

Fluxus: Scheme Livecoding

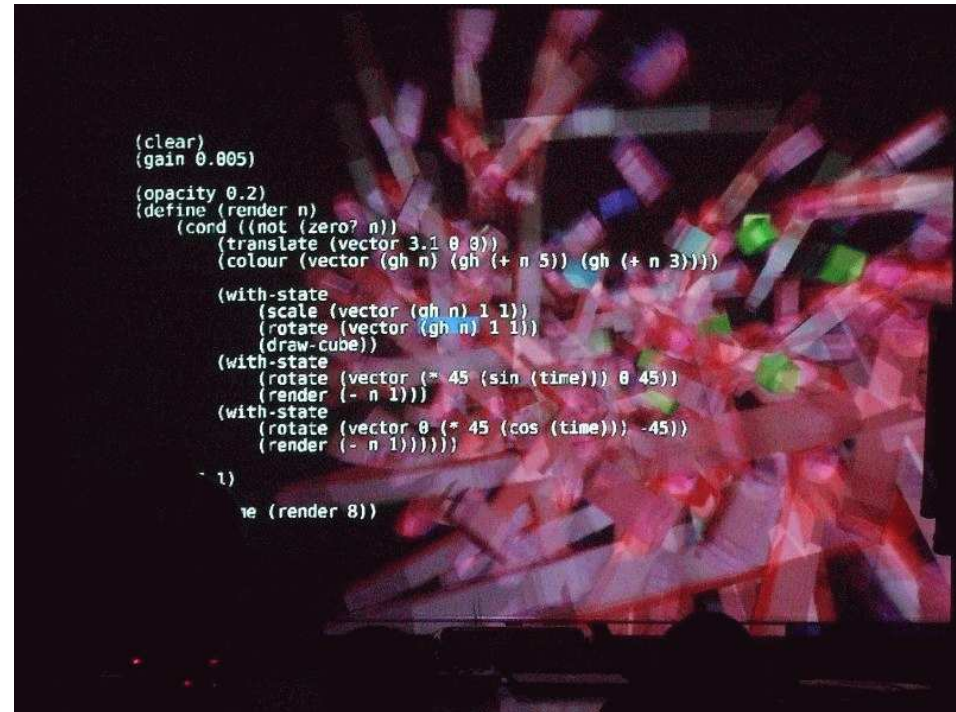
Dave Griffiths

Overview

- What is fluxus?
- Livecoding
- Scheme

What is fluxus?

- Framework for various things:
 - Playing/learning about graphics
 - Workshops
 - Performances
 - Art installations
- Game engine at heart...
- With a livecoding editor
- Source released under GPL
- 4 or 5 developers working on it
- Works on Linux OSX and Windows (ish)
- Uses the Scheme language

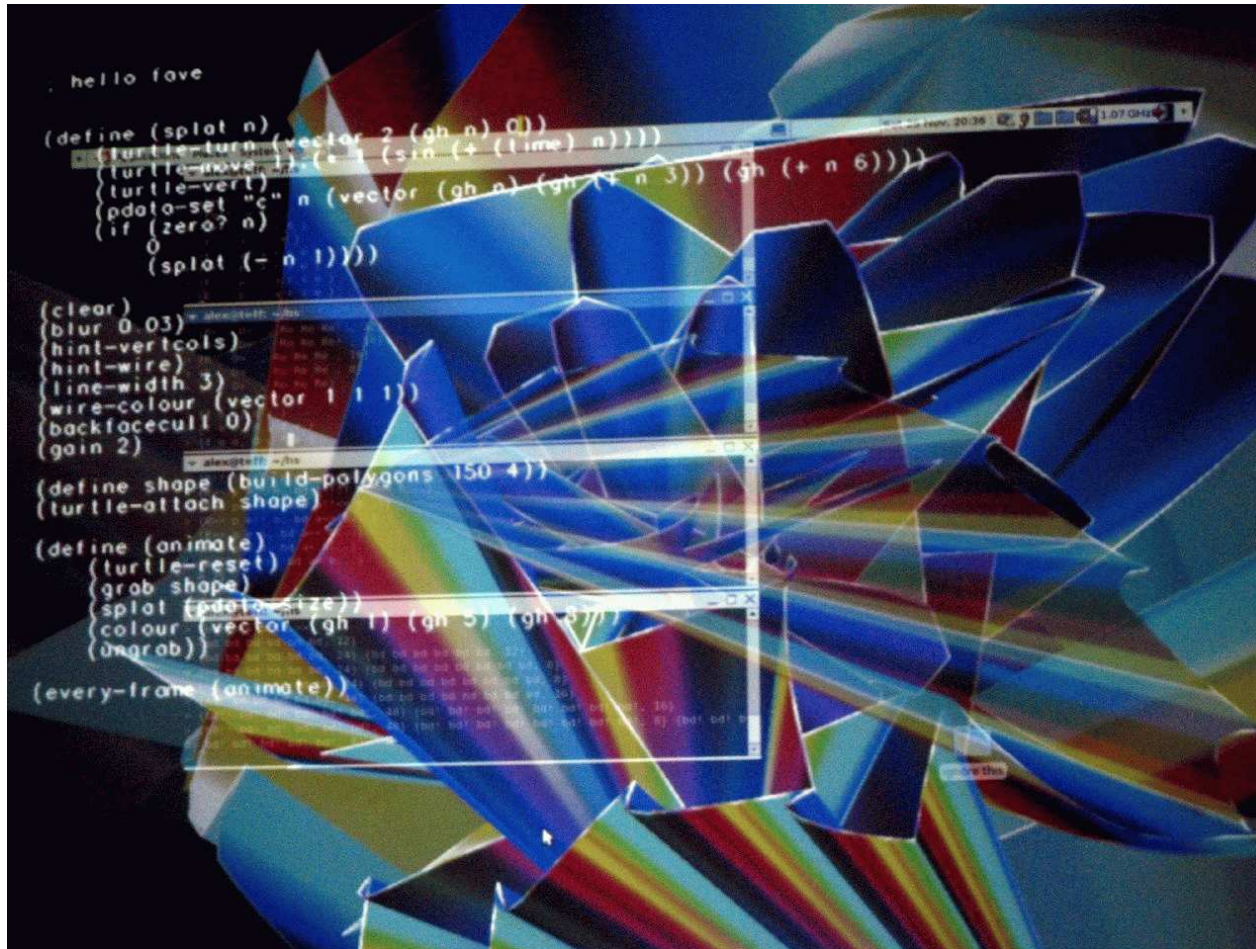


Boring Feature List

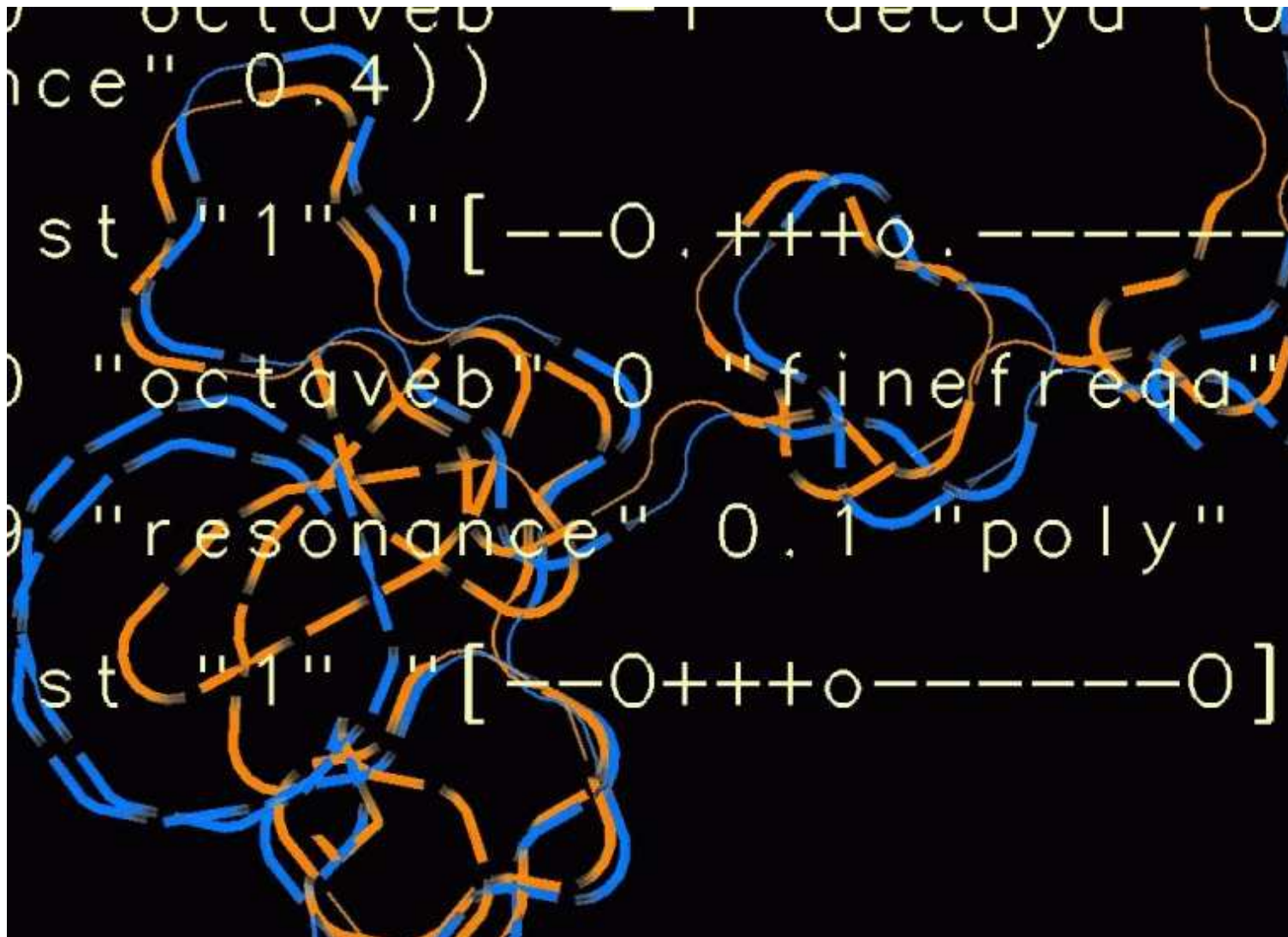
- Immediate mode drawing
- Scenegraph
- Primitives
 - Polys
 - Particles
 - NURBS patches
 - Blobbies (implicit surfaces)
 - Pixels (procedural texture access)
- Unified access to primitive data (vertex arrays, texture data)
- More advanced stuff
 - GLSL Hardware shading
 - ODE physics
 - Shadows
 - Skinning/Skeletons
- Audio synthesis



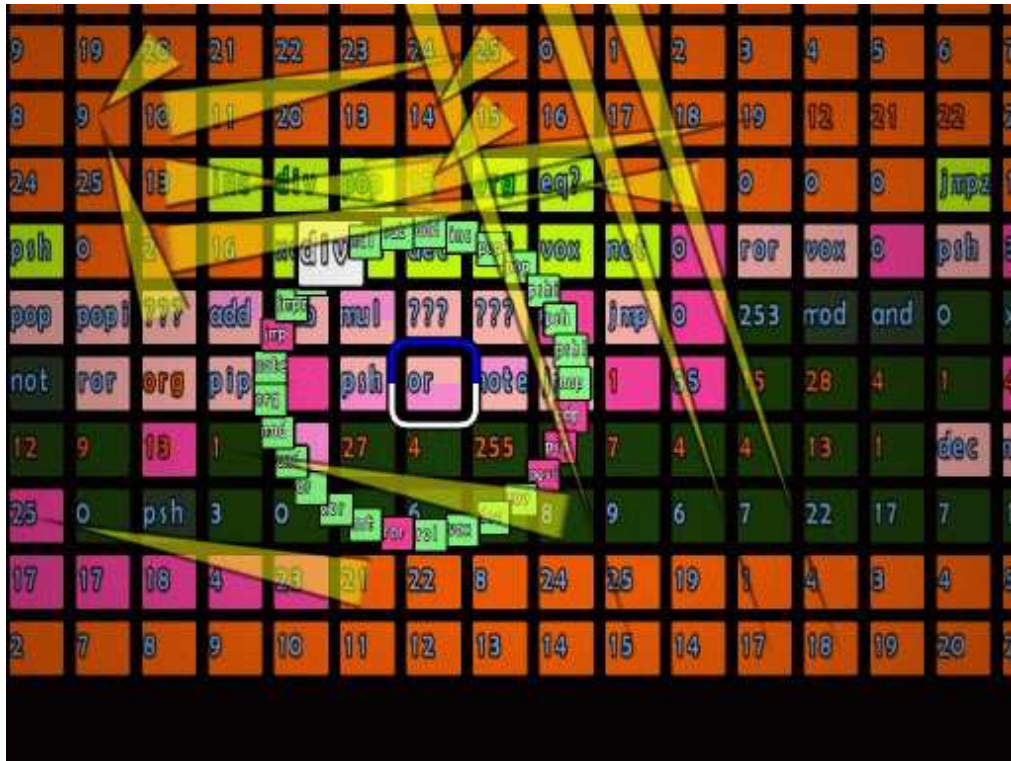
I use fluxus for...



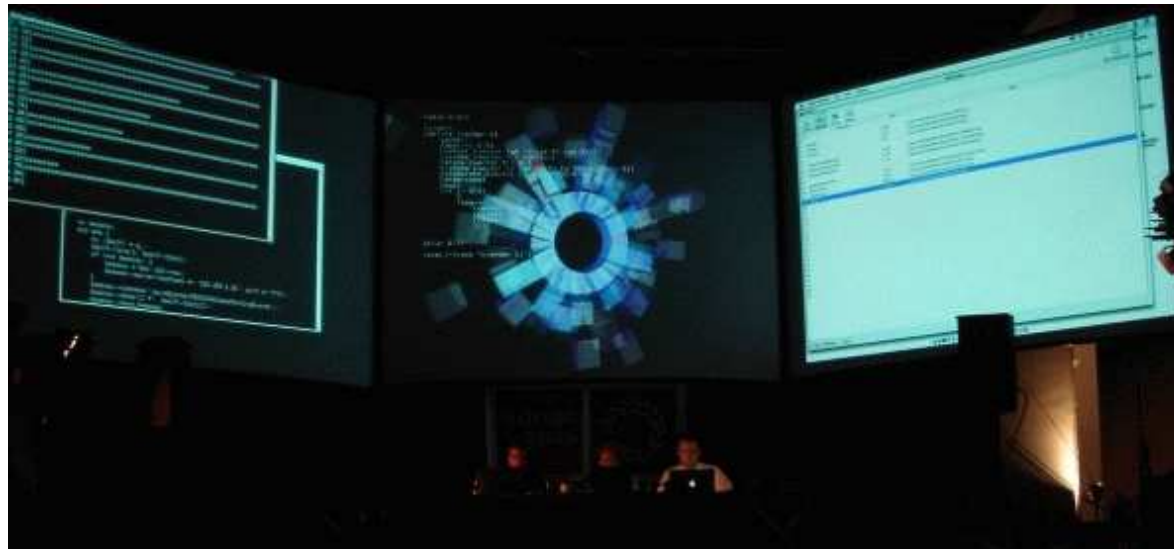
Live coding graphics, using live audio input



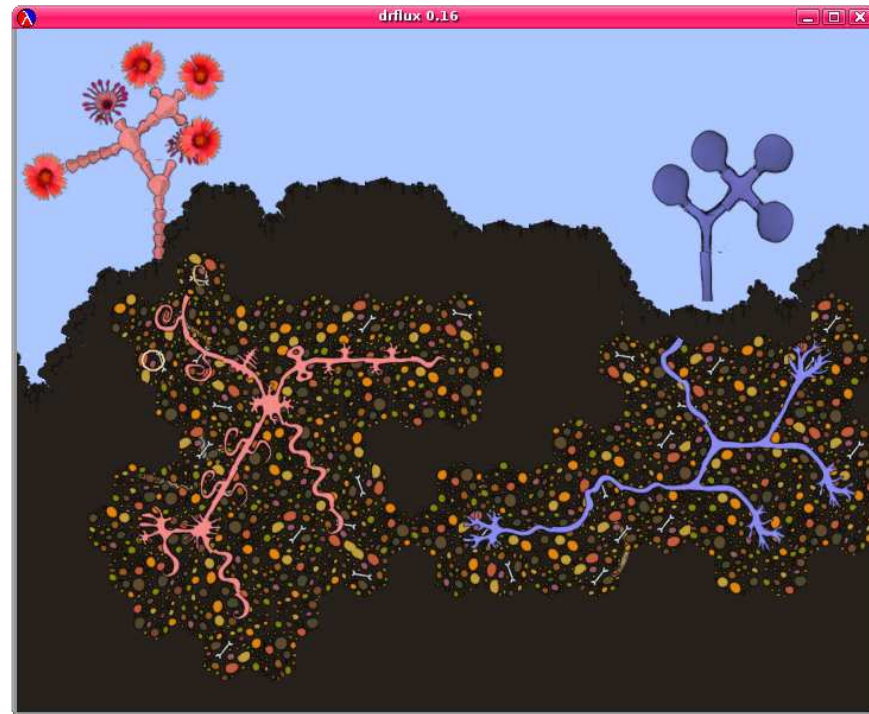
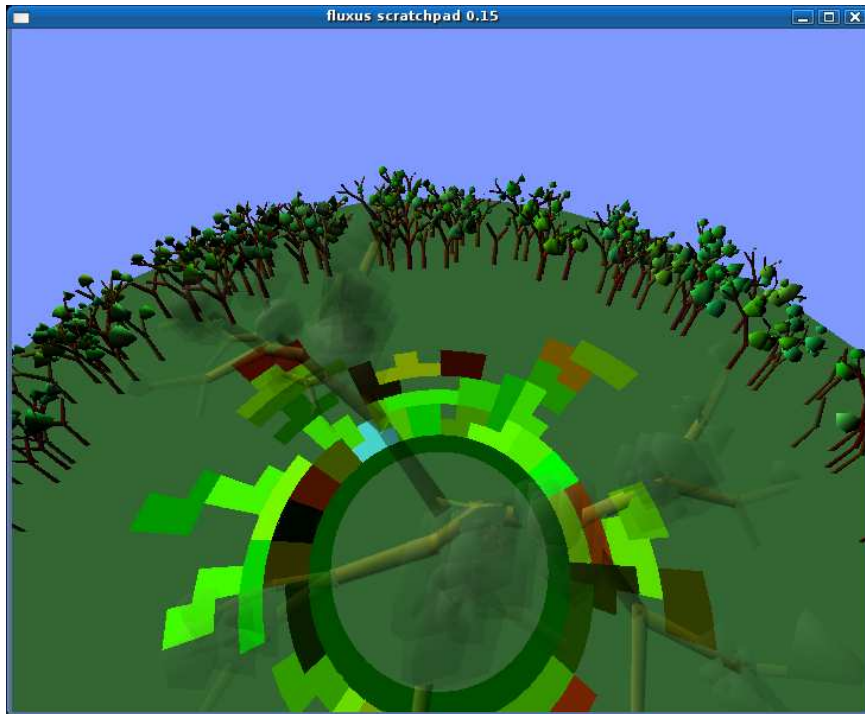
Live coding graphics and audio at the same time



Make installations and performances



slub



Games prototyping

Livcoding

Livcoding

- Performance programming
- Comes from a musical background
- Reaction against the normal laptop performance
- Improvisation
- Showing the audience what you're doing
- Thinking out loud



TOPLAP

- Formed February 2004 in a smokey Hamburg bar
- Now grown to 100's of livecoders
- Role is to promote live coding as a unique art form
- Currently planning a Uk Planetarium Livecoding Tour



TOPLAP MANEFESTO

We demand:

- Give us access to the performer's mind, to the whole human instrument.
- Obscurantism is dangerous. Show us your screens.
- Programs are instruments that can change themselves.
- The program is to be transcended - Artificial language is the way.
- Code should be seen as well as heard, underlying algorithms viewed as well as their visual outcome.
- Live coding is not about tools. Algorithms are thoughts. Chainsaws are tools. That's why algorithms are sometimes harder to notice than chainsaws.

We recognise continuums of interaction and profundity, but prefer:

- Insight into algorithms
- The skillful extemporisation of algorithm as an expressive/impressive display of mental dexterity
- No backup (minidisc, DVD, safety net computer)

We acknowledge that:

- It is not necessary for a lay audience to understand the code to appreciate it, much as it is not necessary to know how to play guitar in order to appreciate watching a guitar performance.
- Live coding may be accompanied by an impressive display of manual dexterity and the glorification of the typing interface.
- Performance involves continuums of interaction, covering perhaps the scope of controls with respect to the parameter space of the artwork, or gestural content, particularly directness of expressive detail. Whilst the traditional haptic rate timing deviations of expressivity in instrumental music are not approximated in code, why repeat the past? No doubt the writing of code and expression of thought will develop its own nuances and customs.

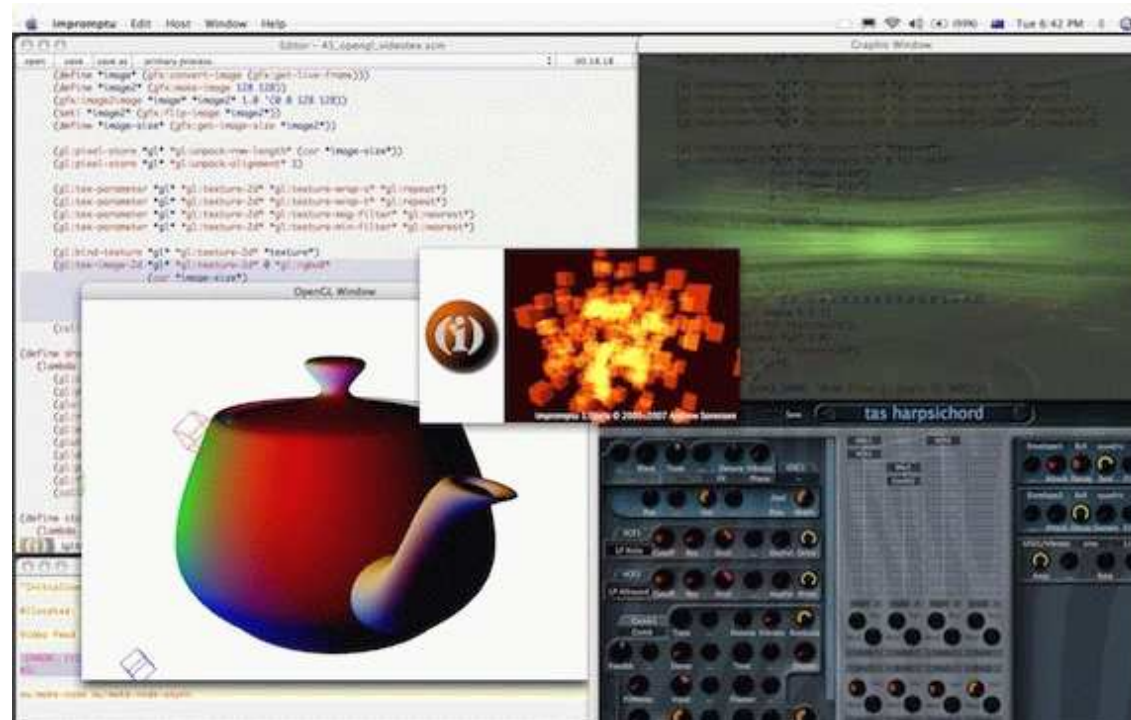
Livecoding & Fluxus

- Fluxus is part of the livecoding movement
- People using it for performance ('no copy paste' from Budapest)
- Fluxus/Supercollider Workshop at the first Livecoding festival in Sheffield
- The movement has greatly influenced fluxus development



Some other livecoding systems

Impromptu



SuperCollider

SuperCollider File Edit Lang UI Format Window Help

SynthDef.help simple performance setup2 SCinSC-c.rtf

SynthDef

definition of a synth and Group layout:

Evaluates a UGen function, generating a ugenGraph that describes all constants, Controls, and UGens that will be used in synthesis.

***new(synthDefName, ugenGraphFunc, rates, prependArgs)**
Create a synthDef instance, evaluate the ugenGraphFunc

synthDefName
string or symbol: "name", 'name', or \name

ugenGraphFunc

SuperCollider Help

Select any of the items listed below by double clicking on it and helpfile. See More-On-Getting-Help for further information.

Essential Topics

More-On-Getting-Help
Server-Architecture
Server-Command-Reference
Tutorial
Writing-Classes
UGen-Plugins
NodeMessaging
Internal-Snooping
ClientVsServer
SC3vsSC2
Order-of-execution
Backwards-Compatibility
MultiChannel
UGens-and-Synths

Language

Intro-to-Objects
Literals
Method-Calls
Assignment
Comments

Group layout:

(diagram drawn in OmniGraffle)

```
// mass production of synths..
(
40.do{ arg i;
  SynthDef("rperc" ++ i.asString, { arg i_bus = 0, amp = 0.1, rate = 1;
    var n = 12;
    var exc, out;
    exc = WhiteNoise.ar * Decay.kr(Impulse.kr(0,0,amp*0.1), rrand(0.2,1.0))
    out = Klank.ar([
      {exprand(100.0, 10000.0)}.dup(n),
      { rrand(0.1,1.0) }.dup(n),
      {exprand(0.05,1.0)}.dup(n)
    ], exc, rate);
    DetectSilence.ar(out, 0.0001, 0.1, 2);
    Out.ar(i_bus, PanAz.ar(4, out, rrand(-1.0,1.0)));
  }).load(s);
});

// a process to use them.
(
var s;
s = Server.local;
Task{
  var dur=0.2, inst = \rperc0, amp = 0.05;
  inf.do{
    if (0.3.coin, {
      inst = "rperc" ++ 40.rand.asString;
      amp = exprand(0.02,0.2) * 0.5;
    });
    s.sendBundle(0.2, [\s_new, inst, -1, 0, 0, \amp, amp, \rate, exprand(0.02,0.2), dur]);
    if (dur.coin, { dur = [ 0.075, 0.1, 0.15, 0.2, 0.3, 0.4, 0.6, 0.8, 1.6 ].chose; });
    dur.wait;
  });
});
```

```
BufRd : MultiOutUGen {
  *ar { arg numChannels, bufnum=0,
    ^this.multiNew('audio', numChannels, bufnum)
  }
  *kr { arg numChannels, bufnum=0,
    ^this.multiNew('control', numChannels, bufnum)
  }
  init { arg argNumChannels ... the inputs = theInputs;
    ^this.initOutputs(argNumChannels)
  }
  argNamesInputsOffset { ^2 }
  checkInputs {
    if (rate == 'audio' and: {
      ^("phase input is not a float")
    });
    nil
  }
}

BufWr : UGen {
  *ar { arg inputArray, bufnum=0,
    this.multiNewList(['audio', 'control'], bufnum, phase, loop) ++ inputArray.asArray
  }
  *kr { arg inputArray, bufnum=0, phase=0.0, loop=1.0;
    this.multiNewList(['control', 'bufnum', phase, loop] ++ inputArray.asArray)
  }
  checkInputs {
    if (rate == 'audio' and: {
      ^("phase input is not a float")
    });
    nil
  }
}
```

```
LinRand : UGen {
  // linear distribution
  // if minmax <= 0 then skewed toward lo
  // else skewed toward hi.
  *new { arg lo = 0.0, hi = 1.0, minmax = 0;
    ^this.multiNew('scalar', lo, hi, minmax)
  }
}

NRand : UGen {
  // sum of N uniform distributions.
  // n = 1 : uniform distribution - same as Rand
  // n = 2 : triangular distribution
  // n = 3 : smooth hump
  // as n increases, distribution converges towards gaussian
  *new { arg lo = 0.0, hi = 1.0, n = 0;
    ^this.multiNew('scalar', lo, hi, n)
  }
}

ExpRand : UGen {
  // exponential distribution
  *new { arg lo = 0.01, hi = 1.0;
    ^this.multiNew('scalar', lo, hi)
  }
}

TEExpRand : UGen {
  // uniform distribution
}
```

localhost:57110

Boot K localhost:57110

Avg CPU: 0.9 %
UGens: 7
Groups: 2

localhost:57110

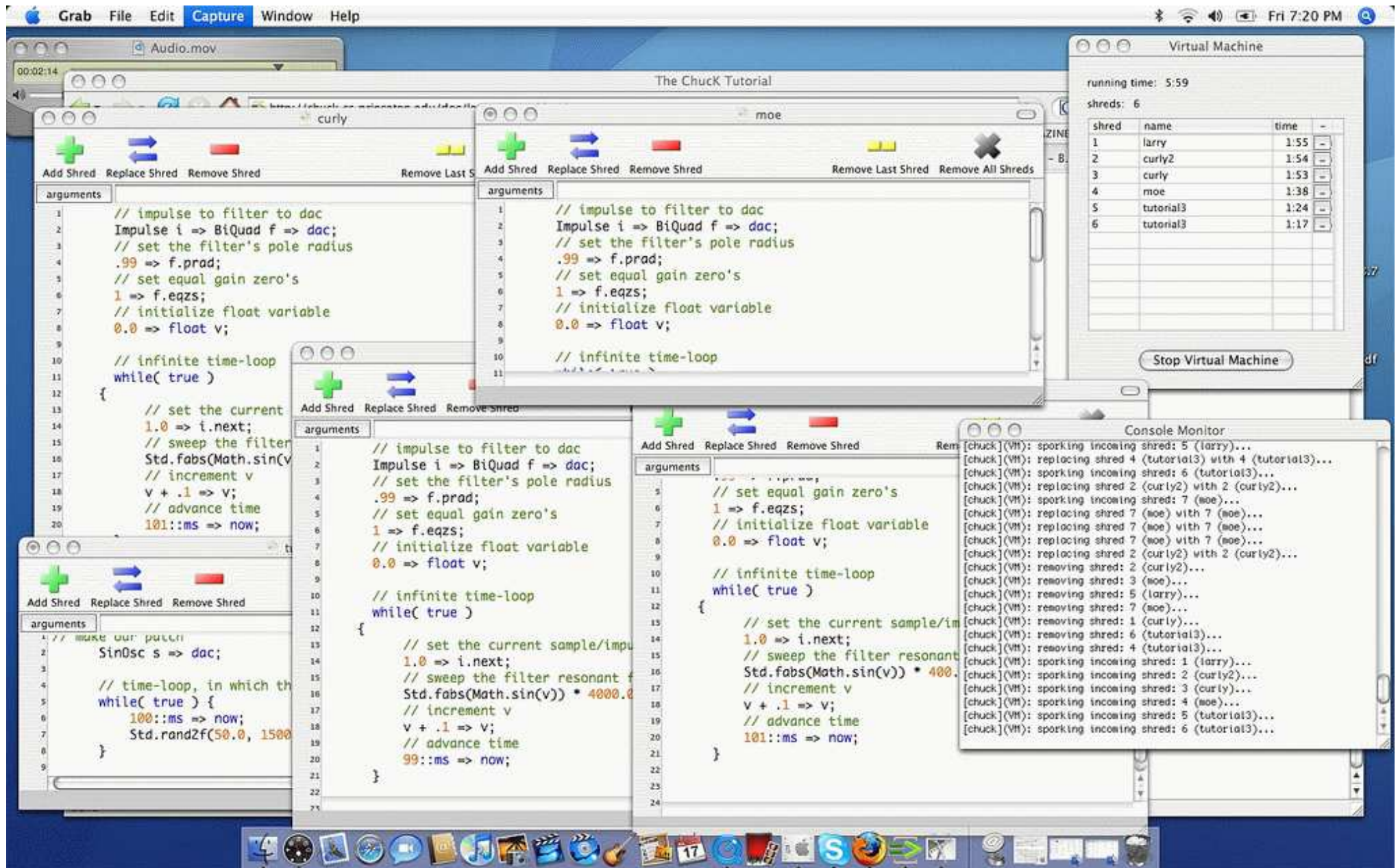
Boot K localhost:57110

Avg CPU: 7.1 %
UGens: 50
Groups: 1

tree

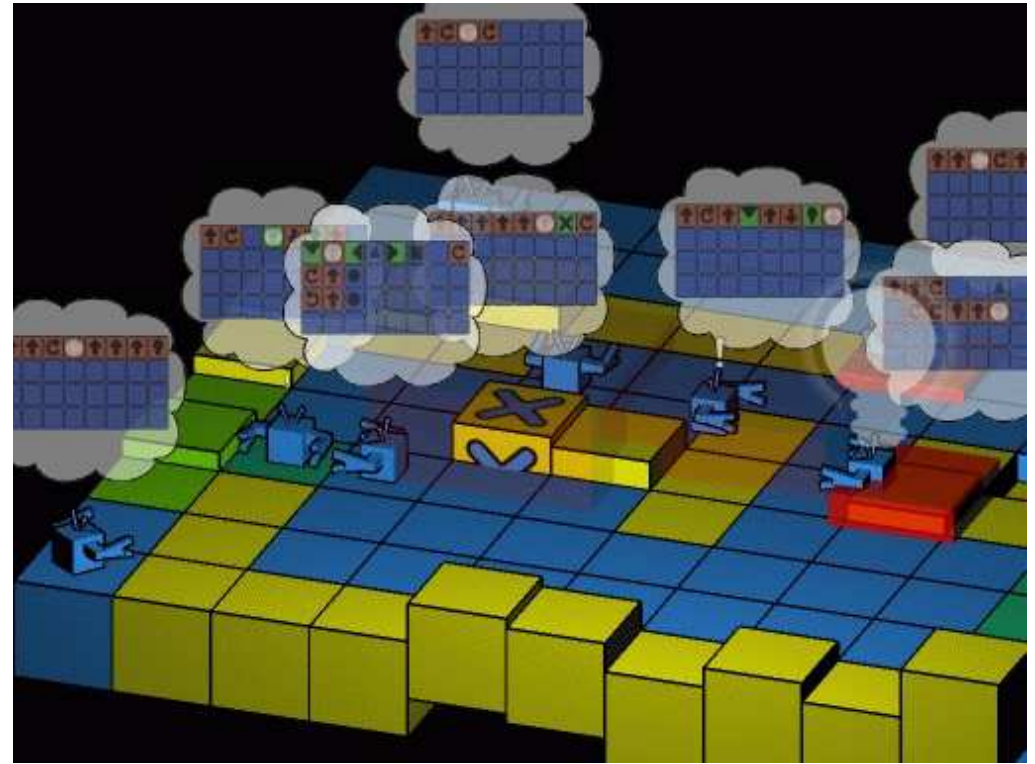
tree2

Chuck



Gamepad Livecoding

- Live coding doesn't have to be about text editors
- Live coding doesn't have to be hard
- Making fun, simplified languages



About Scheme

Scheme

- Invented in 1975 by Jerald J. Sussman and Guy L. Steel Jr.
- A simplified dialect of Lisp
- A "high level" language
- A language for learning programming
- Influences modern languages such as Python and C#

... if you are used to a C based language
it can seem very strange

```
(define (factorial n)
  (if (zero? n)
      1
      (* n (factorial (- n 1)))))
```

Scheme is good for live coding

- Functional
- Minimal syntax
- Maximum complexity out of minimum code

