

# Fluxus: Scheme Livecoding

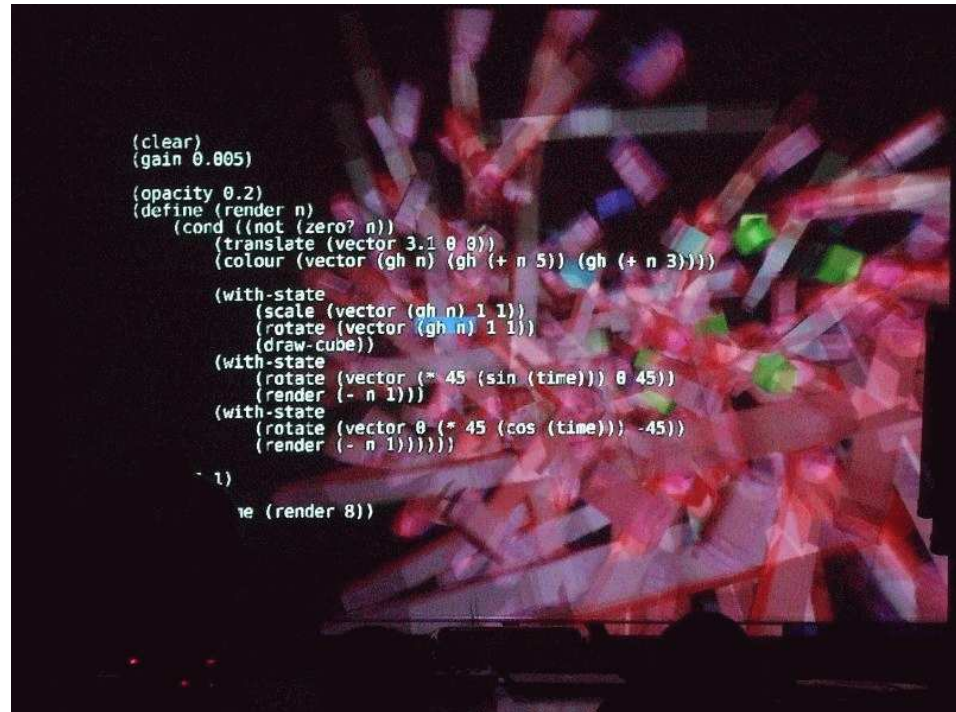
Dave Griffiths

# Overview

- Fluxus introduction
- What it's built from
- Why Scheme
- Livecoding
- How it works - some examples
- Functional Reactive Programming
- Gamepad Livecoding

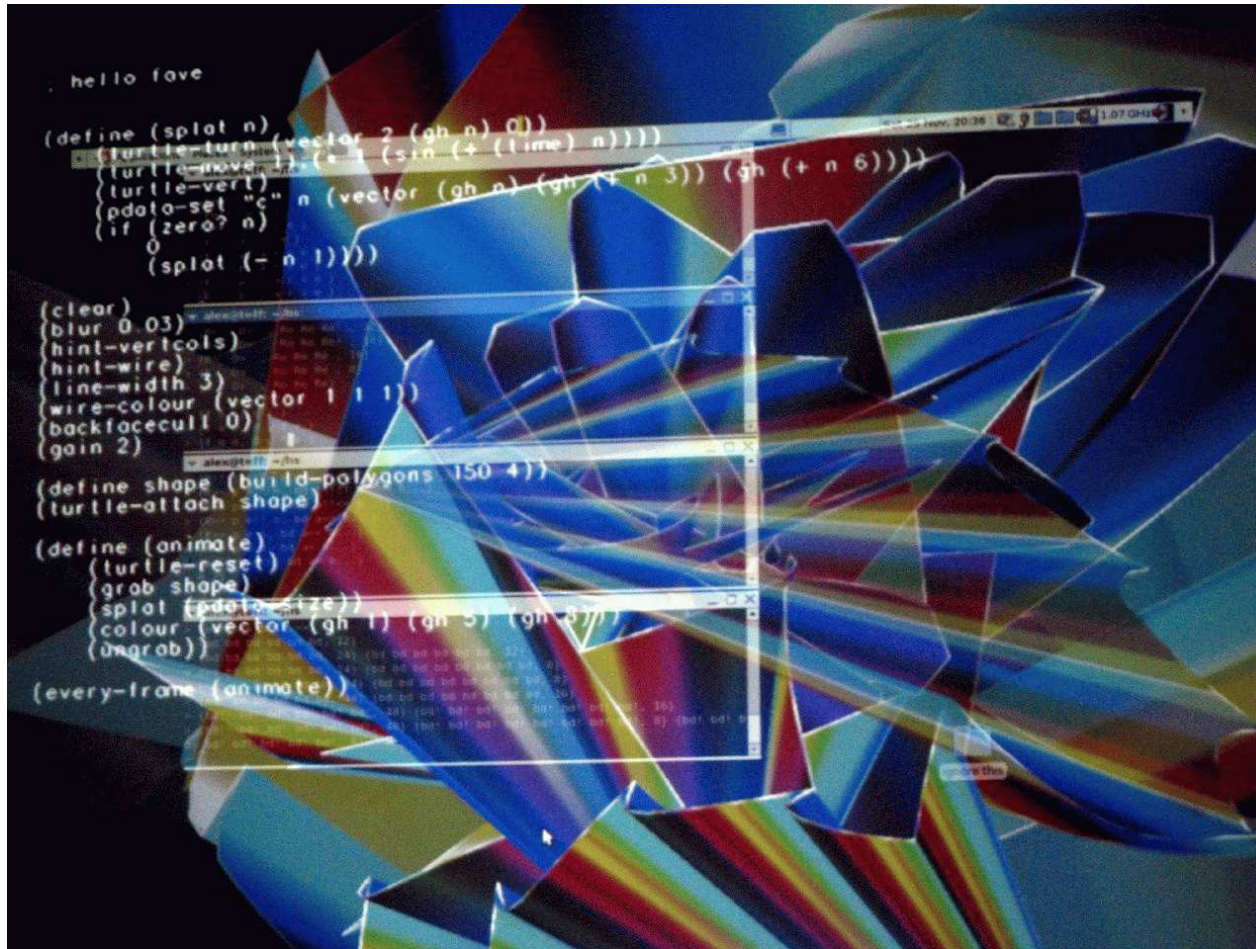
# What is fluxus?

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

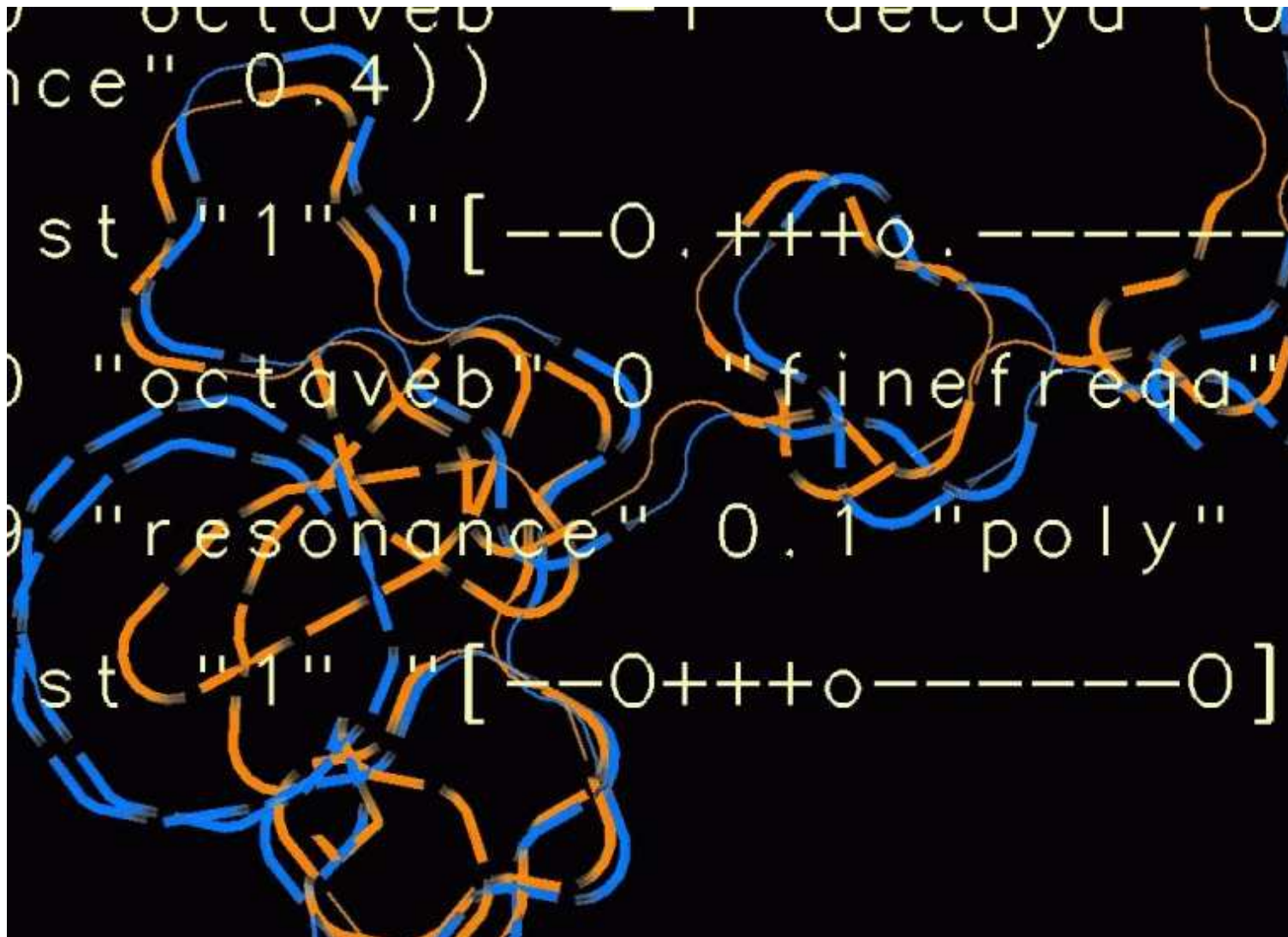


## **Quick demo**

**I use fluxus for...**



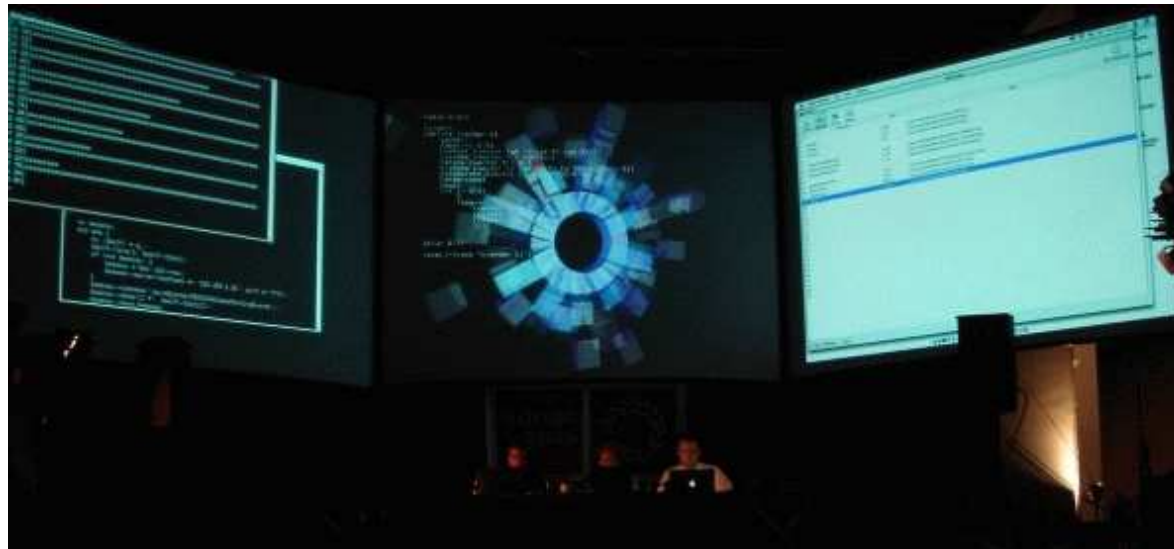
Live coding graphics, using live audio input



Live coding graphics and audio at the same time



8



slub

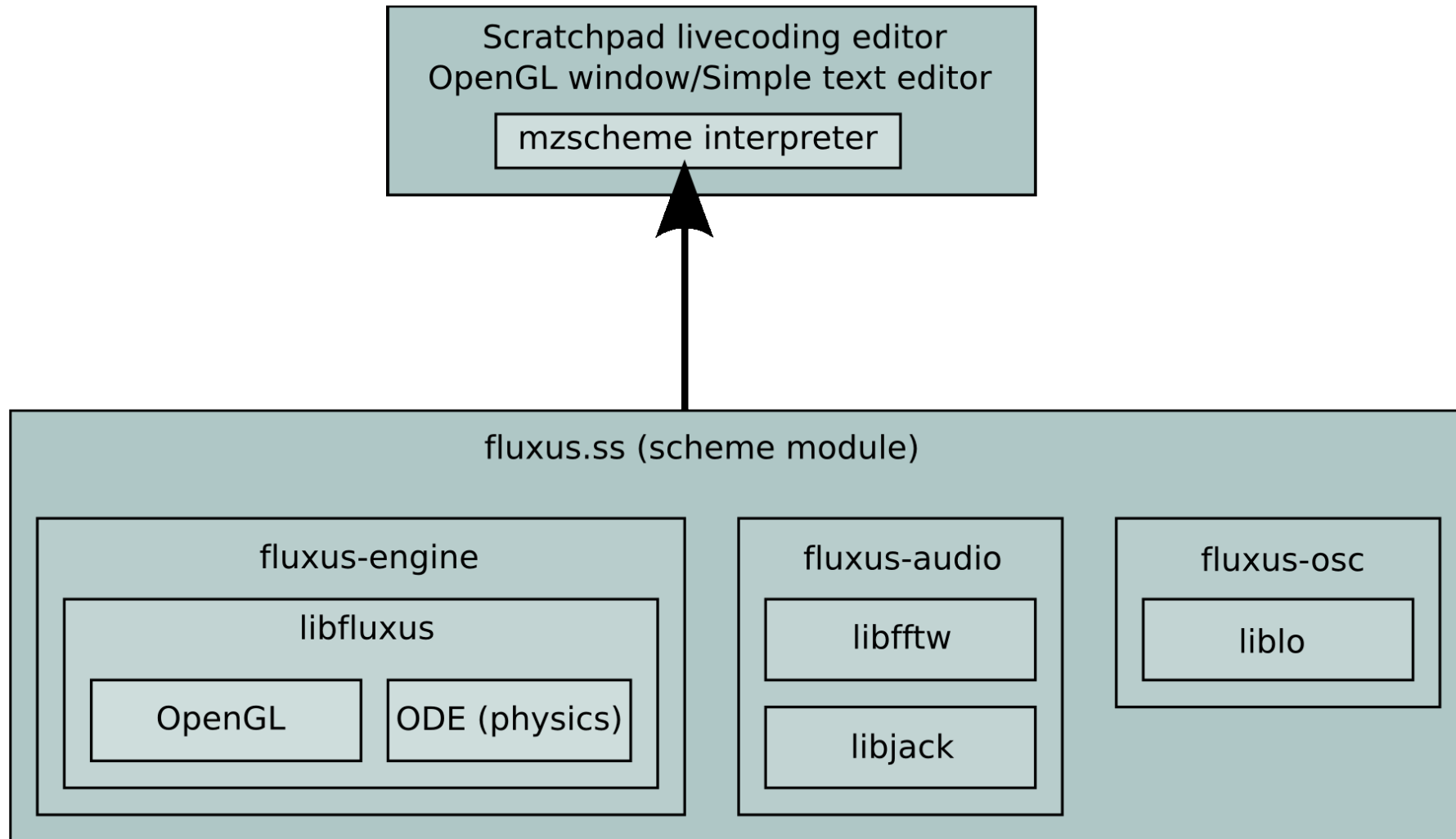
**What's inside?**

# Boring Feature List

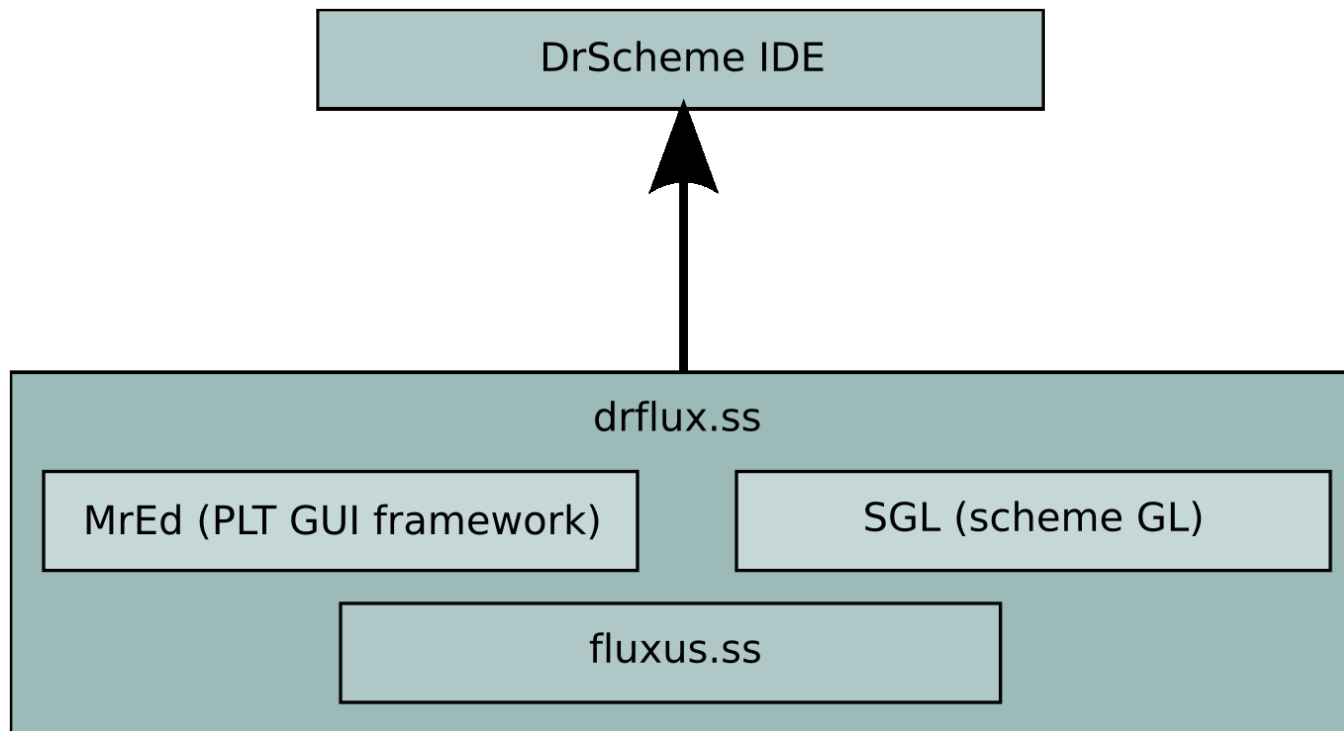
- Immediate mode drawing
- Scenegraph
- Primitives
  - Polys
  - Particles
  - NURBS patches
  - Blobbies (implicit surfaces)
  - Pixels (procedural texture access)
- Unified access to primitive data (vertex arrays, texture data)
- More advanced stuff
  - GLSL Hardware shading
  - ODE physics
  - Shadows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



# Architecture



# DrScheme integration



# DrScheme integration

The screenshot displays the DrScheme environment integrated with a terminal and a graph visualization.

**Terminal (xterm) Output:**

```
[~]> oscjoy 127.0.0.1:4444
-> OSC target is 127.0.0.1:4444
-> Local UDP port is 8000
-> Update rate is 100Hz
-> There are 1 joysticks attached
-> Joystick 0: Saitek PLC Saitek P2600 Rumble Force Pad was opened.
-> Watching joystick 0: (Saitek PLC Saitek P2600 Rumble Force Pad)
-> Joystick has 6 axes, 0 hats, 0 balls, and 12 buttons
drflux 0.14
```

**DrScheme Script (daisy.scm):**

```
(define (delete)
  (destroy root)
  (inner (void) delete))

(define (balance this-id graph)
  (define (repulse)
    (foldl
      (lambda (vertex-item vec)
        (cond
          ((eq? (car vertex-item) this-id) vec)
          (else
           (send graph inc-repulses) ; record this calculation
           (let ((v (vsub pos (send (cdr vertex-item) get-position))))
             (vadd vec (vmul (vnormalise v) (/ 1 (vmag v)))))))
        (vector 0 0 0)
        (send (send graph get-vertices) get-row-list)))
    (vector 0 0 0)
    (send (send graph get-vertices) get-row-list)))

  (define (attract)
    (foldl
      (lambda (vertex-id vec)
        (vadd vec (vsub (send (send (send graph get-vertices) get vertex-id) get-position)
          (vector 0 0 0))
        (vector 0 0 0)
        (send (send graph get-vertices) get-row-list)))
      (vector 0 0 0)
      (send (send graph get-vertices) get-row-list)))

  (send graph get-connected-outvert-ids this-id))(newline)
  (display count)(newline)
  (display avg)(newline)

  (with-primitive root
    (translate avg)
    (set! pos (vtransform (vector 0 0 0) (get-transform)))
    (set! position pos)))

(Welcome to DrScheme, version 370 [3m].
Language: Pretty Big \(includes MrEd and Advanced Student\) custom.
none
(43)
0
#(-0.00209599989466369 15 -0.00018200009071733803 0.0)
(5)
0
#(0.00012799978139818264 -0.004311999771744013 0.0)
(55)
1
#(0.000000000000000 0.000000000000000 0.0)

Programming language:
Pretty Big (includes MrEd and Advanced Student) custom
```

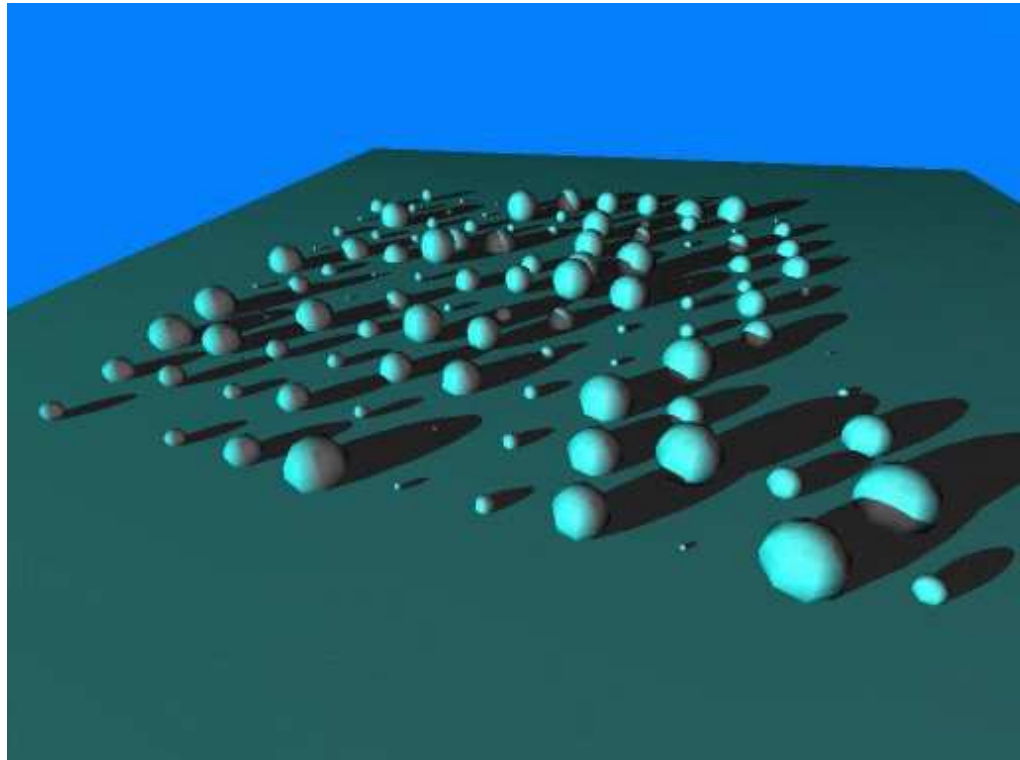
# DrScheme integration

- Much better editor
- Debugger
- Profiler
- Syntax highlighting
- Less control over OpenGL
  - No GLSL shader support
  - No stencilmap shadows
  - Slower
- Not suitable for livecoding performance

Why Scheme?

# Fluxus philosophy

- 'Creative' code in Scheme
- OpenGL grunt work in C++



# Scheme for creative code

- Fast feedback (use an interpreter)
- Expressive power
- Few keypresses needed to get interesting results
  - Functional roots
  - Use of macros to shape the language
  - Dynamic typing
- Lots of interesting research going on with Scheme



Livcoding

# Livecoding

- Performance programming
- Mainly a musical field
- Reaction against the normal laptop performance
- Showing the audience what you're doing



# 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



## 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.

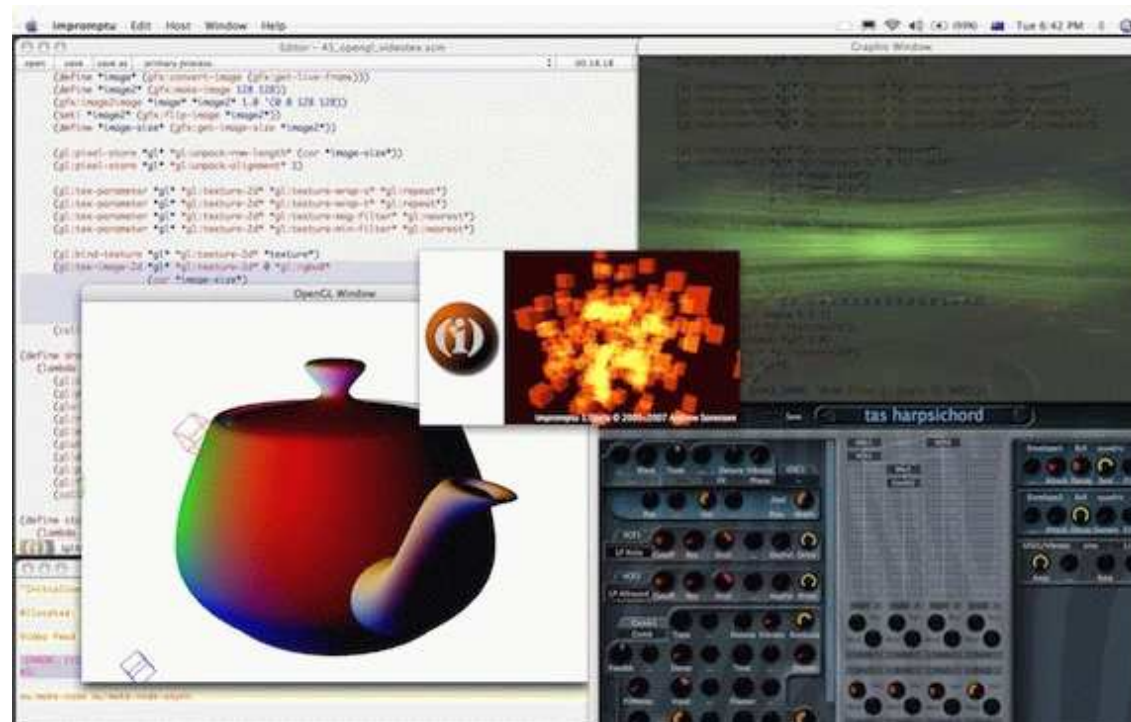
# Livecoding & Fluxus

- Fluxus is part of the livecoding movement
- People using it for performance ('no copy paste' from Budapest)
- Fluxus/Supercollider Workshop at the LOSS 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 arc

Evaluates a UGen function, generating a ugenGraph that desc All constants, Controls, and UGens that will be used in synths

**\*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-Compati  
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;
  });
});
)

(
var w, startButton, cl
var sendConfig, cmdPer
var numTracks, channel
var totalChannels, tot
var mixerGroupID, trac
var series, trackGroup
var finalMixSynthDef;
var inputBus, outputBu
var ampControlBus, tra
var server;
var b, ez;

```

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

BufWr : UGen {
  *ar { arg inputArray, bufnum=0,
    this.multiNewList(['audio',
    loop] ++ inputArray.asArray)
    ^inputArray
  }
  *kr { arg inputArray, bufnum=0, phase=0.0, loop=1.0;
    this.multiNewList(['control', bufnum, phase,
    loop] ++ inputArray.asArray)
    ^inputArray
  }
  checkInputs {

```

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

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)
  }
}

TEpRand : UGen {
  // uniform distribution

```

localhost

Boot K localhost

Avg CPU: 0.9 %

UGens: 7

Groups: 2

internal s

Boot K intern

Avg CPU: 7.1 %

UGens: 50

Groups: 1

tree

tree2

# Chuck

The screenshot displays the Chuck programming environment on a Mac OS X desktop. Several windows are open, each representing a different 'shred' (a unit of audio processing). The windows are titled 'curly', 'moe', and 'tutorial3'. Each window contains a code editor with Chuck code and a control panel with buttons like 'Add Shred', 'Replace Shred', and 'Remove Shred'.

The 'Virtual Machine' window shows the running time as 5:59 and a list of shreds with their names and times:

shred	name	time
1	larry	1:55
2	curly2	1:54
3	curly	1:53
4	moe	1:38
5	tutorial3	1:24
6	tutorial3	1:17

The 'Console Monitor' window shows a log of events, including spawning and removing shreds:

```
[chuck](VM): spawning incoming shred: 5 (larry)...
[chuck](VM): replacing shred 4 (tutorial3) with 4 (tutorial3)...
[chuck](VM): spawning incoming shred: 6 (tutorial3)...
[chuck](VM): replacing shred 2 (curly2) with 2 (curly2)...
[chuck](VM): spawning incoming shred: 7 (moe)...
[chuck](VM): replacing shred 7 (moe) with 7 (moe)...
[chuck](VM): replacing shred 7 (moe) with 7 (moe)...
[chuck](VM): replacing shred 2 (curly2) with 2 (curly2)...
[chuck](VM): removing shred: 2 (curly2)...
[chuck](VM): removing shred: 3 (moe)...
[chuck](VM): removing shred: 5 (larry)...
[chuck](VM): removing shred: 7 (moe)...
[chuck](VM): removing shred: 1 (curly)...
[chuck](VM): removing shred: 6 (tutorial3)...
[chuck](VM): removing shred: 4 (tutorial3)...
[chuck](VM): spawning incoming shred: 1 (larry)...
[chuck](VM): spawning incoming shred: 2 (curly2)...
[chuck](VM): spawning incoming shred: 3 (curly)...
[chuck](VM): spawning incoming shred: 4 (moe)...
[chuck](VM): spawning incoming shred: 5 (tutorial3)...
[chuck](VM): spawning incoming shred: 6 (tutorial3)...
```

Fluxus code

# Scene description

- Describing 3D scene and behaviours
- Simple example:

```
(define (draw-my-scene)  
      (draw-cube))  
(every-frame (draw-my-scene))
```

# Scene description

- It's essentially a state machine:

```
(translate (vector 0 1 0))  
(scale (vector 0.1 0.1 2))  
(rotate (vector 45 0 0))  
(colour (vector 1 0 0))  
(texture (load-texture "brick.png"))  
(shader "blinn.vert.glsl" "blinn.frag.glsl")  
(draw-cube)  
...  
(draw-sphere)  
(draw-torus)
```

## Example Demos

# Mutable state

- Everything so far relies on mutable state
- Some of this is for performance
- Conventional way of using scene graphs
  - Init step: build scene graph
  - Every frame step: update scene graph
- Complexity starts to bite with big scenes
- Is there a better way?

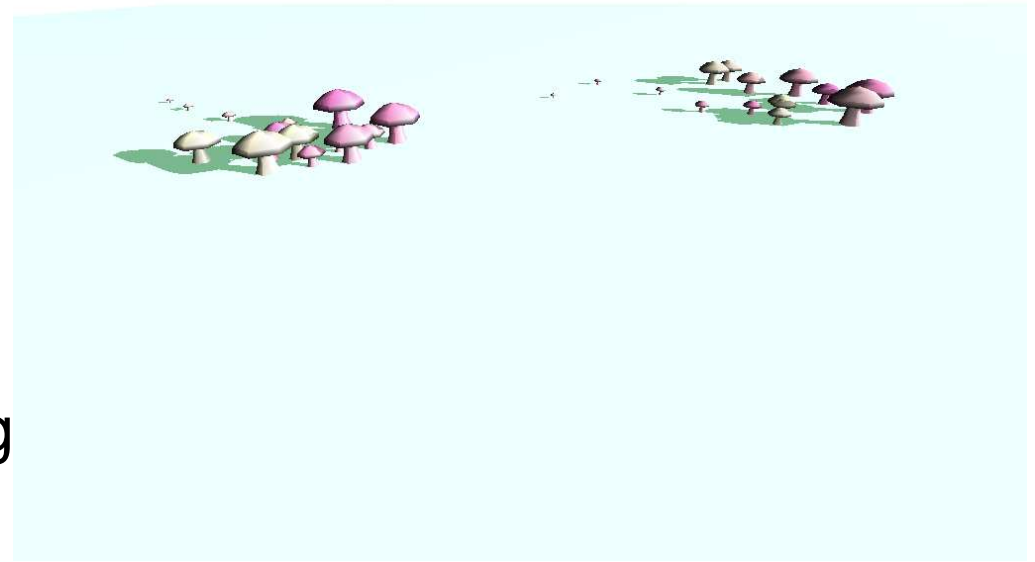
# Functional Reactive Programming

- Applying declarative programming for reactive systems
- First class behaviours and events
- Examples:
  - Fran - Functional reactive animation (Haskell)
  - Yampa - Haskell
  - Flapjax - Javascript
  - FrTime - PLT Scheme
- Avoids the complexity of state machines for behaviour
- More scalable
- Starting to be looked at here and there for games



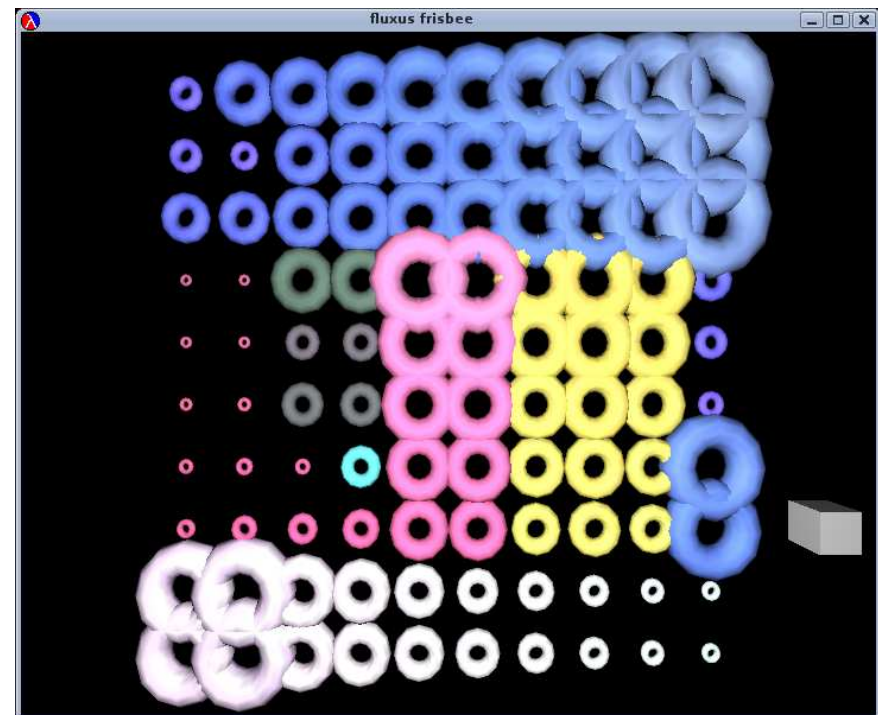
# Frisbee

- FRP in fluxus
- Based on FrTime - by Gregory Cooper
- Trying to build a game engine on top of FrTime
- Could be a more expressive language for fluxus
- Easier to teach game programming
- Increase what is possible while livecoding



# Frisbee disclaimers

- Frisbee is very much work in progress
- FrTime is still undergoing optimisation work
- A few glitches still
- We're working on it...



## Frisbee Demo

# Gamepad Livecoding

# Gamepad Livecoding

- Make watching live coding a bit more accessible
- Live coding doesn't have to be about text editors
- Live coding doesn't have to be hard
- Making fun, simplified languages

