fluxus & Scheme

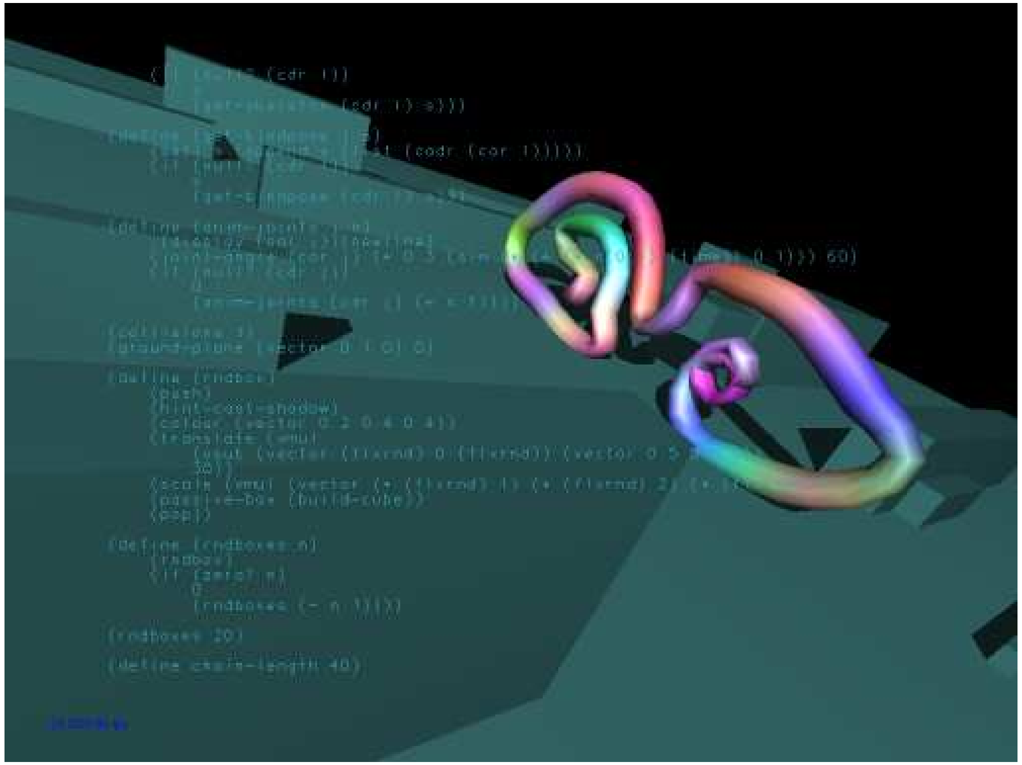
Dave Griffiths

Fluxus facts

- A 3D rendering engine
- Livecoding interpreter
- Uses mzscheme and OpenGL
- Free Software
- Works on Linux and sometimes OSX
- Version 0.1 released Tuesday, August 5th 2003 17:29 (courtesy of freshmeat)
- Current stable version 0.12

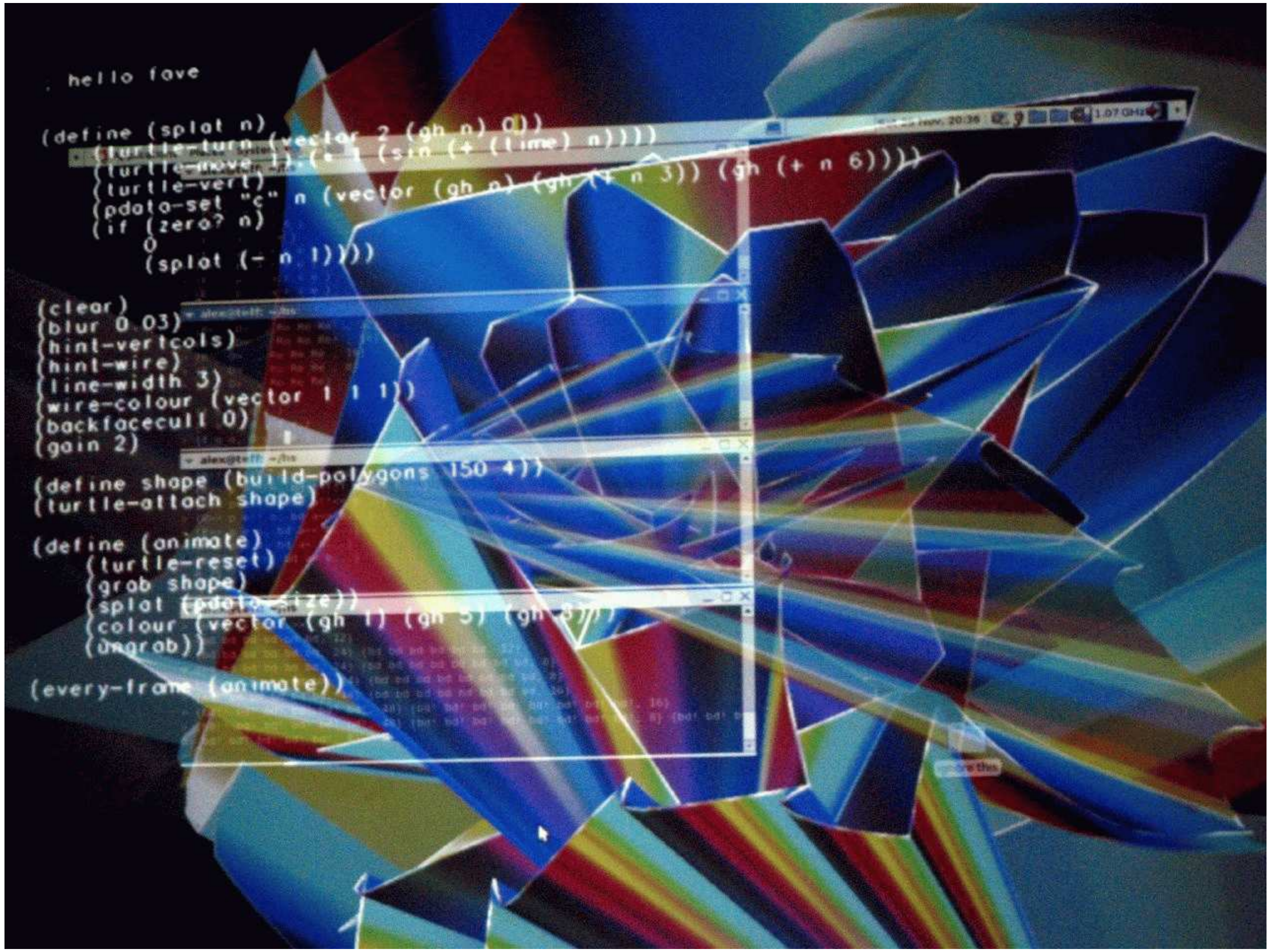
Boring Feature List

- 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



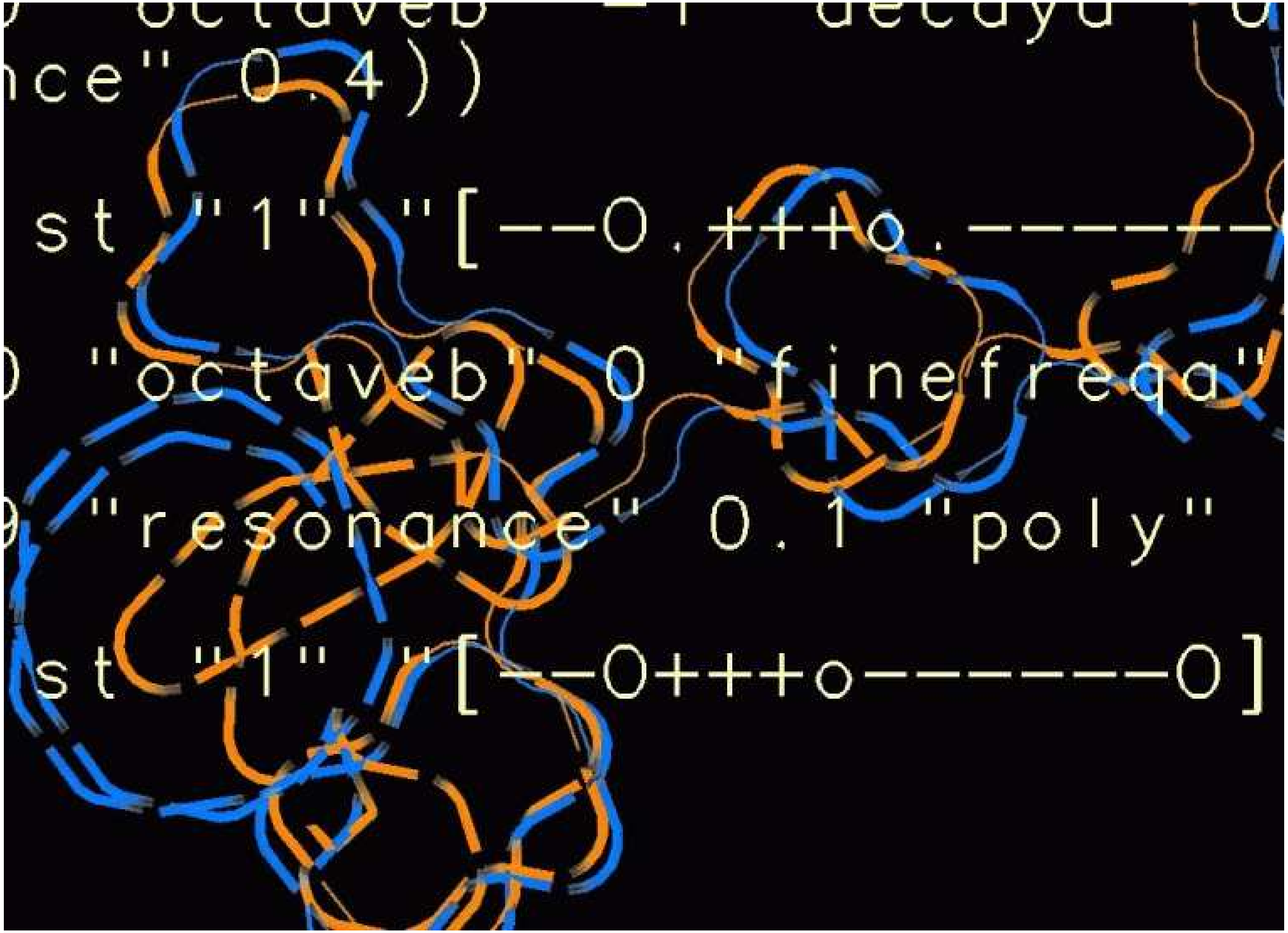
```
  (if [illegible] (cdr l))
      (set-[illegible] (cdr l) a)))
(define [illegible] l)
  [illegible] (cadr (car l)))))
  (if (null? (cdr l))
      0
      (set-[illegible] (cdr l) a)))
(define (anim-joints l n)
  [illegible]
  (joint-angle (car l) (* 0.5 [illegible] 1))) 60)
  (if (null? (cdr l))
      0
      (anim-joints (cdr l) (+ n 1))))

(collisions 1)
(ground-plane (vector 0 1 0) 0)

(define (rndbox)
    (push)
    (hint-cast-shadow)
    (colour (vector 0.2 0.4 0.4))
    (translate (vmul
        (vadd (vector (flxrnd) 0 (flxrnd)) (vector 0 0.5 [illegible]
        30)))
    (scale (vmul (vector (* (flxrnd) 1) (* (flxrnd) 2) (* [illegible]
    (passive-box (build-cube))
    (pop))

(define (rndboxes n)
    (rndbox)
    (if (zero? n)
        0
        (rndboxes (- n 1))))

(rndboxes 20)

(define chain-length 40)
```

I use fluxus for...



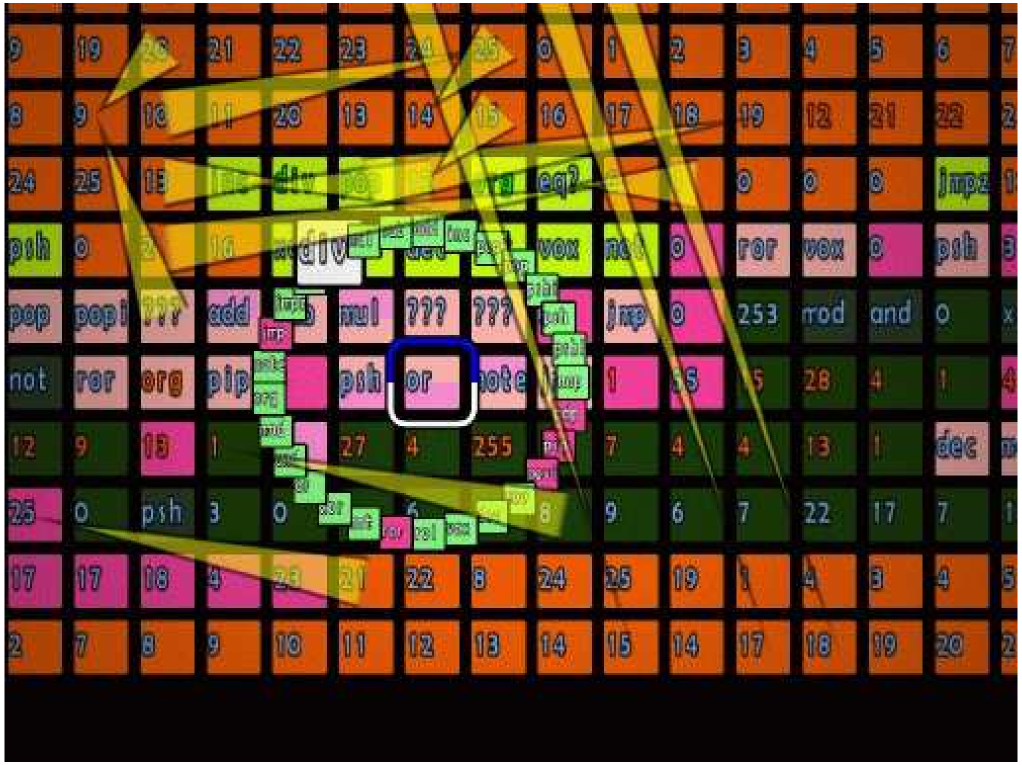
```
; hello fave

(define (splat n)
    (turtle-turn (vector 2 (gh n) 0))
    (turtle-move (* 1 (sin (+ (time) n))))
    (turtle-vert)
    (pdata-set "c" n (vector (gh n) (gh (+ n 3)) (gh (+ n 6))))
    (if (zero? n)
        0
        (splat (- n 1))))

(clear)
(blur 0.03)
(hint-vertcols)
(hint-wire)
(line-width 3)
(wire-colour (vector 1 1 1))
(backfacecull 0)
(gain 2)

(define shape (build-polygons 150 4))
(turtle-attach shape)

(define (animate)
    (turtle-reset)
    (grab shape)
    (splat (pdata-size))
    (colour (vector (gh 1) (gh 5) (gh 8)))
    (ungrab))

(every-frame (animate))
```

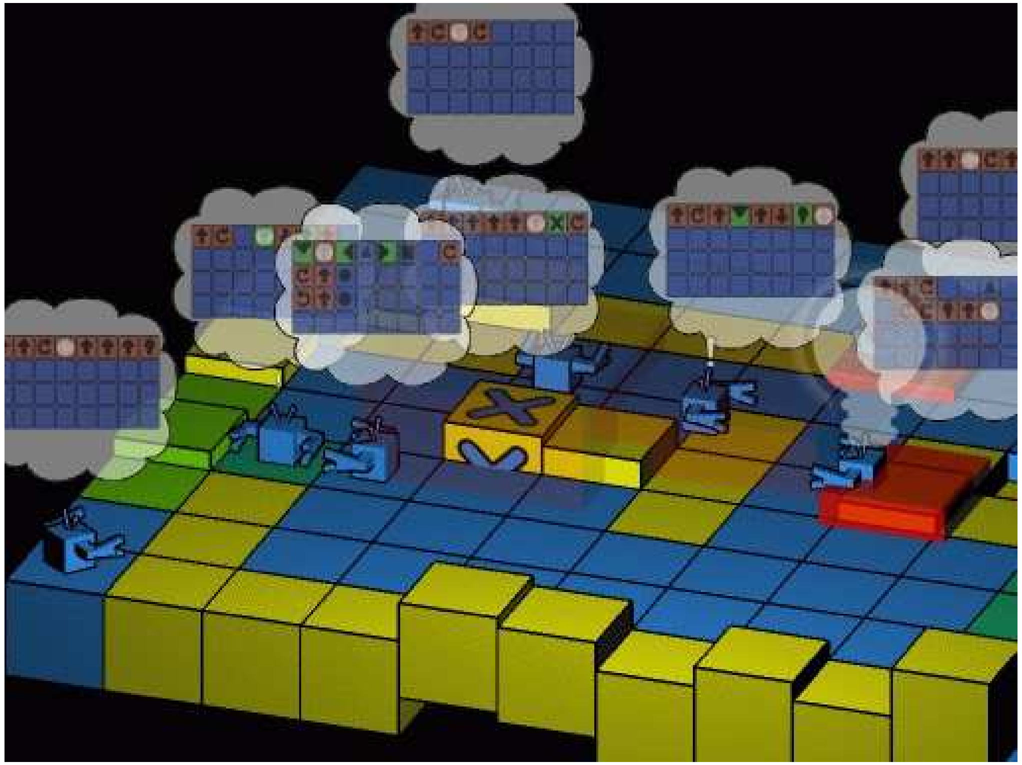
Live coding graphics, using live audio input



```
0 "octaveb" -1 "decaya" 0
nce" 0.4))

st "1" "[--0.+++o.-------

0 "octaveb" 0 "finefreqa"

9 "resonance" 0.1 "poly"

st "1" "[--0+++o-------0]
```

Live coding graphics and audio at the same time





As a framework for developing new livecoding languages

Scheme

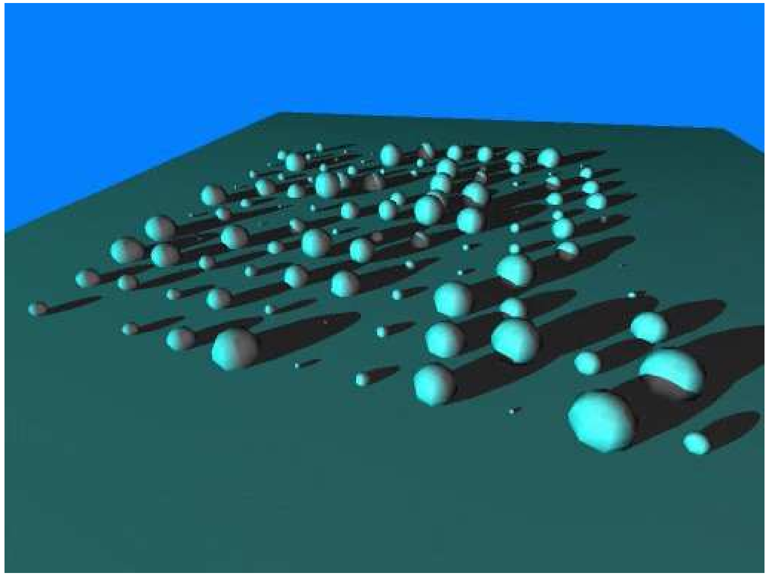
- Invented in 1975 by Jerald J. Sussman and Guy L. Steel Jr.
- A simplified dialect of Lisp
- A "high level" language
- A language for learning programming
- Influences modern languages such as Python and C#

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

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

Scheme is good for live coding

- Minimal syntax
- Functional
- Minimum code, maximum complexity



but can take a bit of getting used to...

```
(define !
  (lambda (n)
    ((lambda (n)
       ((n (lambda (x) (+ x 1))) 0))
     ((lambda (n)
        ((lambda (p)
           (p (lambda (x)
                (lambda (y)
                  y))))
         ((n (lambda (p)
               (((lambda (x)
                   (lambda (y)
                     (lambda (fun)
                       ((fun x) y))))
                 ((lambda (n)
                    (lambda (f)
                      (lambda (x)
                        (f ((n f) x)))))
                  ((lambda (p)
                     (p (lambda (x)
                          (lambda (y)
                            x))))
                                  p)))
                (((lambda (x)
                    (lambda (y)
                      ((y ((lambda (x)
                             (lambda (y)
                               ((y (lambda (n)
                                     (lambda (f)
                                       (lambda (x)
                                         (f ((n f) x))))))
                                                           x)))
                                                                x))
                       (lambda (f)
                         (lambda (x)
                           x)))))
                  ((lambda (p)
                     (p (lambda (x)
                          (lambda (y)
                            x))))
                                  p))
                 ((lambda (p)
                    (p (lambda (x)
                         (lambda (y)
                           y))))
                                 p)))))
          (((lambda (x)
              (lambda (y)
                (lambda (fun)
                  ((fun x) y))))
            ((lambda (n)
```

Recursion

"Fishes and Scales" M.C. Escher

- Recursion is the only way to loop
- Elegant but hard to explain