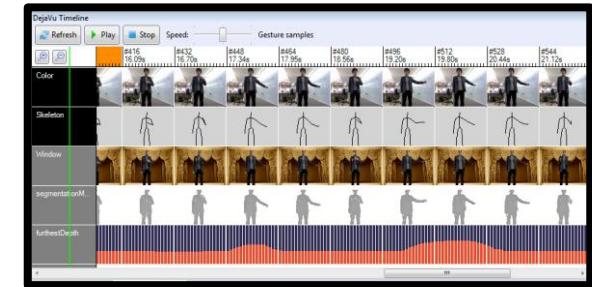
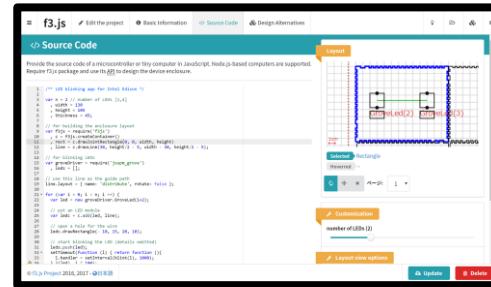
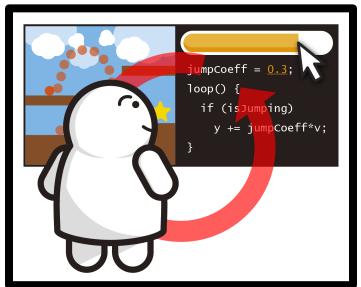


User Interfaces for Live Programming

Jun Kato

<https://junkato.jp>

Researcher, 



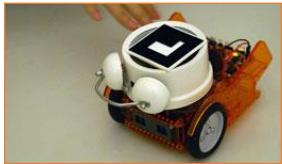
LIVE 2017 Keynote, 10/24/2017



Research Topic

Computer Science (Human-Computer Interaction, Programming Language)

Phybots



DejaVu



ACM UIST'12

Picode



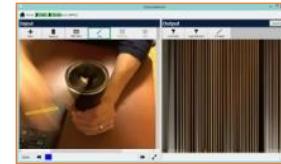
ACM CHI'13

It's Alive!



ACM PLDI'13

VisionSketch



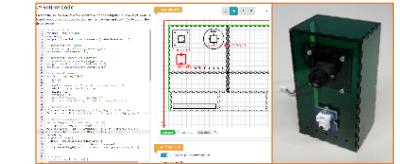
GI'14

TextAlive



ACM CHI'15

f3.js



ACM DIS'17

- Created **Tools and Environments** for **Creativity/Productivity Support**
- **Application Domains:** Prototyping, Physical Computing, Computer Vision, Robots, Internet of Things, Animation Authoring, ...
- Founded **SIGPX** (SIG on Programming Experience) <https://sigpx.org/en>

P(x)

SIGPX

A group of **researchers/engineers/teachers** in Japan, studying ...

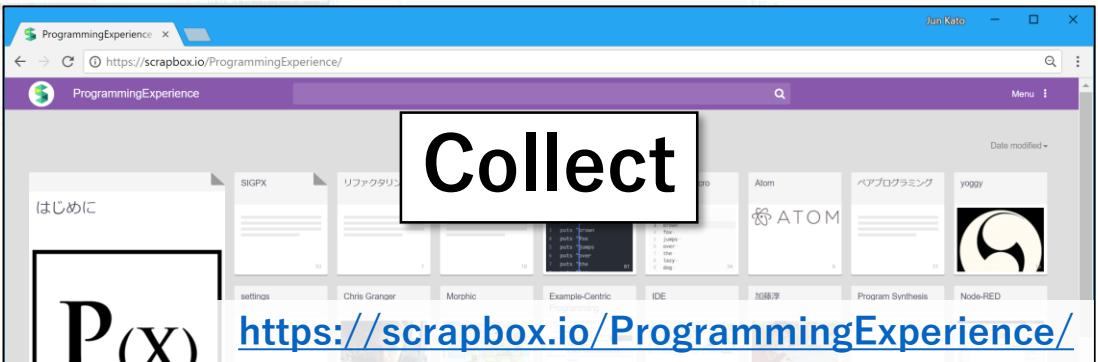
<https://sigpx.org/en>

Programming Experience in the intersection of **HCI/PL/SE**

Meet & Discuss



Collect



Publish

PX Special Issue in IPSJ Journal (Nov 2017)
Emerging Research on Programming Experience:
From Language Design to Industrial Applications

User Interfaces for Live Programming

implementation

analysis

Visketch Canvas

Today, I'm going to talk about ...

- What is Live Programming?
- UIs for Live Programming with end-users
- UIs for Live Programming of this material world
- UIs for Live Programming with time travel
- Live Programming as User Interface research

It is about ...

- Showcase of user interfaces for programming
- Not only my work but also others' notable work
- Discussion on live programming system design

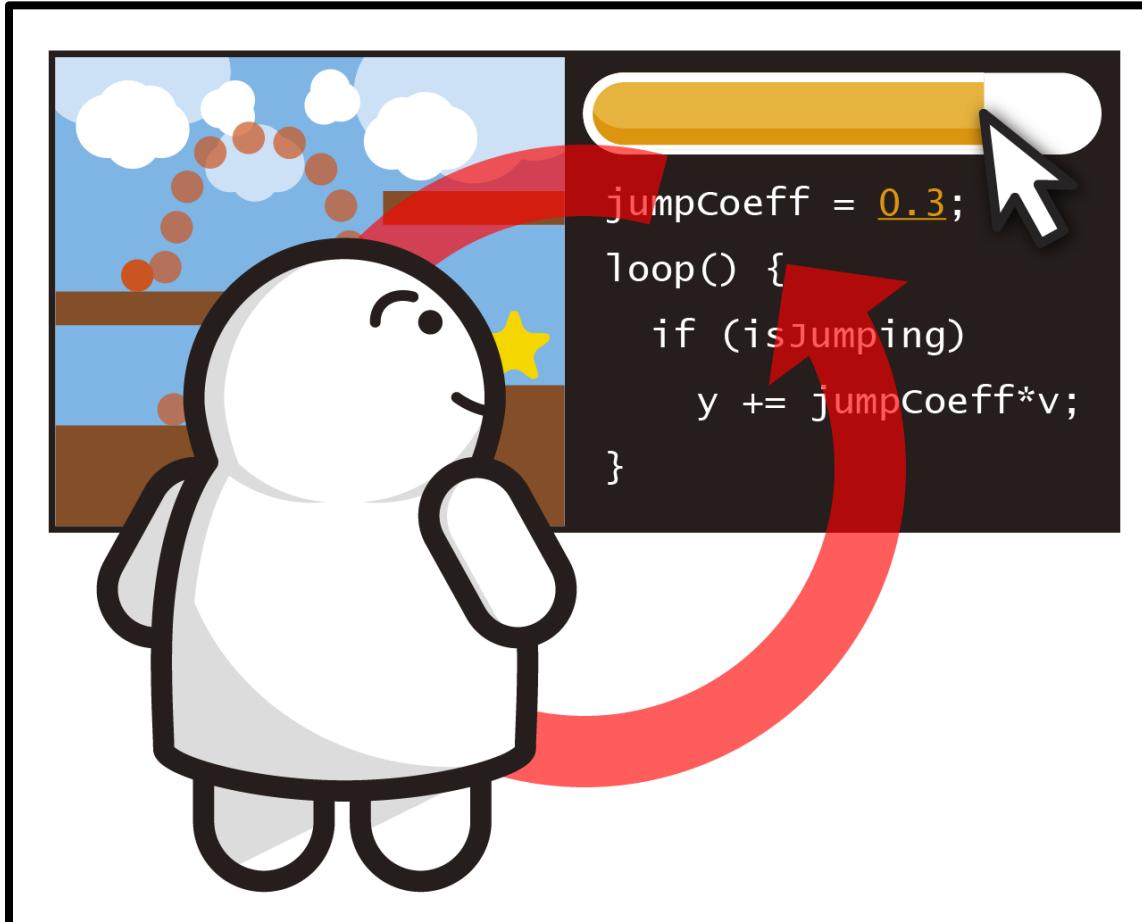
It is not about ...

- No λ or greek symbols in slides
- Not a consensus in the field (it's ongoing!)
- No peer review involved (my personal view)

Today, I'm going to talk about ...

- **What is Live Programming?**
 - UIs for Live Programming with end-users
 - UIs for Live Programming of this material world
 - UIs for Live Programming with time travel
 - Live Programming as User Interface research
-

What is Live Programming?



- Programming experience
- Continuous feedback
- Concrete information
- Early examples in VPL and OOP
- Attracting much attention these days

Not new but hot!



Text-based Programming

Dartmouth BASIC [1964]

```
(C) Copyright Microsoft 1983,1984,1985,1986,1987  
60300 Bytes free  
0k  
10 READ A1, A2, A3, A4  
15 LET D = A1 * A4 - A3 * A2  
20 IF D = 0 THEN 65  
30 READ B1, B2  
37 LET X1 = (B1*A4 - B2 * A2) / D  
42 LET X2 = (A1 * B2 - A3 * B1)/D  
55 PRINT X1, X2  
60 GO TO 30  
65 PRINT "NO UNIQUE SOLUTION"  
70 DATA 1, 2, 4  
80 DATA 2, -7, 5  
85 DATA 1, 3, 4, -7  
90 END  
  
RUN  
4 -5.5  
.6666667 .1666667  
-3.666667 3.833333  
Out of DATA in 30  
0k  
100% 2 RUN+ 3 LOADW 4 STATUS 5 CONIN 6,"LP111 7 TURBO+ 8 TURBO+ 9 KEY 0 SCREEN
```

- Text-based editor
- Text-based debugger
- Text-based ...

Visual Studio Code [as of today]

```
apps - textalive - Visual Studio Code  
ファイル(F) 編集(E) 選択(Q) 表示(V) 移動(G) デバッグ(D) タスク(T) ヘルプ(H)  
エクスプローラー TS app.ts x  
開いているエディター  
TS app.ts  
TEXTALIVE  
 Scripts  
 stores  
 test  
 views  
 .gitignore  
 .htpasswd  
 JS app.js  
 TS app.ts  
 JS config.js  
 TS config.ts  
 Gruntfile.js  
 package-lock.json  
 package.json  
 README.md  
 TS refs.dts  
 textalive.njsproj  
 DOCKER  
 GITLENS  
問題 出力 デバッグ コンソール ターミナル 1: powershell  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
PS C:\Users\arc\Documents\Source\textalive\nodejs\textalive>  
Jun Kato, 2 months ago 行 16, 列 46 スペース 2 UTF-8 with BOM CRLF TypeScript 2.5.3 ▲ TSLint
```



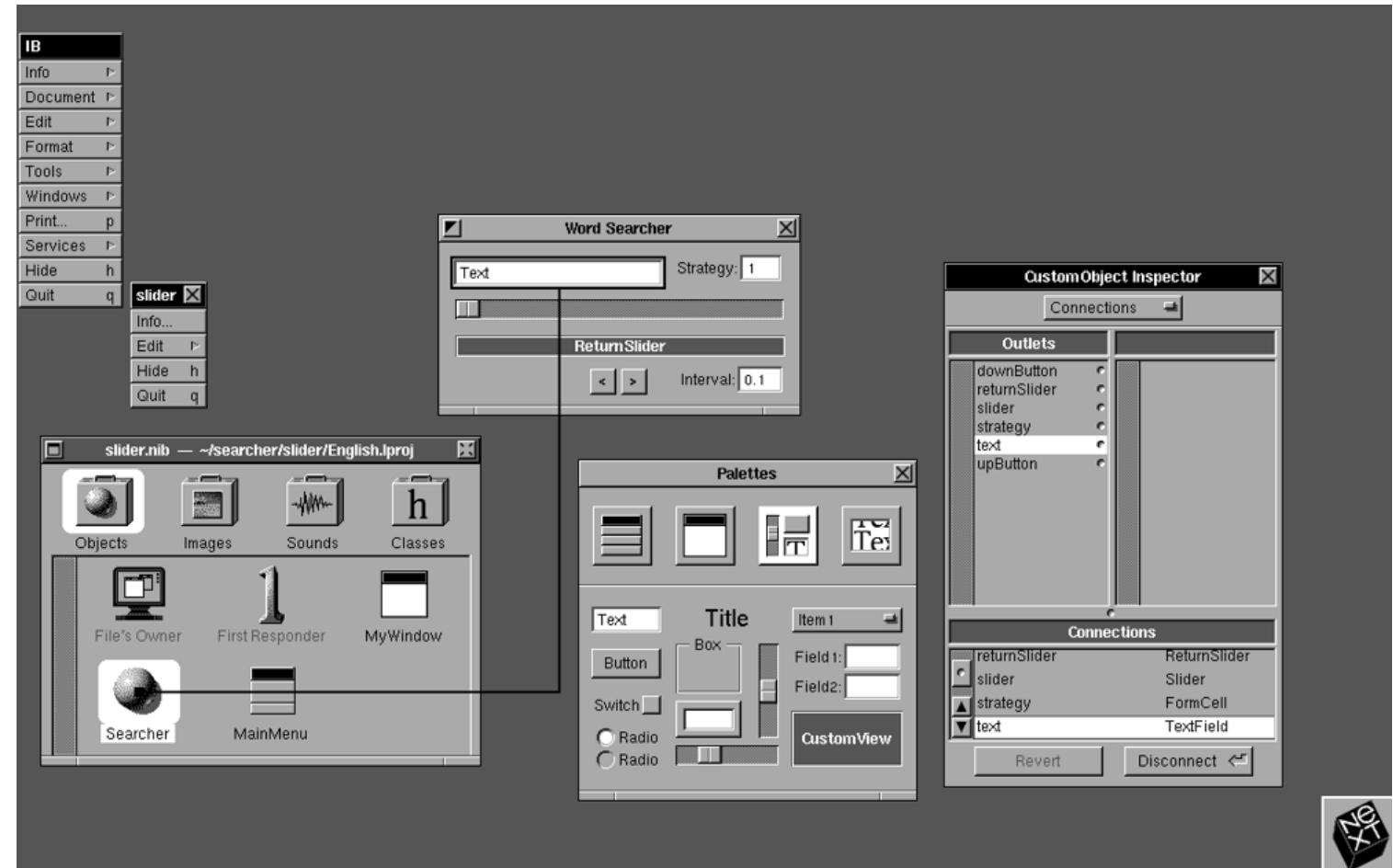
IDEs haven't changed much

With some exceptions ...

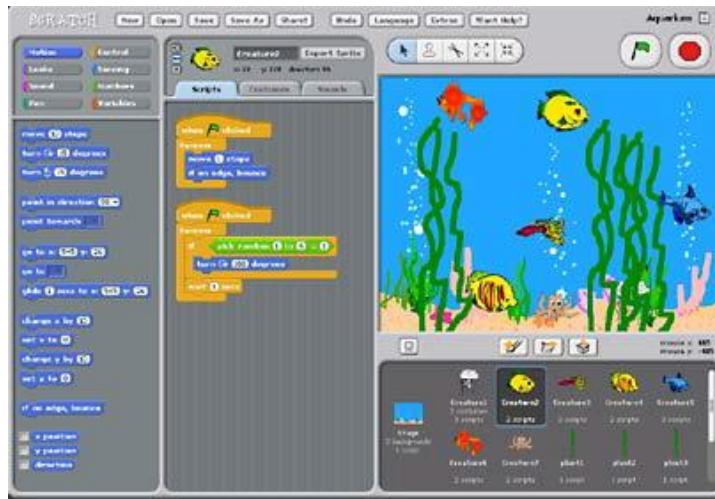
[since 1986]

Interface Builder

- A tool for NeXT UI development
- Later integrated into Xcode
- Many IDEs have similar built-in tools

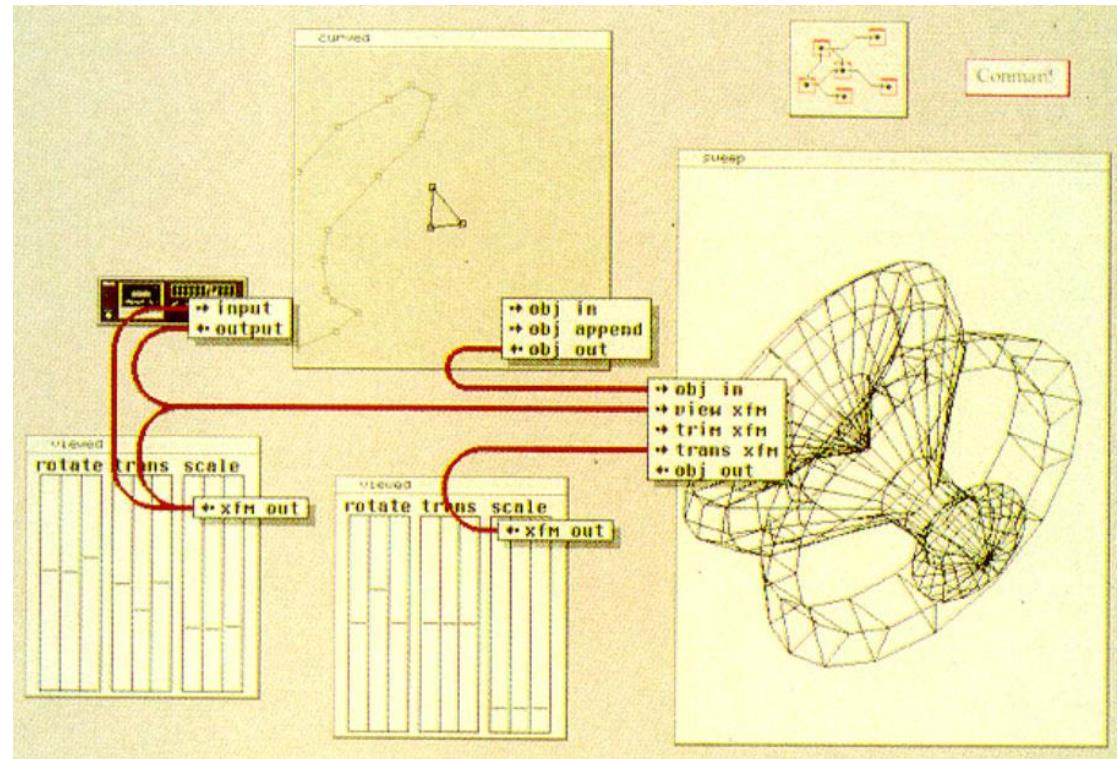


Visual Programming



Scratch
MIT

ConMan
Haberli
[SIGGRAPH 1988]

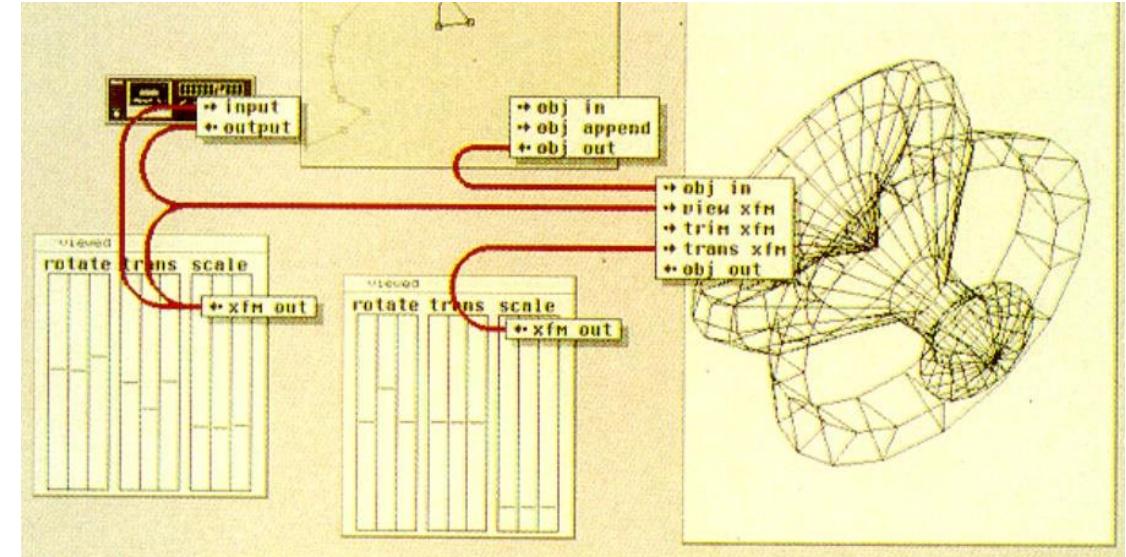


- Mostly dealing with symbolic representations of programs
- Often considered as tools for novices and good for education
- Dataflow languages: early examples of live programming

Character-based UIs **or** Graphical UIs?

```
(C) Copyright Microsoft 1983,1984,1985,1986,1987  
60300 Bytes free  
0k  
10 READ A1, A2, A3, A4  
15 LET D = A1 * A4 - A3 * A2  
20 IF D = 0 THEN 65  
30 READ B1, B2  
37 LET X1 = (B1*A4 - B2 * A2) / D  
42 LET X2 = (A1 * B2 - A3 * B1)/D  
55 PRINT X1, X2  
60 GO TO 30  
65 PRINT "NO UNIQUE SOLUTION"  
70 DATA 1, 2, 4  
80 DATA 2, -7, 5  
85 DATA 1, 3, 4, -7  
90 END  
  
RUN  
4 -5.5  
.6666667 .1666667  
-3.666667 3.833333  
Out of DATA in 30  
0k  
1LIST 2RUN+ 3LOAD+ 4SAVE+ 5CONT+ 6,"LPT1 7IRON+ 8TOFFF+
```

|
or
|

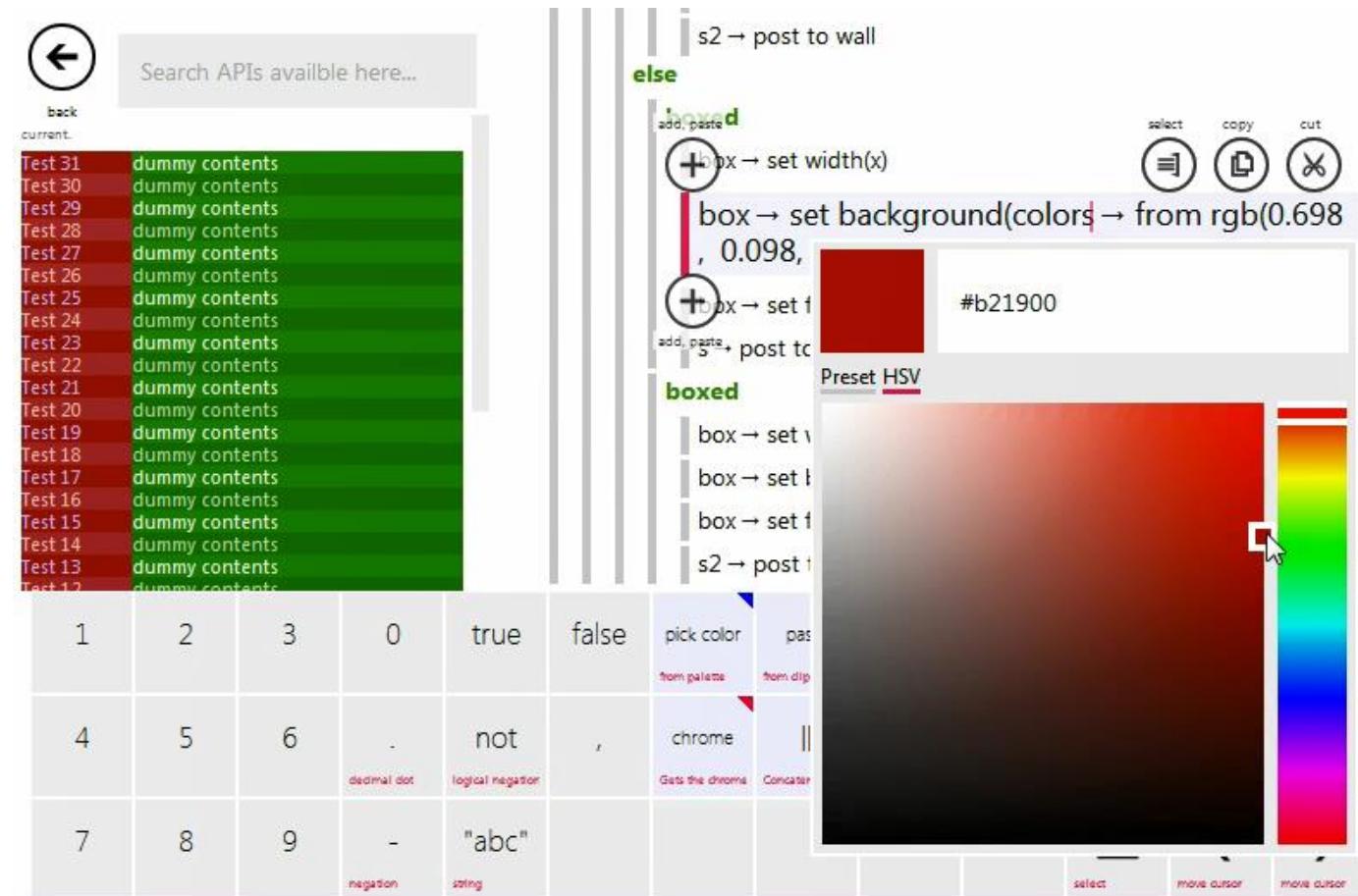


Modern



Live Programming as a hybrid approach

User Interfaces with Text **and** Graphics



TouchDevelop

[PLDI 2014]

<https://touchdevelop.com>

User Interfaces with Text **and** Graphics

f3.js

IoTコンテンツの編集 基本情報 ソースコード 組み立て

```
21 var f3js = require('f3js')
22 , x = 10
23 , y = 10
24 , useCountdown = true /* Use the countdown feature. */
25 , width = useCountdown ? 130 : 60
26 , height = 105
27 , thickness = 36 /* Thickness [10, 100] */
28 , dw = 5 /* Joint width [0, 10] */
29 , dh = 2 /* Joint height (panel thickness) [0, 10] */;
30
31 // put base board
32 var rect = f3js.drawJointRectangle(x, y, width, height, dw
33 var planes = rect.extrude(thickness);
34 planes[4].x = width; planes[4].y = 0; f3js.add(planes[4]);
35
36 // put sensors and actuators
37 var leftMargin = 28 // Left margin [0, 100]
38 , topMargin = 30; // Top margin [0, 100]
39 f3js.add(camera, { x: x + leftMargin + 0, y: y + topMargin
40 f3js.add(button, { x: x + leftMargin + 0, y: y + 75 });
41
42 var circle;
43 if (useCountdown) {
44     circle = new gcl.GroveCircularLED(5, 4);
45     f3js.add(circle, { x: x + width - 30, y: y + topMargin
46     f3js.drawJointCircle(x + width - 30, y + topMargin, 10, 5, 2);
```

Camera(0)

GroveCircularLED(5,4)

GroveButton(3)

... . . .

105

カスタマイズ

Use the countdown feature.

Thickness (36)

Joint width (5)

Joint height (panel thickness) (2)

Left margin (28)

f3.js

[DIS 2017]

<http://f3js.org>

Character-based UIs and Graphical UIs

- It's like text **and** figures in research papers
- **Text** is good at abstraction
- **Graphics** are good at presenting concrete information

Integrated Graphical Representations [2014, dissertation] [2016]



They complement each other

Picode: Inline Photos Representing Posture Data in Source Code

Jun Kato

The University of Tokyo, Tokyo, Japan – {jun.kato | d.sakamoto | takeo}@acm.org

Daisuke Sakamoto

Takeo Igarashi

ABSTRACT

Current programming environments use textual or symbolic representations. While these representations are appropriate for describing logical processes, they are not appropriate for representing raw values such as human and robot posture data, which are necessary for handling gesture input and controlling robots. To address this issue, we propose *Picode*, a text-based development environment augmented with inline visual representations: photos of human and robots. With *Picode*, the user first takes a photo to bind it to posture data. She then drag-and-drops the photo into the code editor, where it is displayed as an inline image. A preliminary user study revealed positive effects of taking photos on the programming experience.

Author Keywords

Development Environment; Inline Photo; Posture Data.

ACM Classification Keywords

H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces – GUI; D.2.6. Software Engineering: Programming Environments – Integrated environments.

INTRODUCTION

A programming language is an interface for the programmer to input procedures into a computer. As with other user interfaces, there have been many attempts to improve its usability. Such attempts include visual programming languages to visualize the control flow of the program, structured editors to prevent syntax errors, and enhancement to code completion that visualizes possible inputs [8]. However, programming languages usually consist of textual or symbolic representations. While these representations are appropriate for precisely describing logical processes, they are not appropriate for representing the posture of a human or a robot. In such a case, the programmer has to list raw numeric values or to maintain a reference to the datasets stored in a file or a database.

To address this issue, Ko and Myers presented a framework called “Barista” for implementing code editors which are capable of showing text and visual representations [5]. This framework enhances comments for an image processing

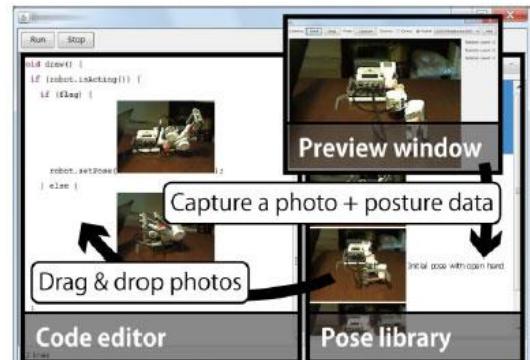


Figure 1. Overview of *Picode*

method by including an image that shows a concrete example of what the method does. Yeh et al. presented a development environment named “Sikuli,” with which the programmer can take a screenshot of a GUI element and paste the image into a text editor [12]. In Sikuli, the image serves as an argument of the API functions. Our goal was to apply a similar idea to facilitate the programming of applications that handle human and robot postures.

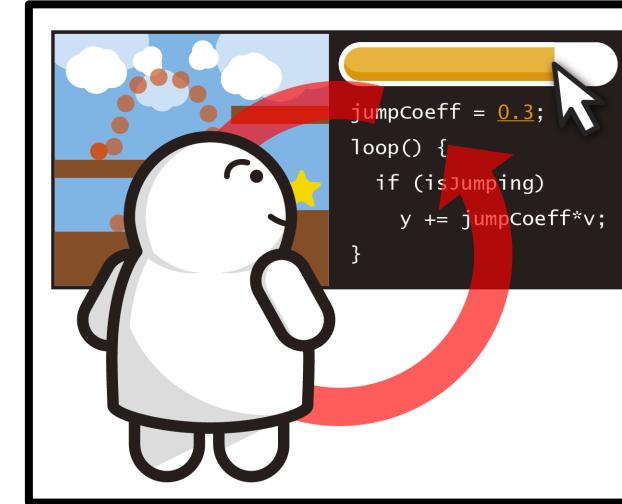
We propose a development environment named *Picode* that uses photographs of human and robots to represent their posture data in a text editor (Figure 1). It helps the development process of applications for handling gesture input and/or controlling robots. The programmer is first asked to take a photo of a human or a robot to bind it to the posture data. She then drag-and-drops the photo into the code editor, where it is shown as an inline image. Our environment provides a built-in API which methods take photos as arguments. It allows the user to easily understand when the photo was taken and what the code is meant to do.

RELATED WORK

After the Microsoft Kinect and its Software Development Kit (SDK) hit the market, many interactive applications have been developed that handle human posture. At the same time, some toolkits and libraries have been proposed that support the development of such applications. They can typically recognize preset poses and gestures. When the programmer wants to recognize her own poses and gestures, however, she has to record the examples outside the development environment. On the other hand, our development environment is designed to support the entire prototyping process of application development. It fully

In Live Programming systems, we ...

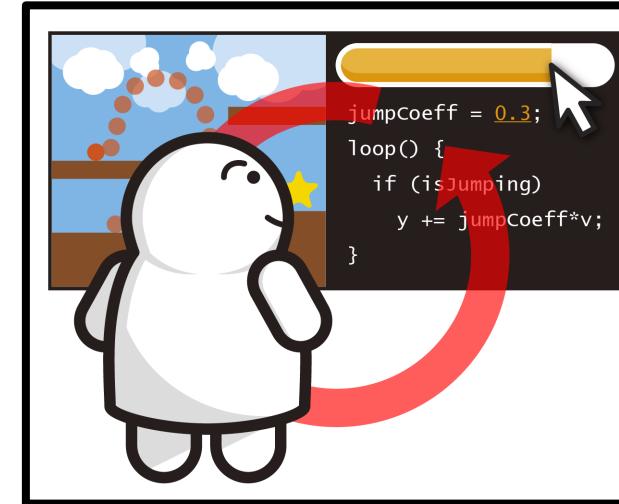
- first have **vague ideas**
- then explore the ideas with **concrete examples**
- gradually start turning the ideas into **programs**



Live Programming requires
decent UIs for exploring the problem space

UIs for Live Programming should ...

- avoid sudden changes in the program behavior
- keep the program and its output relevant
- allow continuously exploring the problem space



Appropriate user interface design differs
from application to application

When designing live programming systems ...

Don't be afraid to be domain-specific

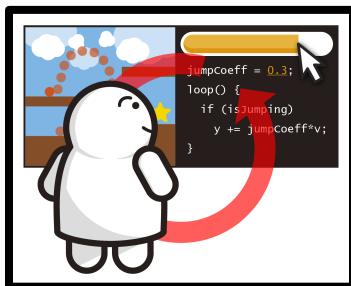
- Good UI is always specifically designed for the target domain
- It might be like replaying the history of end-user computing in the domain of programming
- We might need PX workbench (cf. language workbench)

Cf. Programming eXperience Toolkit (PXT)

<https://github.com/microsoft/pxt>

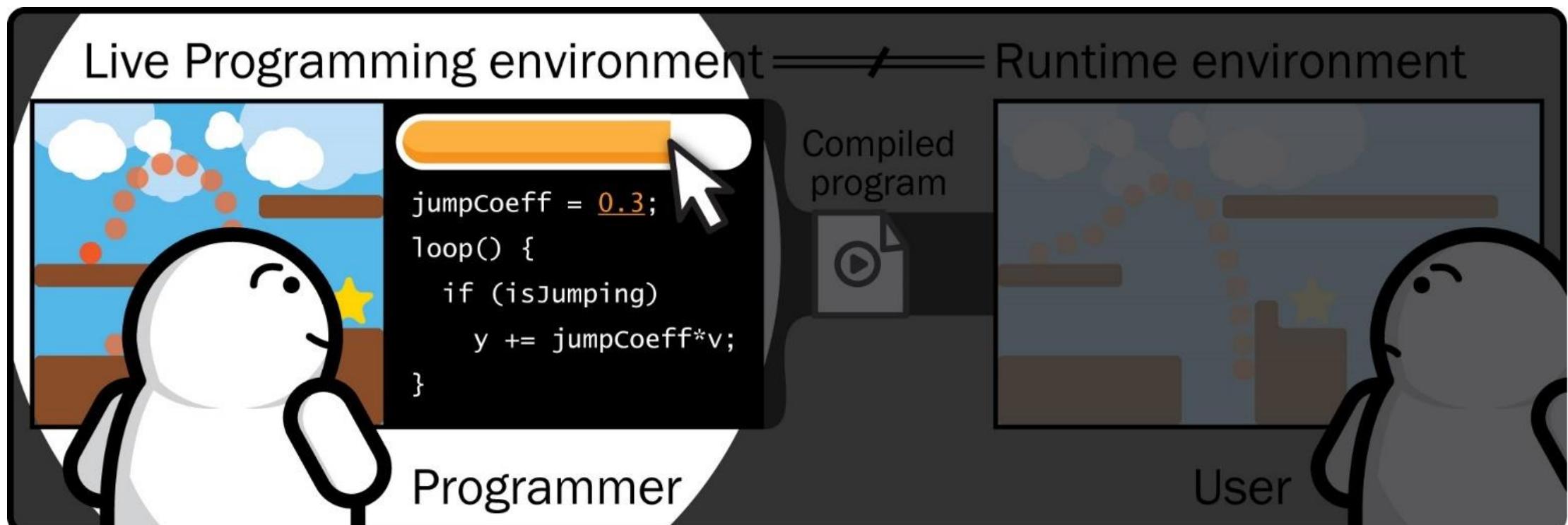
Today, I'm going to talk about ...

- What is Live Programming?
- **UIs for Live Programming with end-users**
- UIs for Live Programming of this material world
- UIs for Live Programming with time travel
- Live Programming as User Interface research



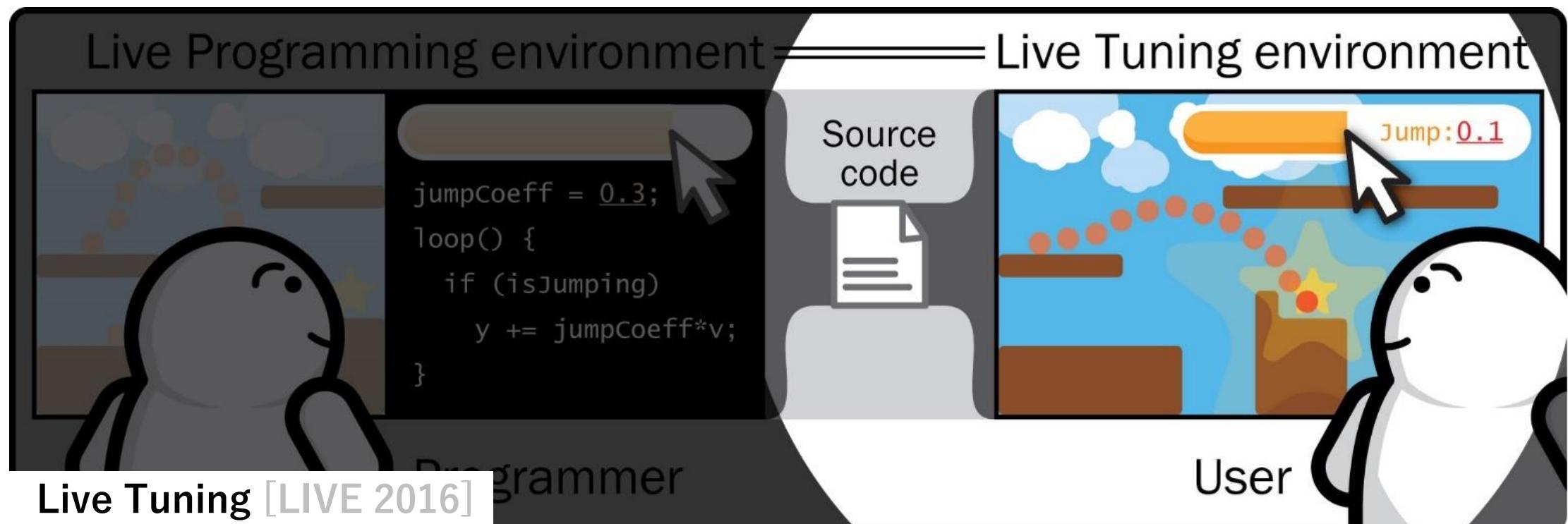
UIs for Live Programming

Good mixture of text-based and graphical user interfaces

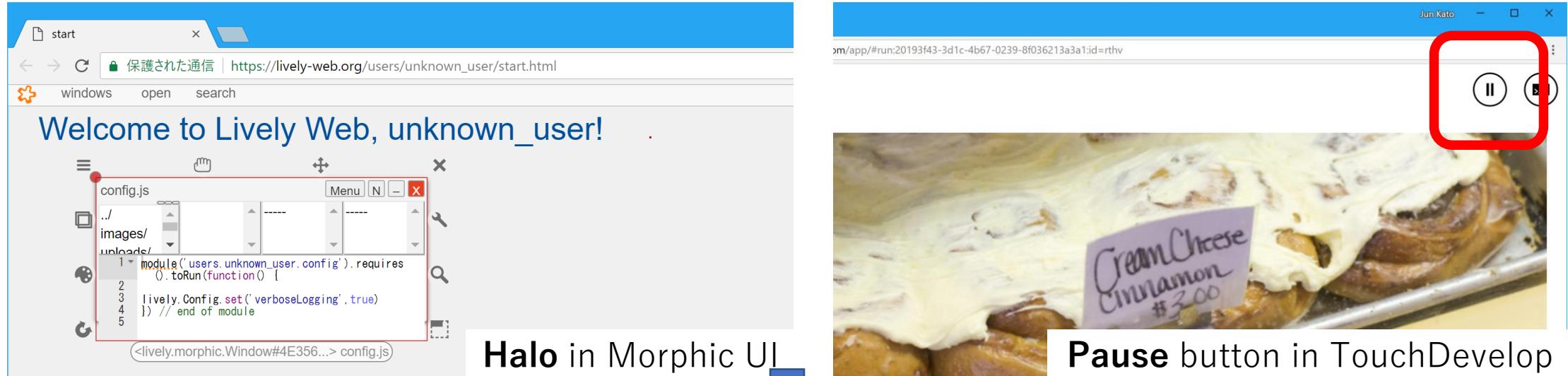


UIs for Live Programming

Why not expose GUI to users so that they can edit programs?



Mode Switch between “Use” and “Build”



What if we add another layer for **users**?

Promoting universal usability with multi-layer interface design
Ben Shneiderman [2002]



TextAlive

Songs

Videos

Templates

Edit

Login

Full screen

Help ▾

A large, semi-transparent white play button icon is centered on a yellow background. The background features a grid of smaller, semi-transparent yellow rectangles of varying sizes.

TextAlive
[CHI 2015] <http://textalive.jp>



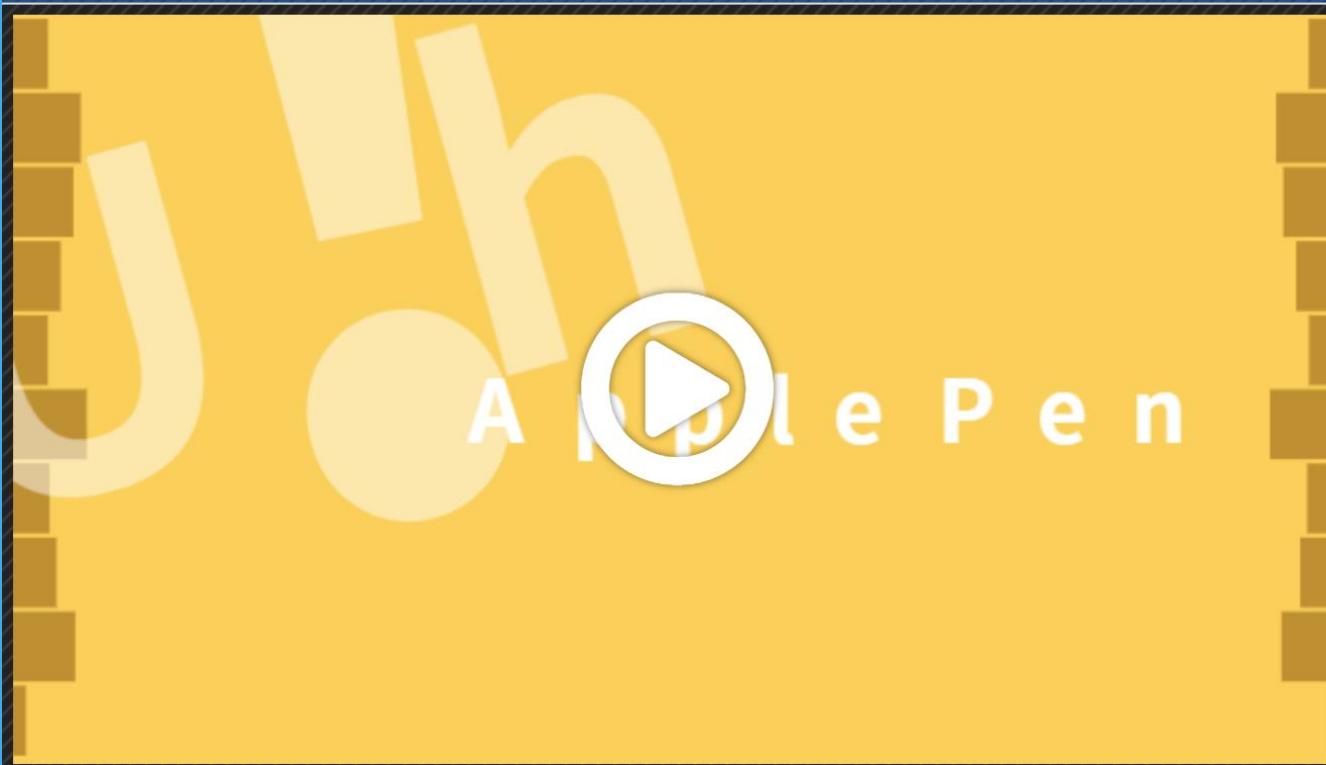
TextAlive



arc@dmz

Full screen

Help



Parameter tweaking (横書き配置)

文字色



単語padding



文字padding



Fadein Time

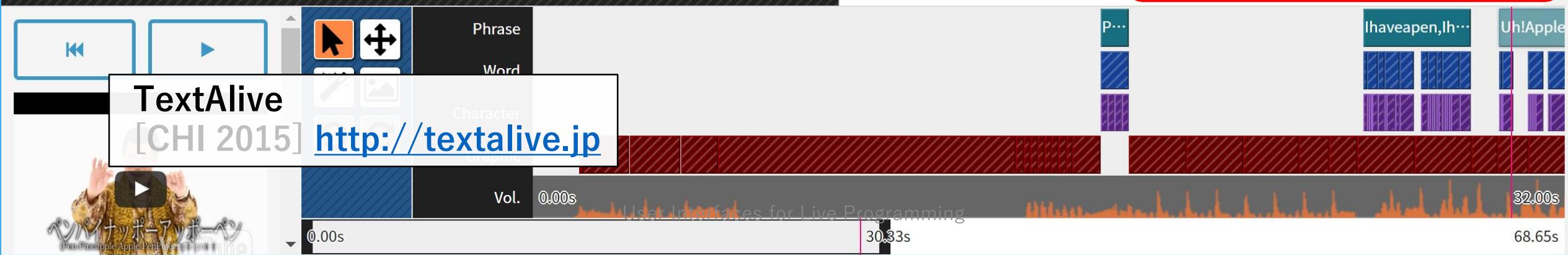


Fadeout Time



Propose

Programming





TextAlive



Save

Options

arc@dmz

Full screen

Help



```
38     var fadeout_progress = (p.endTime - now) / this.fadeoutTime;
39     p.rendering.alpha = fadeout_progress;
40   }
41
42   var hrange = p.advance
43     + (p.wordCount - 1) * this.wpadding
44     + (p.charCount - 1) * this.cpadding;
45   var wx = (width - hrange) / 2;
46   var cx = 0;
47   var w = p.firstWord;
48   for (var i = 0; i < p.wordCount; i++) {
49     w.rendering.tx.translate(wx, (height - w.height) / 2);
50     var c = w.firstChar;
51
52     cx = 0;
53     for (var n = 0; n < w.charCount; n++) {
54       c.color = this.textColor;
55       c.rendering.tx.translate(cx, 0);
56       cx += c.advance + this.cpadding;
57       c = c.next;
58     }
59     wx += cx + this.wpadding;
60     w = w.next;
61   };
62 }
63 }
```

The screenshot shows the TextAlive application interface. At the top, there are navigation buttons (back, forward, search), a save button, and an options menu. The main area features a large yellow background with the text "Uh! Apple Pen". Below this is a control bar with buttons for playback, zoom, and selection, and dropdown menus for "Phrase" and "Word". A preview window displays a video frame with the text "TextAlive [CHI 2015] http://textalive.jp". The bottom section includes a volume slider, a waveform audio player, and a timeline with markers at 0.00s, 30.33s, 32.00s, and 68.65s. On the right, there are two small video frames showing different text segments: "I have a pen, I have an apple pen" and "Uh! Apple".

Co-hosting UIs for programmers and users

Simple spectral analysis

An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1$$

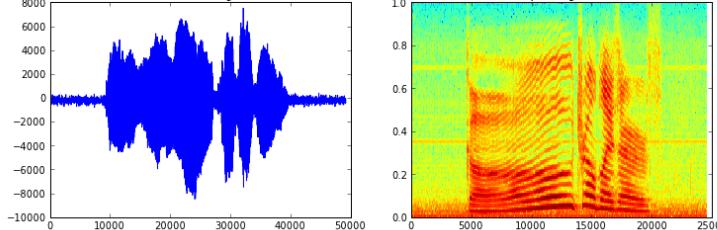
using windowing, to reveal the frequency content of a sound signal.

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin specgram routine:

```
In [2]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



The left plot, titled "Raw audio signal", shows a blue line graph of a sound wave over time, ranging from 0 to 50,000 on the x-axis and -10,000 to 8,000 on the y-axis. The right plot, titled "Spectrogram", is a heatmap showing frequency components over time, with the x-axis from 0 to 25,000 and the y-axis from 0.0 to 1.0.

Literate Programming
in Jupyter (ipython) Notebook

BallShooter | Picode

Sketch File View

BallShooter

```
if (pose.eq( )) {
    showText("Got the command!");
    nxt.setPose( );
}
```

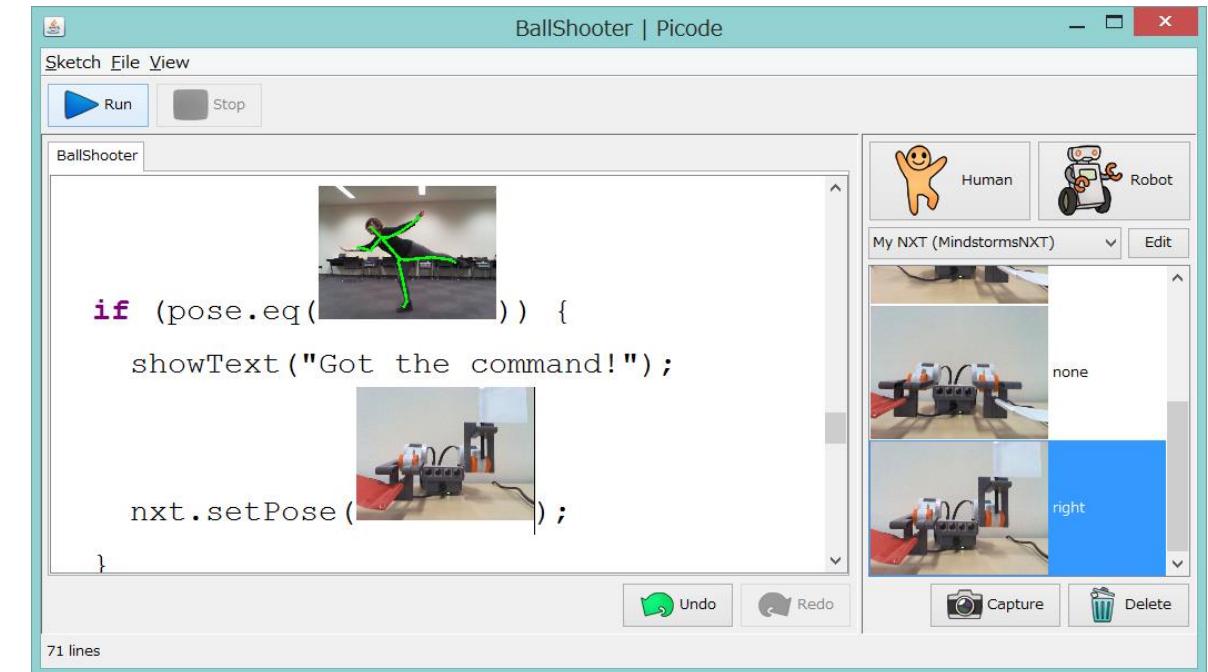
Human Robot

My NXT (MindstormsNXT) Edit

none

right

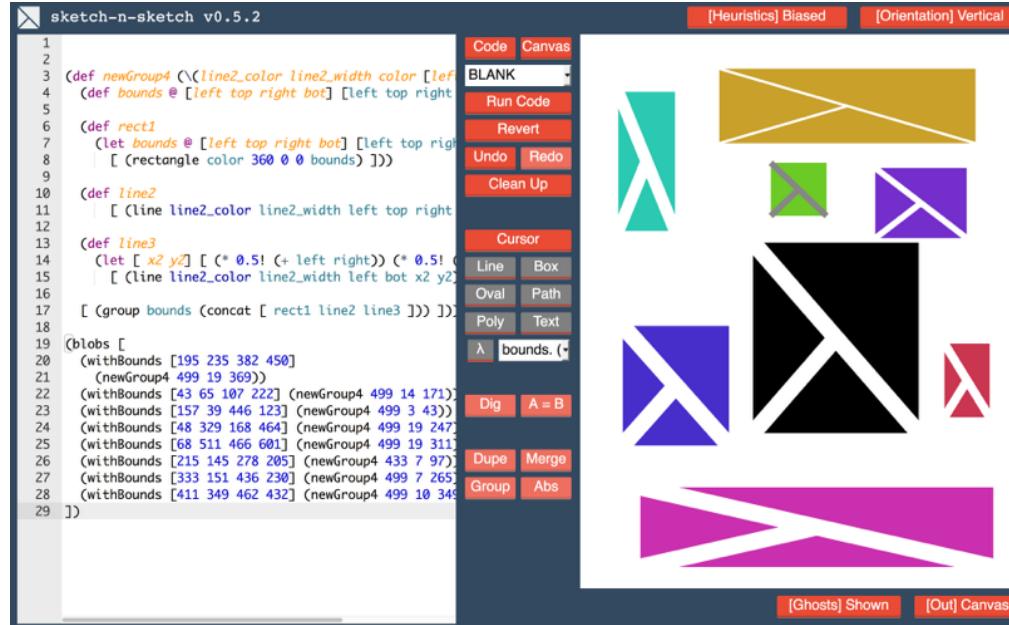
Capture Delete



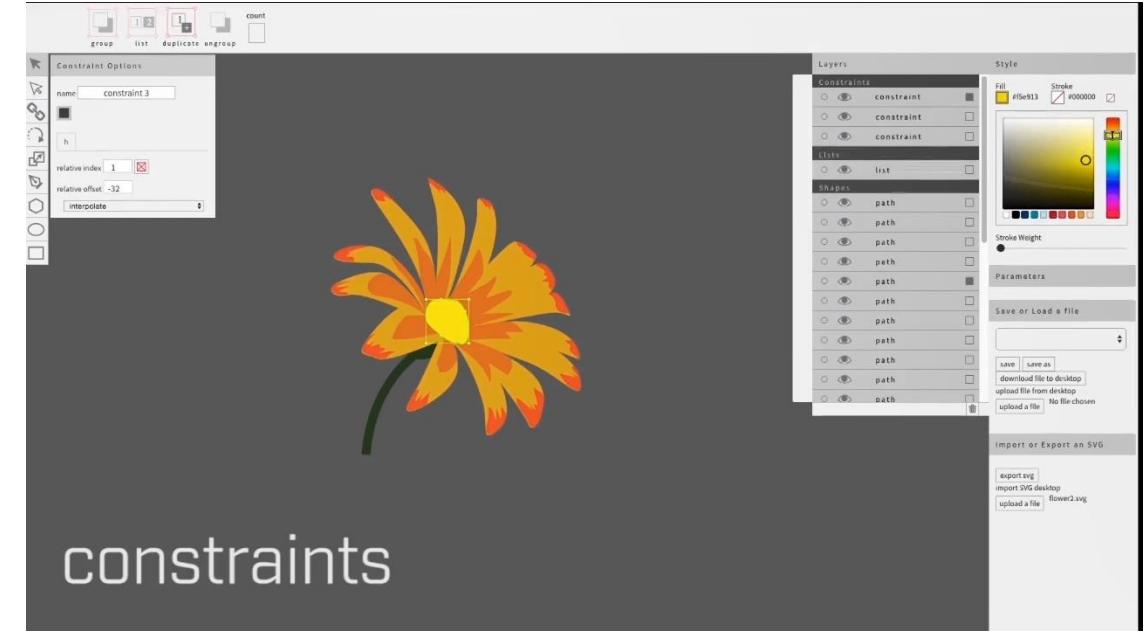
The screenshot shows the Picode IDE interface. The main area displays a sketch titled "BallShooter". It contains a conditional block: "if (pose.eq()) { showText("Got the command!"); nxt.setPose(); }". To the right of the code are two video feeds: one showing a person in a dynamic pose and another showing a robotic arm on a table. On the far right, there are icons for "Human" and "Robot", and sections for "My NXT (MindstormsNXT)" and "Edit". At the bottom, there are "Undo" and "Redo" buttons, along with "Capture" and "Delete" buttons.

Inline Photos
in Picode [CHI 2013]

Merging UIs for programmers and users (direct manipulation on program output)



Sketch-n-Sketch, Hempel et al.
[UIST 2016 etc.]

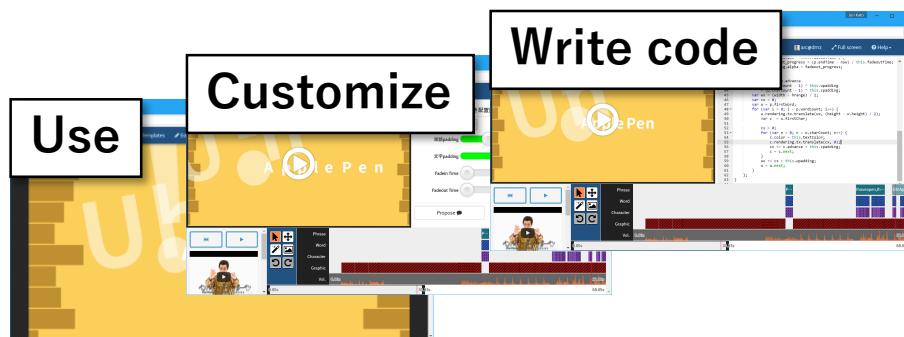


Para, Jacobs et al.
[CHI 2017]

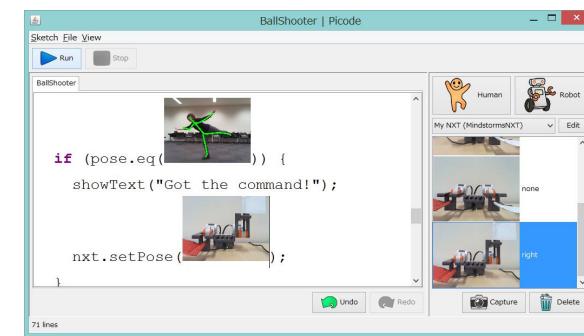
When designing live programming systems ...

How about making the ladder of expertise?

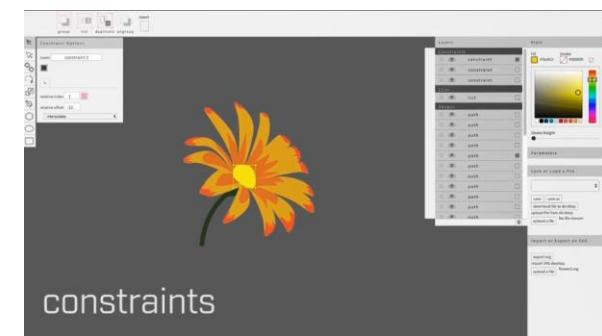
- From live programming for programmers
- To live programming for the community of people



Multi-layer



Co-host

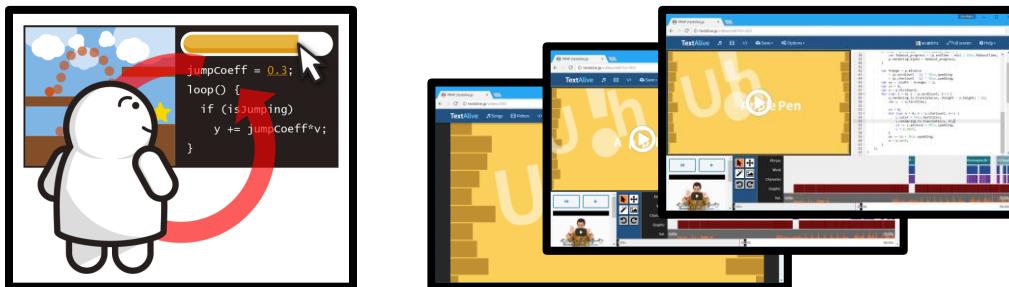


constraints

Merge

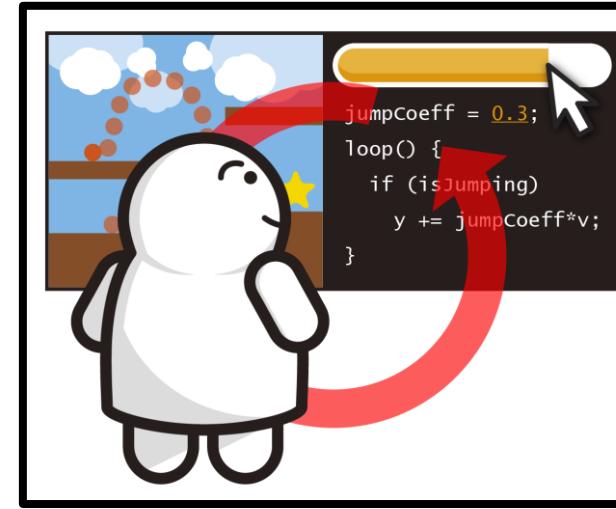
Today, I'm going to talk about ...

- What is Live Programming?
- UIs for Live Programming with end-users
- **UIs for Live Programming of this material world**
- UIs for Live Programming with time travel
- Live Programming as User Interface research



What is “live” and what not?

- System response time:
 - Computation
 - Network
 - Touch display response
- Reflex time:
 - Visual 0.25s
 - Audio 0.17s
 - Touch 0.15s
- and more ...
 - 3D printers and laser cutters
 - Shape changing devices and robots
 - Taste/smell interfaces



How Live are Live Programming Systems?

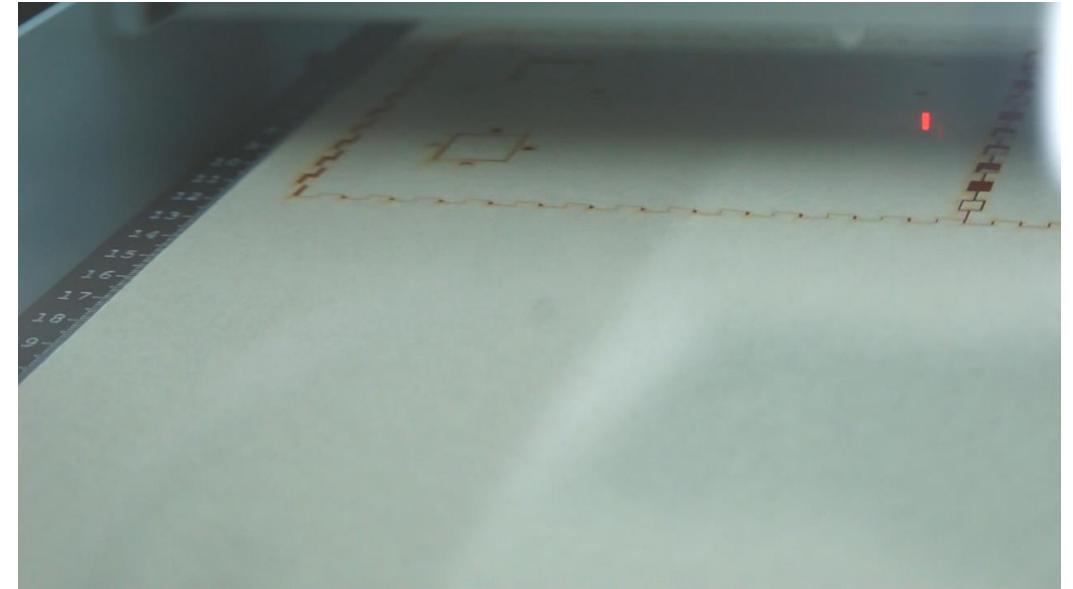
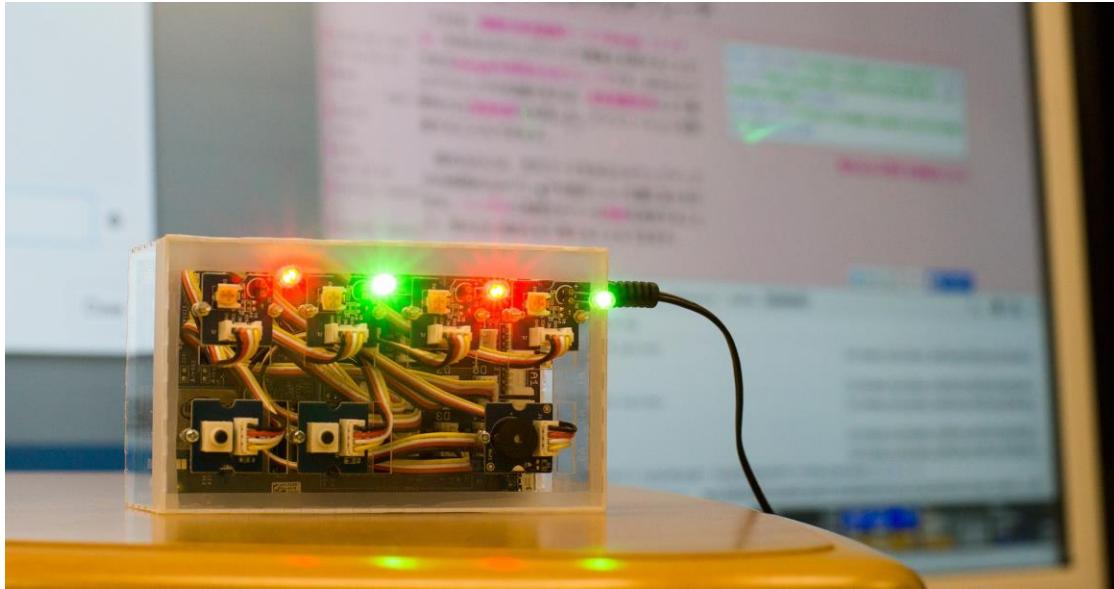
Rein et al. [PX 2016]

Various kinds of “latencies”



Printing physical computing devices

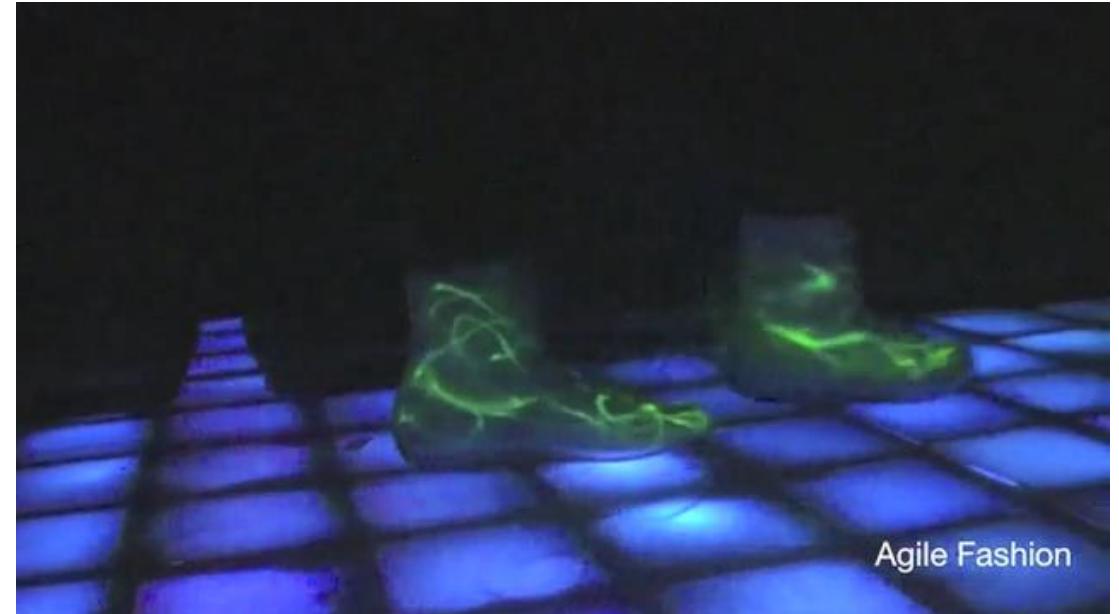
f3.js [DIS 2017]



Slow “framrates” prevent live feedback

Slow display

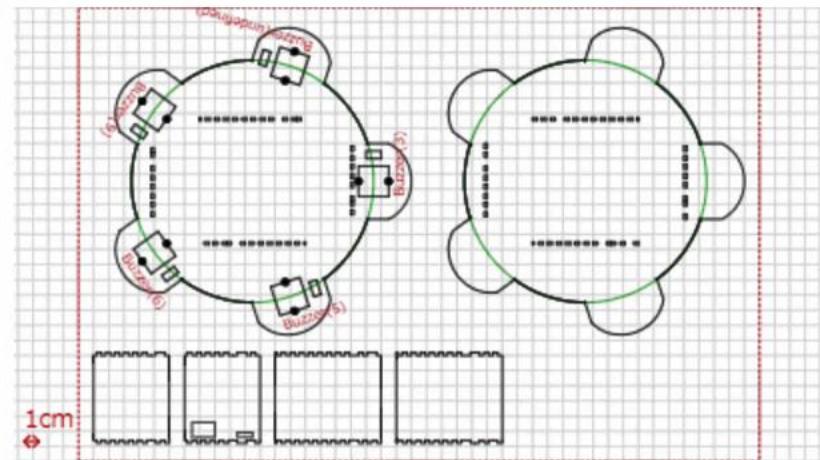
Daniel Saakes et al. [SIGGRAPH Etech 2010]



Slow “framерates” can be useful, too

</> Source Code

Layout



Selected

Hovered

ページ: 1

Customization

Number of buzzers (5)



Layout view options

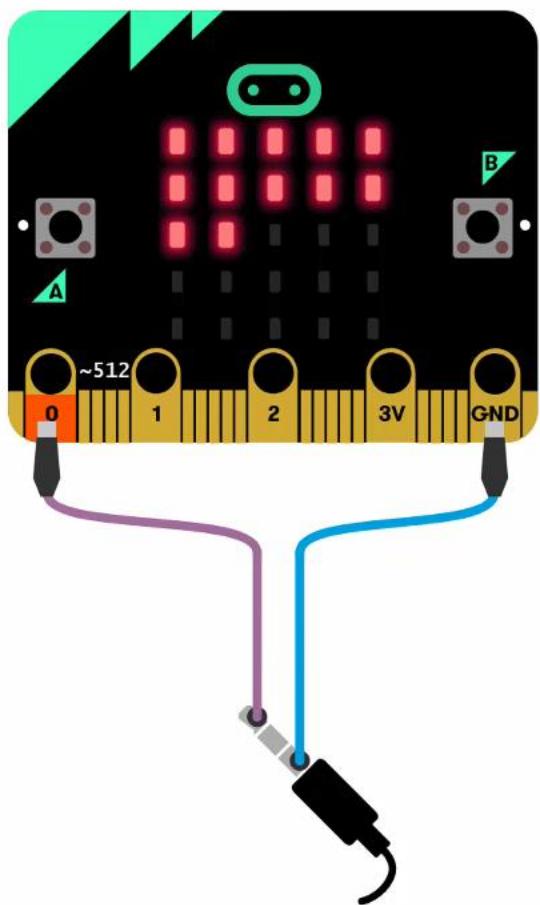
Provide the source code of a microcontroller or tiny computer in JavaScript. Node.js-based computers are supported.
Require f3js package and use its [API](#) to design the device enclosure.

```

1 var WebSocket = require('ws');
2 var serverAddr = 'ws://192.168.10.100:8080/entry';
3 var ws = new WebSocket(serverAddr);
4 var upmBuzzer = require('jsupm_buzzer');
5
6 var numBuzzers = 5; // Number of buzzers [1, 5]
7 var inputs = [3, 5, 6, 9];
8 var buzzers = [];
9 var volume = 0.1;
10 var r = 80;
11
12 var f3js = require('f3js');
13 var c = f3js.createContainer();
14 c.x = 50;
15 c.y = 50;
16
17 var c1 = 43
18   , c2 = 36
19   , x = 70
20   , y = 140
21   , width = 70;
22 var a = c.createPath();
23
24 var ps = a.extrude(60);
25 ps[0].y = 50;
26 ps[0].x = 270;
27 f3js.add(ps[0]);
28
29 var d1 = 115
30   , d2 = 115
31   , d3 = 115
32   , d4 = 115
33   , cc = f3js.createContainer();
34   , nn = cc.createPath()

```

f3.js: IoT apps with enclosures from single code base
 [DIS 2017] <http://f3js.org>



Search...

Basic

Input

Music

Led

Radio

Loops

Logic

Variables

Math

Advanced

forever

ring tone (Hz) Middle C

on start

while true

do for index-y from 0 to 4

do for index from 0 to 4

do toggle x index y index-y

pause (ms) 100

Getting Started

MakeCode for BBC micro:bit, Microsoft Research [2017]<http://makecode.microbit.org>

Download

blinking-leds User Interfaces for Live Programming





Sketch



Picode: inline photos representing posture data in source code [CHI 2013]

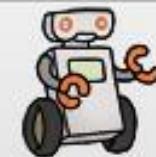
BallShooter



```
if (pose.eq( )) ) {  
    showText("Got the command!");  
    nxt.setPose( );  
}
```



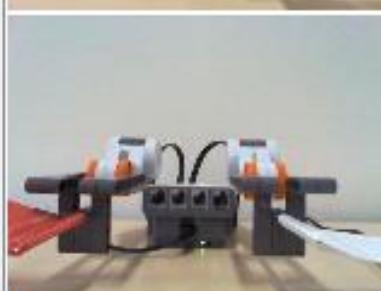
Human



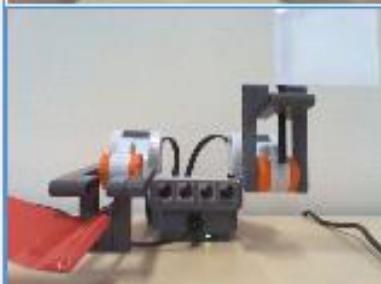
Robot

My NXT (MindstormsNXT)

Edit



none



right



Undo



Redo



Capture



Delete

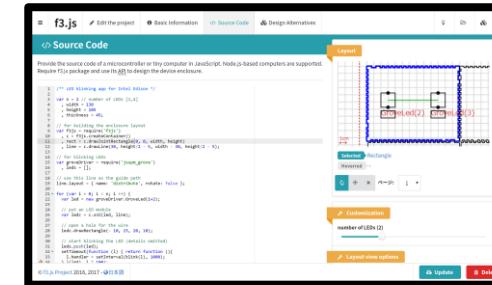
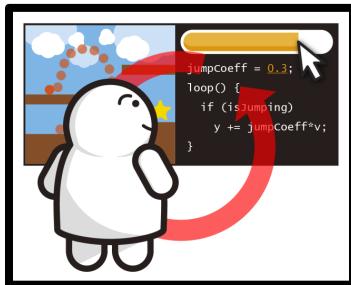
When designing live programming systems ...

Deceiving users' perception is a good thing

- As long as **the lie is reasonable**
- The actual "**framerate**" can be very slow
- Emulating, or sometimes even pretending, is needed to provide the **continuous feedback**
- Make **full use of five senses** in programming environments

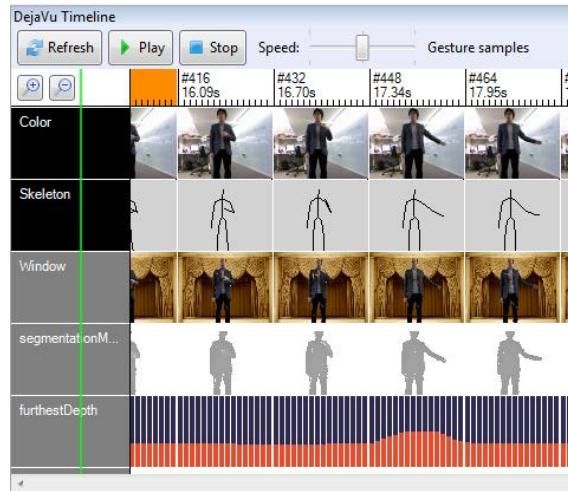
Today, I'm going to talk about ...

- What is Live Programming?
- UIs for Live Programming with end-users
- UIs for Live Programming of this material world
- **UIs for Live Programming with time travel**
- Live Programming as User Interface research



“Live” is not always about “now”

- **UIs for exploring and modifying the past**
- UIs for predicting and choosing the future
- Absolute vs semantic timeline



Window1.xaml.cs

KinectDress.Window1

OnKinectFramesReady(object sender, KinectFrameReadyEventArgs e)

```
86  
87  
88     float sumDistance = 0;  
89     foreach (Joint joint in skeletonData.Joints) {  
90         sumDistance += joint.Position.Z;  
91     }  
92     float userDistance = sumDistance / 20;  
93  
94     //If the user is close enough to the camera, show the virtual window  
95     bool userIsNear = userDistance < 2.5;  
96     ShowStage(userIsNear);  
97  
98  
99  
100  
101  
102
```

DejaVu Canvas

Color Depth Skeleton Window

Color input

Window output

userDistance

Properties Classes DejaVu Canvas



Window1.xaml.cs

KinectDress.Window1

OnKinectFramesReady(object sender, KinectFrameReadyEventArgs e)

```
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
  
    float sumDistance = 0;  
    foreach (Joint joint in skeletonData.Joints) {  
        sumDistance += joint.Position.Z;  
    }  
    float userDistance = sumDistance / 20;  
  
    //If the user is close enough to the camera, show the vir  
    bool userIsNear = userDistance < 3;  
    ShowStage(userIsNear);
```

DejaVu Canvas

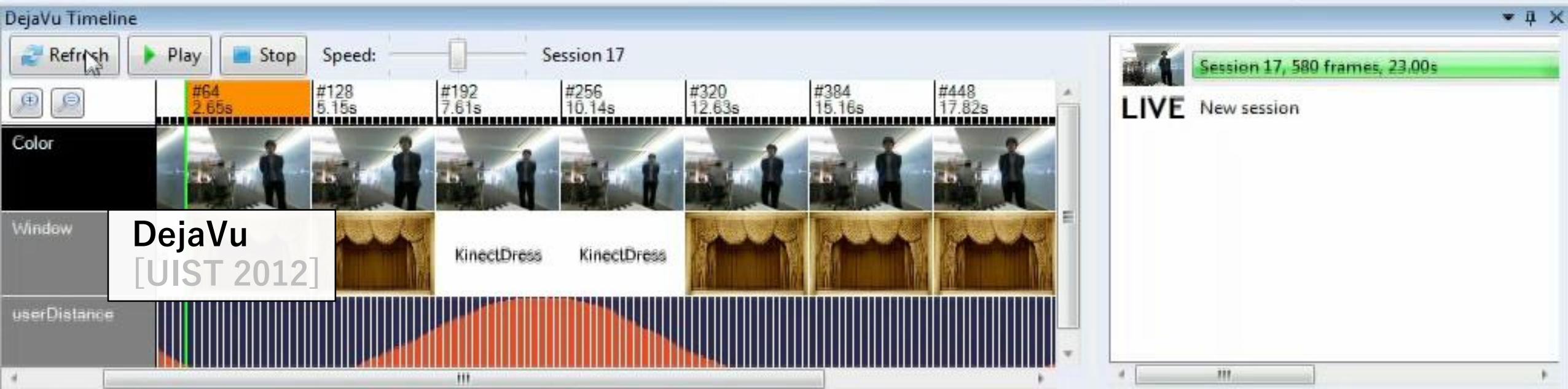
Celer Depth Skeleton Window

Celer input

Window output

userDistance

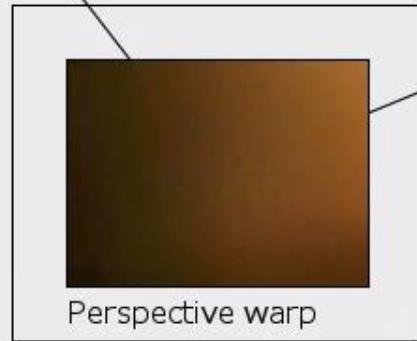
Properties Classes DejaVu Canvas



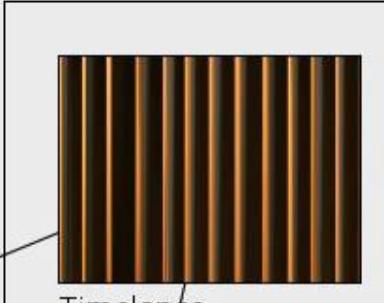


Canvas

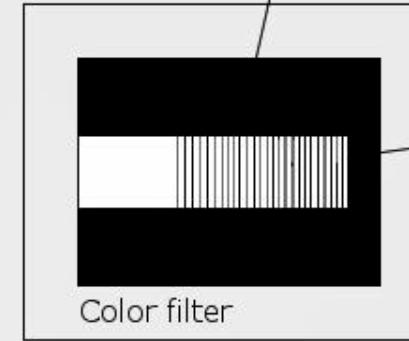
Save Load



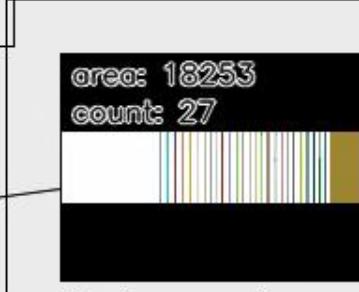
Perspective warp



Timelapse



Color filter



Contour counter

VisionSketch
[GI 2012]

Stop

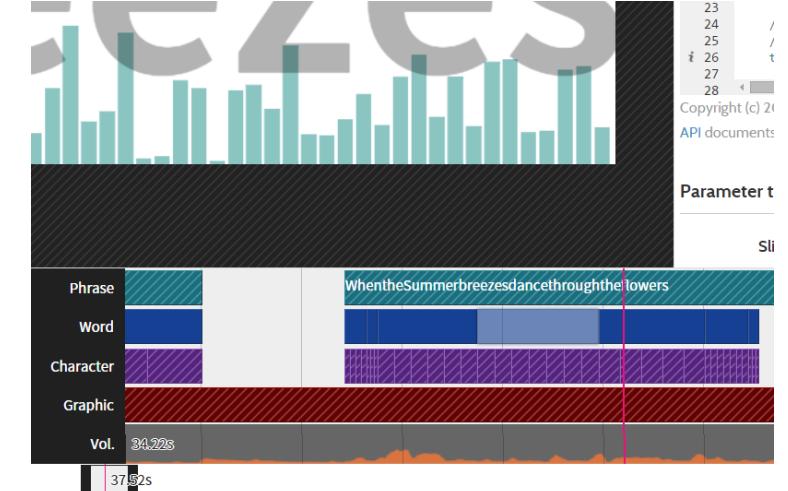
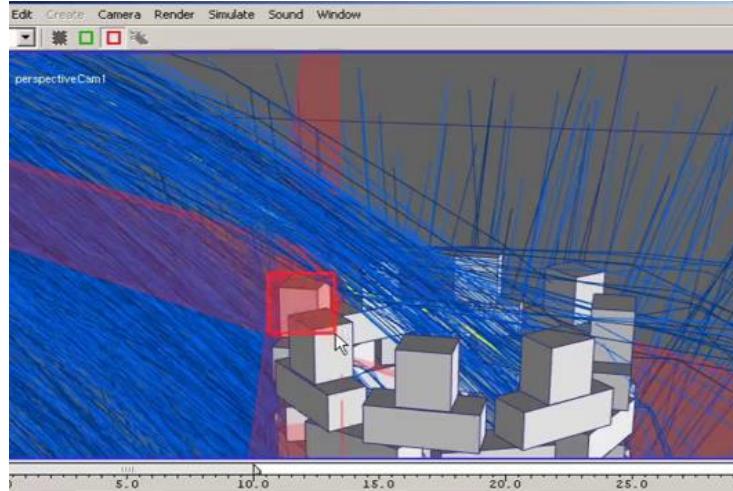
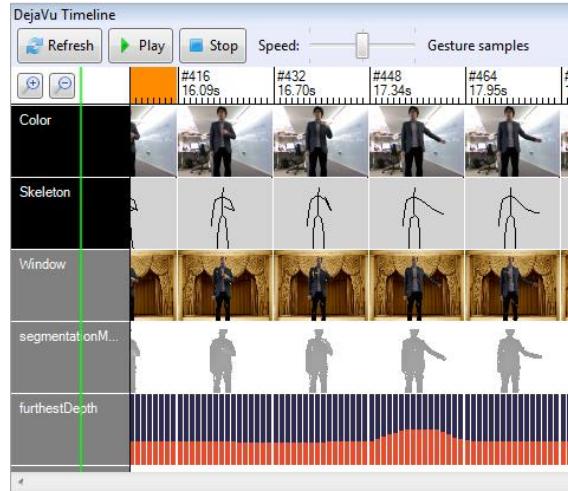
<<

User Interfaces for Live Programming

>>

“Live” is not always about “now”

- UIs for exploring and modifying the past
- **UIs for predicting and choosing the future**
- Absolute vs semantic timeline



Light Table – a new IDE concept, Chris Granger [2012]

<http://www.chris-granger.com/2012/04/12/light-table-a-new-ide-concept/>

```
defn jump (me)
  (let (speed 10)
    (if (and (not (jumping? me))
             (= (y me) 450))
        (set! (vx me) 0)
        (set! (vx me) (+ speed)))
    (set! (y me) (- (y me) speed))
    (set! (jumping? me) true))

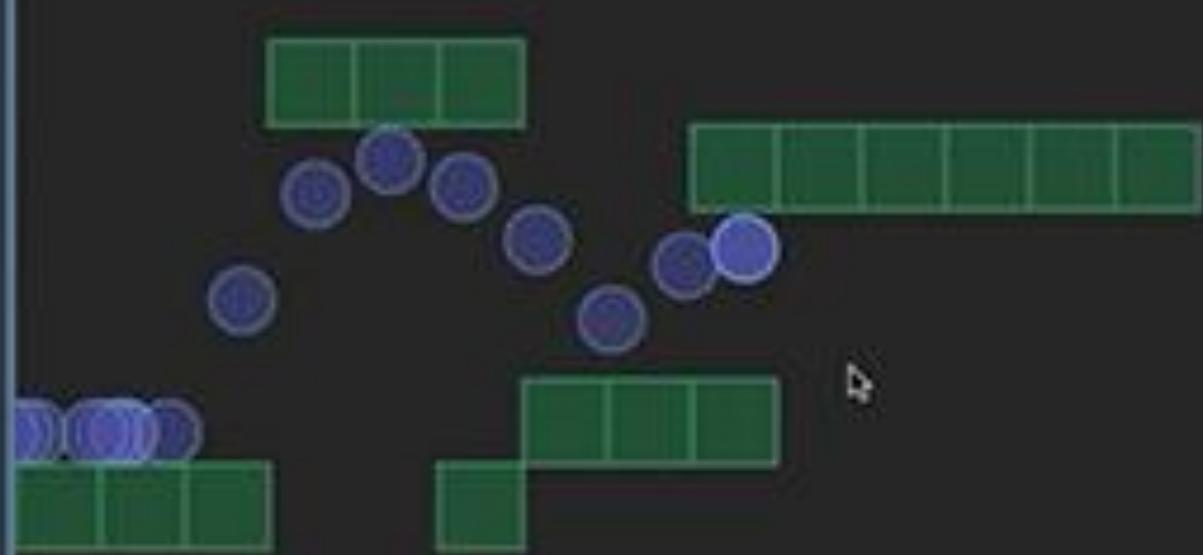
defn move (me)
  (let (speed 7)
    (if (and (not (moving? me))
             (= (y me) 450))
        (set! (vx me) 0)
        (set! (vx me) (+ speed)))
    (if (input? left) (- speed)
        (if (input? right) speed
            (else 0)))
    (set! (vx me) (+ vx (vx me)))
    (set! (y me) (- (y me) speed))
    (if (input? up)
        (set! (y me) (- (y me) speed))
        (set! (jumping? me) false)))

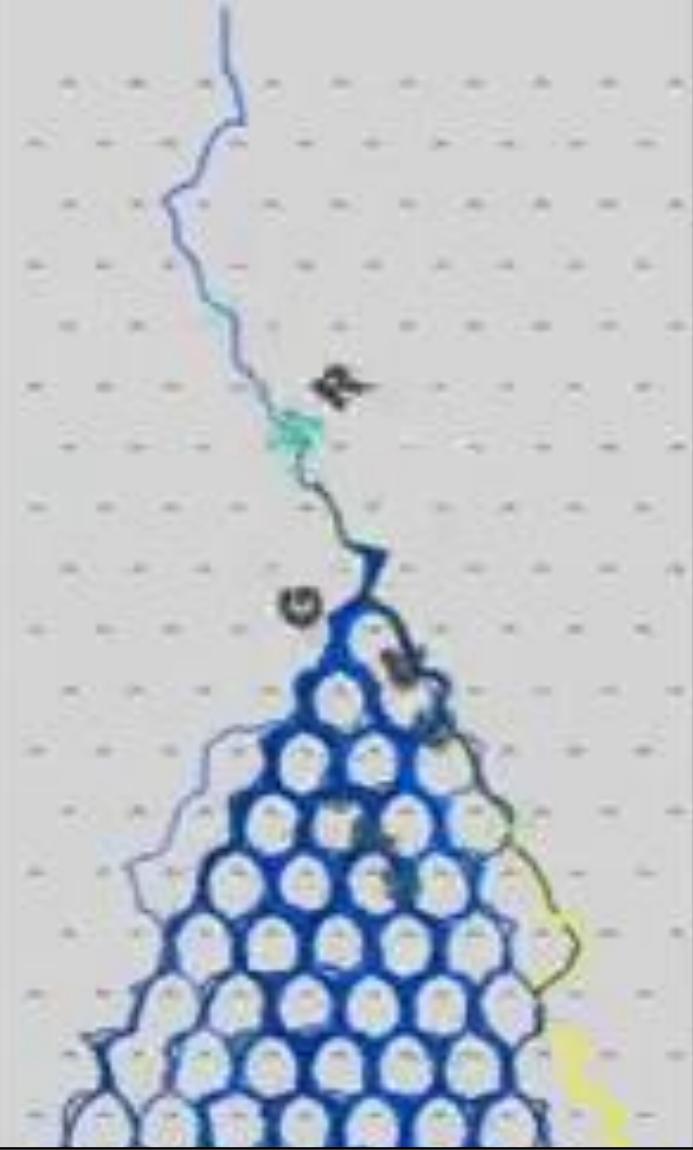
defn update (me)
  (if (and (colliding? me)
           (block-edge? (if (< vx 0)
                           (+ (x me) (x block)) (x me))
                           (- (x me) (x block)))))
      (set! (vx me) (+ vx (block-edge)))
      (set! (y me) (- (y me) speed)))
    (set! (y me) (- (y me) speed))
    (set! (jumping? me) false))

defn reset (me)
  (if (> (y me) 450)
      (set! (vx me) 0)
      (set! (y me) 450)
      (set! (vy me) 0))

me)

defn update-player (me)
  (set! (vx me) 0)
  (set! (vy me) 0)
  (set! (jumping? me) false)
  (set! (colliding? me) false)
  (set! (block-edge? me) false)
```





Many-Worlds Browsing for Control of Multibody Dynamics
Twigg et al. [SIGGRAPH 2007]

5.0

10.0

User Interfaces for Live Programming

15.0

20.0

25.0

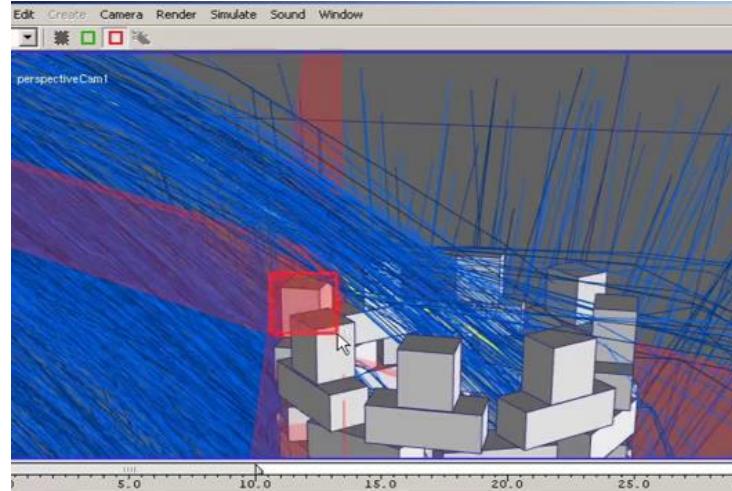
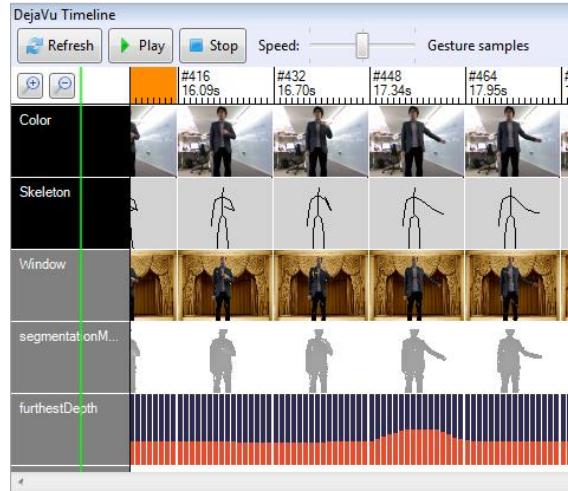
3



Rktcr, McCann (TCHOW)
[2013]

“Live” is not always about “now”

- UIs for exploring and modifying the past
- UIs for predicting and choosing the future
- **Absolute vs semantic timeline**



Absolute time vs semantic time

The screenshot shows the DejaVu software interface. At the top, there is a code editor window for "KinectDress - SharpDevelop" with a file named "Window1.xaml.cs". The code handles Kinect frame ready events and performs gesture recognition based on joint positions and angles. Below the code editor is a "DejaVu Canvas" window containing a video feed of a person, a hand icon with a "swipe" gesture diagram, and two boxes showing "0.991" and "True" next to "handPosition", and "1.888" and "False" next to "elbowAngle". To the right of the canvas is a "DejaVu Timeline" window showing a sequence of frames from a session, with a play/stop button and a speed slider set to 1.0. The timeline displays four tracks: "Color", "Skeleton", "handPosition" (red wavy line), and "swiped" (red wavy line). On the far left, there are "Errors" and "Output" tabs.

DejaVu [UIST 2012]

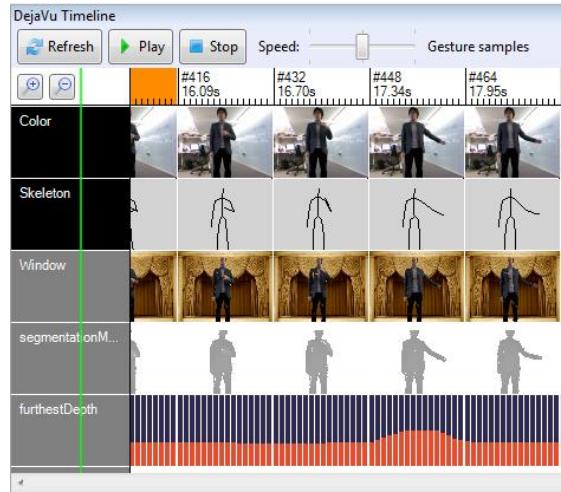
The screenshot shows the TextAlive software interface. The main area features large, semi-transparent text elements ("da", "n", "breezes") and a central play button. Below the text are several small video frames labeled "Gesture samples" and "Session 38, 2706 frames, 4m55.45s". A "Parameter tweaking (Slide In / Word)" section at the bottom includes a "Sliding Speed" slider set to 408, and a "Song" dropdown set to "I think of you". On the right side, there is a code editor window with a file named "SlideInWord.js" containing JavaScript-like code for "SlideInWord" functions. The footer of the application includes copyright information and API documents.

TextAlive [CHI 2015]

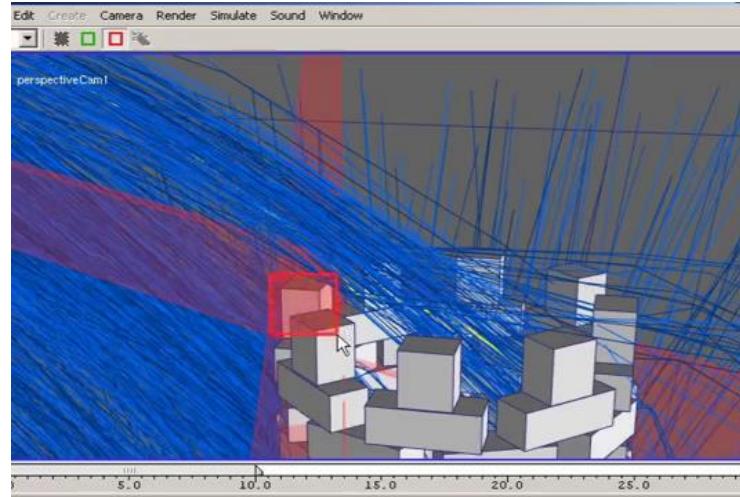
When designing live programming systems ...

Try providing good sense of time

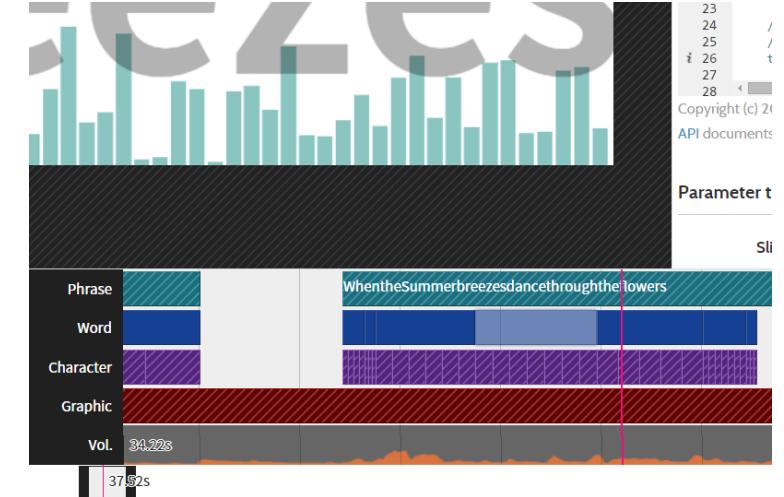
- Enable time travel to find critical timings in the history
- Allow editing the code and program input to explore futures



Replay & Refresh
Superspeed & slowmo



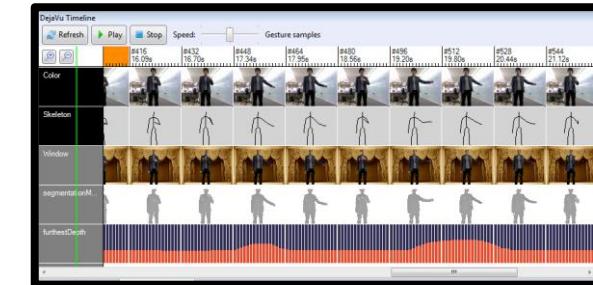
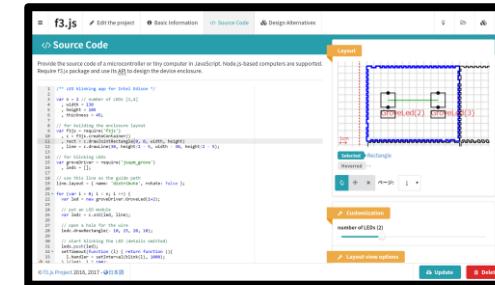
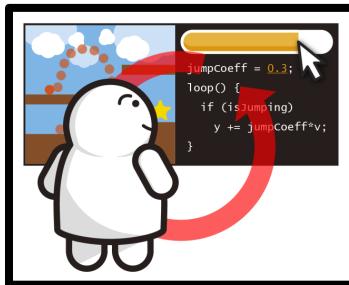
“Many worlds”
Stroboscopic visualization



Timeline
for absolute/semantic time

Today, I'm going to talk about ...

- What is Live Programming?
- UIs for Live Programming with end-users
- UIs for Live Programming of this material world
- UIs for Live Programming with time travel
- **Live Programming as User Interface research**



Live Programming research as User Interface research

- Don't be afraid to be domain-specific
- How about making the ladder of expertise?
- Deceiving users' perception is a good thing
- Try providing good sense of time



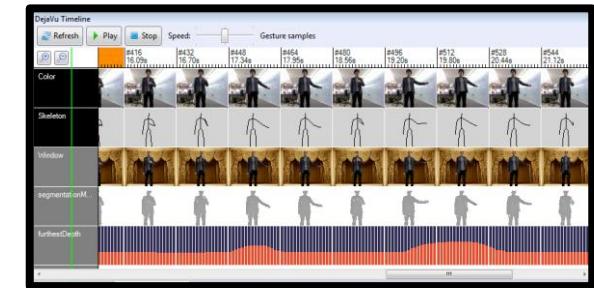
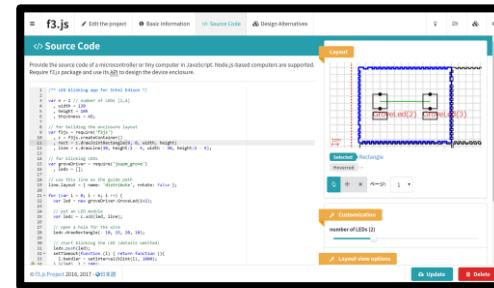
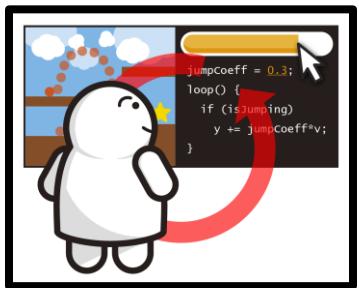
It's **not only** about language design,
a single user, a single UI,
but about **designing the whole experience**

User Interfaces for Live Programming

Jun Kato

<https://junkato.jp>

Researcher, 



LIVE 2017 Keynote, 10/24/2017

References

1. Jun Kato, Masataka Goto, "f3.js: A Parametric Design Tool for Physical Computing Devices for Both Interaction Designers and End-users", In *Proceedings of the 2017 Conference on Designing Interactive Systems*. pp.1099-1110, 2017.
2. Jun Kato, Masataka Goto, "Live Tuning: Expanding Live Programming Benefits to Non-Programmers", In *Proceedings of the Second Workshop on Live Programming Systems*. 2016.
3. Jun Kato, Takeo Igarashi, Masataka Goto, "Programming with Examples to Develop Data-Intensive User Interfaces", In *Computer* 49(7). pp.34-42, Jul. 2016. **Special Issue on 21st User Interfaces**.
4. Jun Kato, Tomoyasu Nakano, Masataka Goto, "TextAlive: Integrated Design Environment for Kinetic Typography", In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. pp.3403-3412, 2015. **ACM CHI 2015 Best Paper Honorable Mention Award**.
5. Jun Kato, Takeo Igarashi, "VisionSketch: Integrated Support for Example-centric Programming of Image Processing Applications", In *Proceedings of the 2014 Graphics Interface Conference*. pp.115-122, 2014.
6. Sebastian Burckhardt, Manuel Fahndrich, Peli Halleux, Sean McDermid, Michal Moskal, Nikolai Tillmann, Jun Kato, "It's Alive! Continuous Feedback in UI Programming", In *PLDI '13: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. pp.95-104, 2013.
7. Jun Kato, Daisuke Sakamoto, Takeo Igarashi, "Picode: Inline Photos Representing Posture Data in Source Code", In *CHI '13: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. pp.3097-3100, Apr. 2013. **ACM CHI 2013 Best Paper Honorable Mention Award**.
8. Jun Kato, Sean McDermid, Xiang Cao, "DejaVu: Integrated Support for Developing Interactive Camera-Based Programs", In *UIST '12: Proceedings of the 25th annual ACM symposium on User Interface Software and Technology*. pp.189-196, Oct. 2012.