], [ 0.4 ], 0 );

// 342 row
// [ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ]

// mahler 1st symphony!!!
~myNoteArray = "\\staff<1> \\clef<\"g\"> \\staff<2> \\clef<\"f\"> \\meter<\"2/4\"> ";
~digiMPTask.stop; ~digiMP.value( [[ 0, 2, 3, 2, 0, 0, 2, 3, 2, 0, 3, 5, 7, 3, 5, 7 ]], [ 1.0, 1.0, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0 ], trans: 70 + (12.rand) );
~digiMPTask.stop;

~myNoteArray = "\\clef<\"g\"> \\meter<\"2/4\"> ";

~myNoteArray = "\\clef<\"g\"> \\meter<\"2/4\"> "; ~digiMPTask.stop; ~digiMP.value( [ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ], [ 1.0 ] );

~myNoteArray = "\\staff<1> \\clef<\"g\"> \\meter<\"2/4\"> \\staff<2> \\clef<\"f\"> \\meter<\"2/4\">
"; ~digiMPTask.stop; ~digiMP.value( [ 0, 6, 11 ], [ 0.25 ] );

// zen?
~digiMPTask.stop; ~digiMP.value( [[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ].scramble], [ 6.0, 4.0,
```

```
2.0, 1.0, 0.5, 0.3, 0.2, 0.1, 0.0001 ], 80 ); ~myAmp=0.8; ~myAttackFactor=0.01;
~myTupletCoin = 0.5;

// zen?
~digiMPTask.stop; ~digiMP.value( [[ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ]], [ 6.0, 4.0,
2.0, 1.0, 0.5, 0.3, 0.2, 0.1, 0.0001 ], 80 ); ~myAmp=0.8; ~myAttackFactor=0.01;
~myTupletCoin = 0.5;

a = [[ 9, 12, 16 ], [ 9, 12, 17 ], [ 9, 12, 19 ]]
(a.size).do{ |i| i.postln; a[i].scramble.postln }


// button 1 and zen:
// zen?
~digiMPTask.stop; ~digiMP.value( [[ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ]], [ 2.2, 2.0,
1.0, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0001, 0.0001, 0.0001 ], 80 ); ~myAmp=0.8; ~myAttackFactor=0.01;
~myTupletCoin = 0.0;

// button 2 nice chords:


// button 3, more movement, higher frequency (and more movement):
// and button 4, frequency responds to a higher hand and more hand movement equals more notes
~digiMPTask.stop; ~digiMP.value(dPatt: [ 0.5 ], trans: 50 + (12.rand)); ~myAmp=1.2;
~digiMPTask.stop; ~digiMP.value(dPatt: [ (0.1.rand) + 0.25 ], trans: 50 + (12.rand)); ~myAmp=1.2;
~myTupletCoin = 0.0;

~digiMPTask.value;
~digiMPTask.stop;



//=========================================
// rhythm
//=========================================

(
     ~chordStreamsDurArray = [[ 2.0, 1.5, 1.0, 0.5 ], [ 1.0, 0.5 ], [ 0.5, 0.25 ] ].choose;
     ~chordStreamsDurArray.postln;
     Tempo.bpm_(120);
     ~chordStreams[0].stop;
     ~makeChordStream.value(0, inf, [(40+(5.rand*12))], [ 0, 2, 4, 1, 1, 3, 2, 4 ].scramble,
~chordStreamsDurArray, [ 10, 20, 30, 40 ]);
     ~chordStreams[0].stop; ~chordStreams[0].reset;
~chordStreams[0].stop; ~chordStreams[0].reset; ~chordStreams[0].play;
)

     ~chordStreams[0].stop;

(
~chordStreamsDurArray = [ 2.0, 1.0, 0.5 ];
Tempo.bpm_(120);
          for ( 0, 5, { arg i; ~chordStreams[i].stop; } );
for ( 0, 5, { arg i;
     ~chordStreams[i].stop;
     ~makeChordStream.value(i, inf, [(40+(i*12))], [ 0, 2, 4, 1, 1, 3, 2, 4 ].scramble,
~chordStreamsDurArray, [ 10, 20, 30, 40 ]);
     ~chordStreams[i].stop; ~chordStreams[i].reset; ~chordStreams[i].play;
} );
// a = 3; ~chordStreams[a].stop; ~chordStreams[a].reset; ~chordStreams[a].play;
// ~chordStreams[3].stop; ~chordStreams[3].reset; ~chordStreams[3].play;
)

(
~chordStreamsDurArray = [ 1.0, 0.5, 0.25 ];
Tempo.bpm_(160);
          for ( 0, 5, { arg i; ~chordStreams[i].stop; } );
for ( 0, 5, { arg i;
     ~chordStreams[i].stop;
     ~makeChordStream.value(i, inf, [(40+(i*12))], [ 0, 2, 4, 1, 1, 3, 2, 4 ].scramble,
~chordStreamsDurArray, [ 10, 20, 30, 40 ]);
     ~chordStreams[i].stop; ~chordStreams[i].reset; ~chordStreams[i].play;
} );
// a = 3; ~chordStreams[a].stop; ~chordStreams[a].reset; ~chordStreams[a].play;
```

```
// ~chordStreams[3].stop; ~chordStreams[3].reset; ~chordStreams[3].play;
)

~chordStreams[0].stop; ~chordStreams[0].reset; ~chordStreams[0].play;
~chordStreams[1].stop; ~chordStreams[1].reset; ~chordStreams[1].play;
~chordStreams[2].stop; ~chordStreams[2].reset; ~chordStreams[2].play;
~chordStreams[3].stop; ~chordStreams[3].reset; ~chordStreams[3].play;
~chordStreams[4].stop; ~chordStreams[4].reset; ~chordStreams[4].play;
~chordStreams[5].stop; ~chordStreams[5].reset; ~chordStreams[5].play;

chordStreams[0].stop; ~chordStreams[0].reset;
~chordStreams[1].stop; ~chordStreams[1].reset;
~chordStreams[2].stop; ~chordStreams[2].reset;
~chordStreams[3].stop; ~chordStreams[3].reset;
~chordStreams[4].stop; ~chordStreams[4].reset;
~chordStreams[5].stop; ~chordStreams[5].reset;

~chordStreams[3].stop;
~chordStreams[3].reset; ~chordStreams[2].play;

Tempo.bpm_(140);

(
for ( 0, 5, { arg i;
     ~chordStreams[i].stop;
} );
)
```