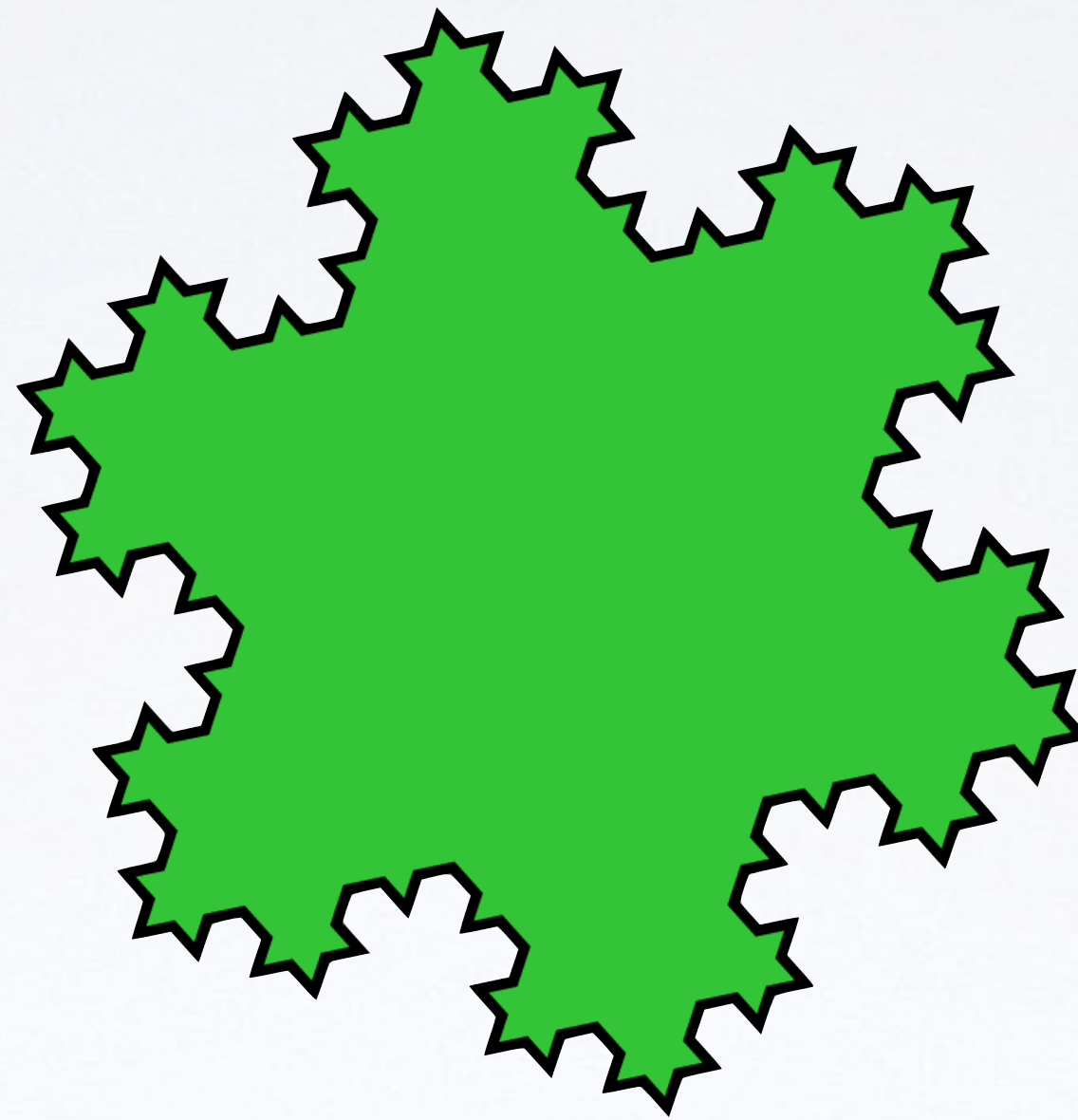


Sketch-n-Sketch: Output-Directed Programming for SVG

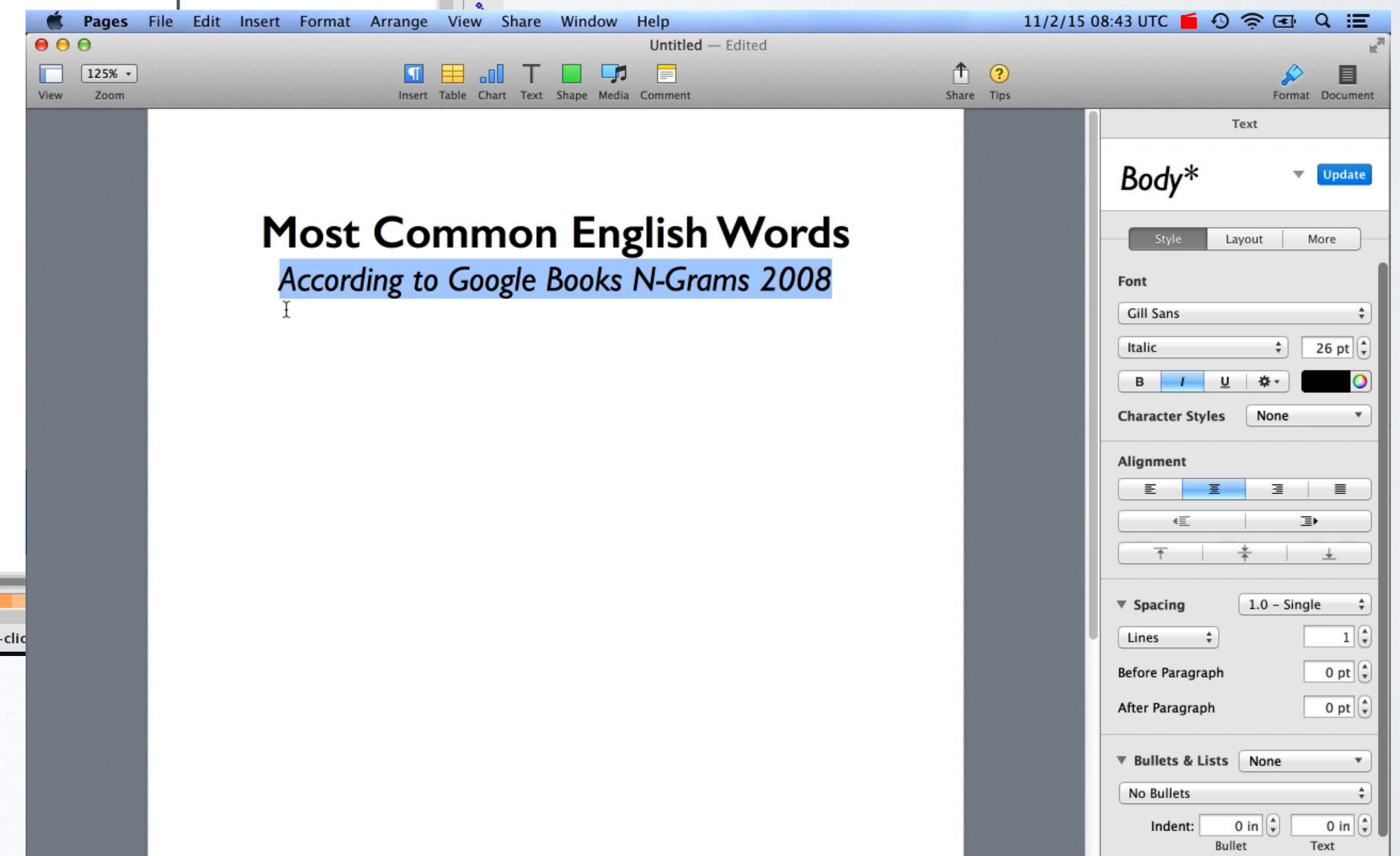
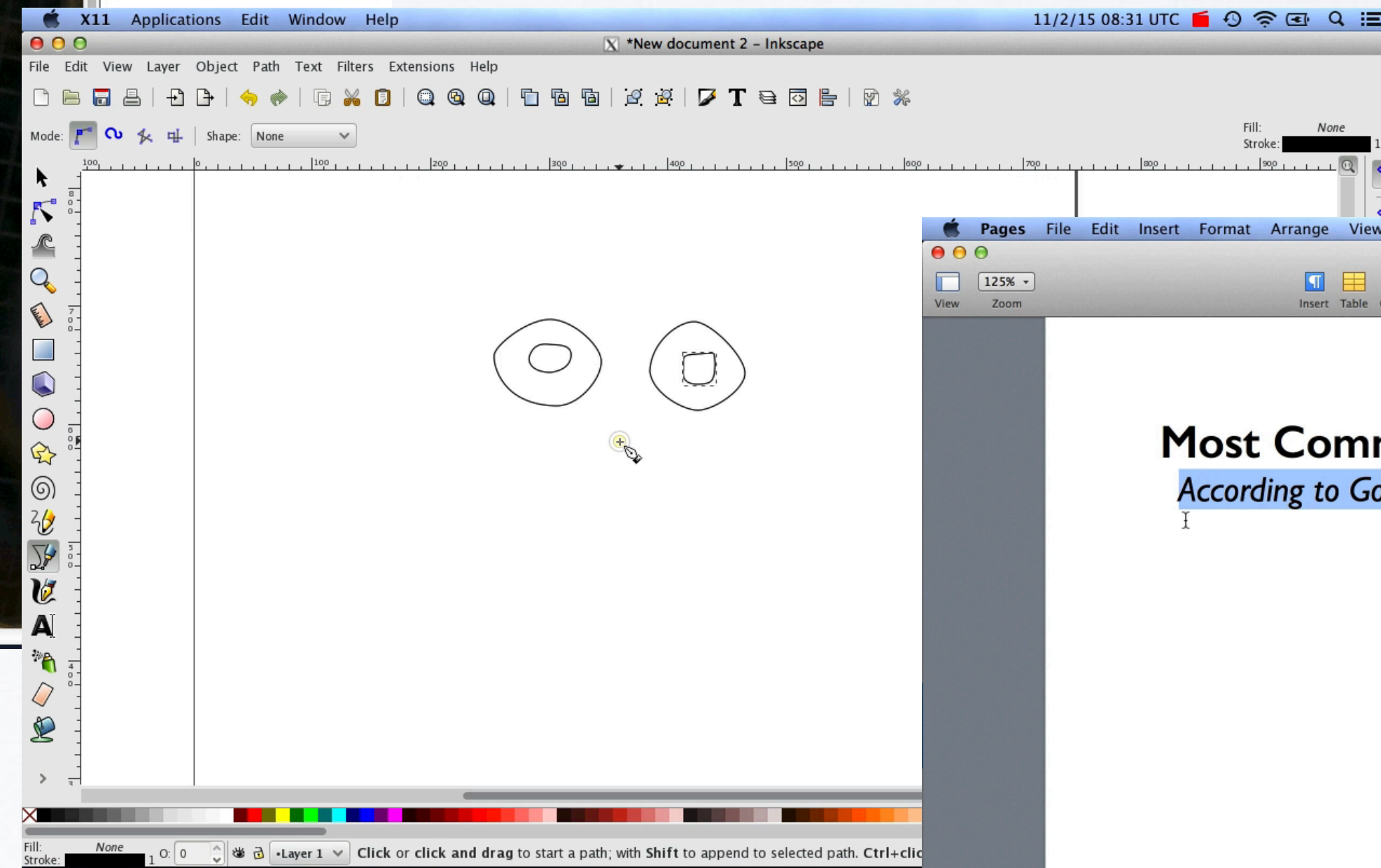
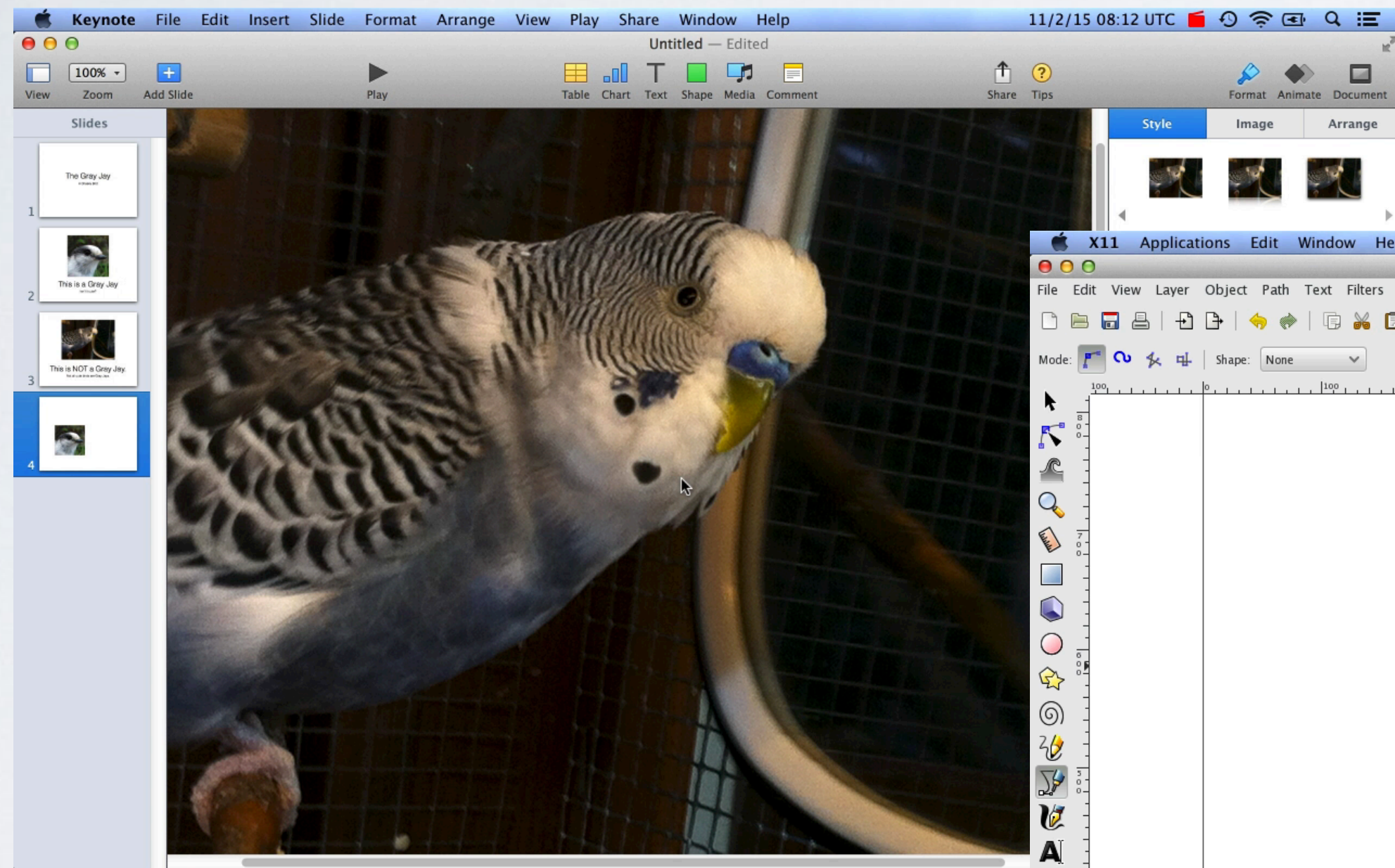


Brian Hempel, Justin Lubin, Ravi Chugh

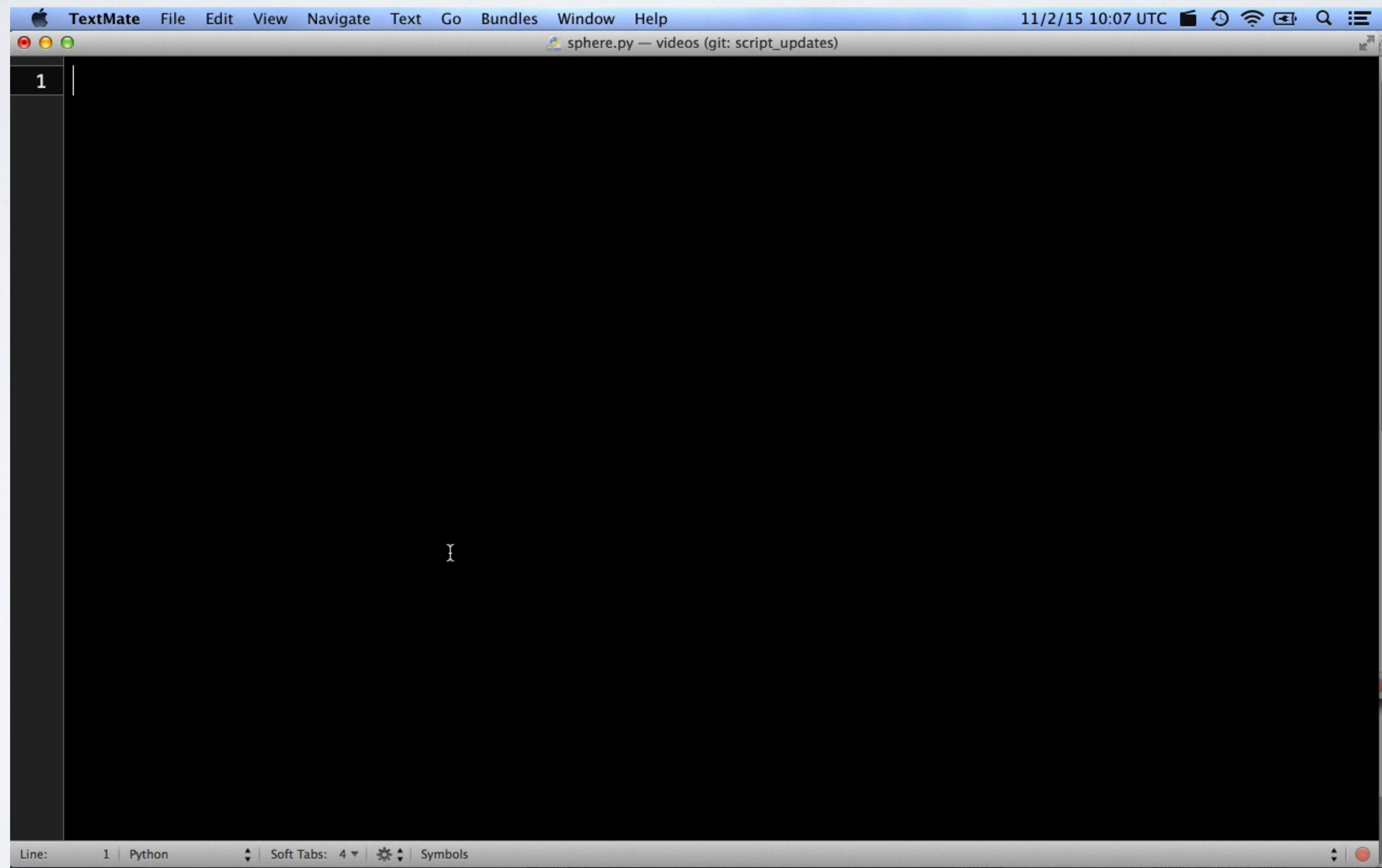


THE UNIVERSITY OF
CHICAGO

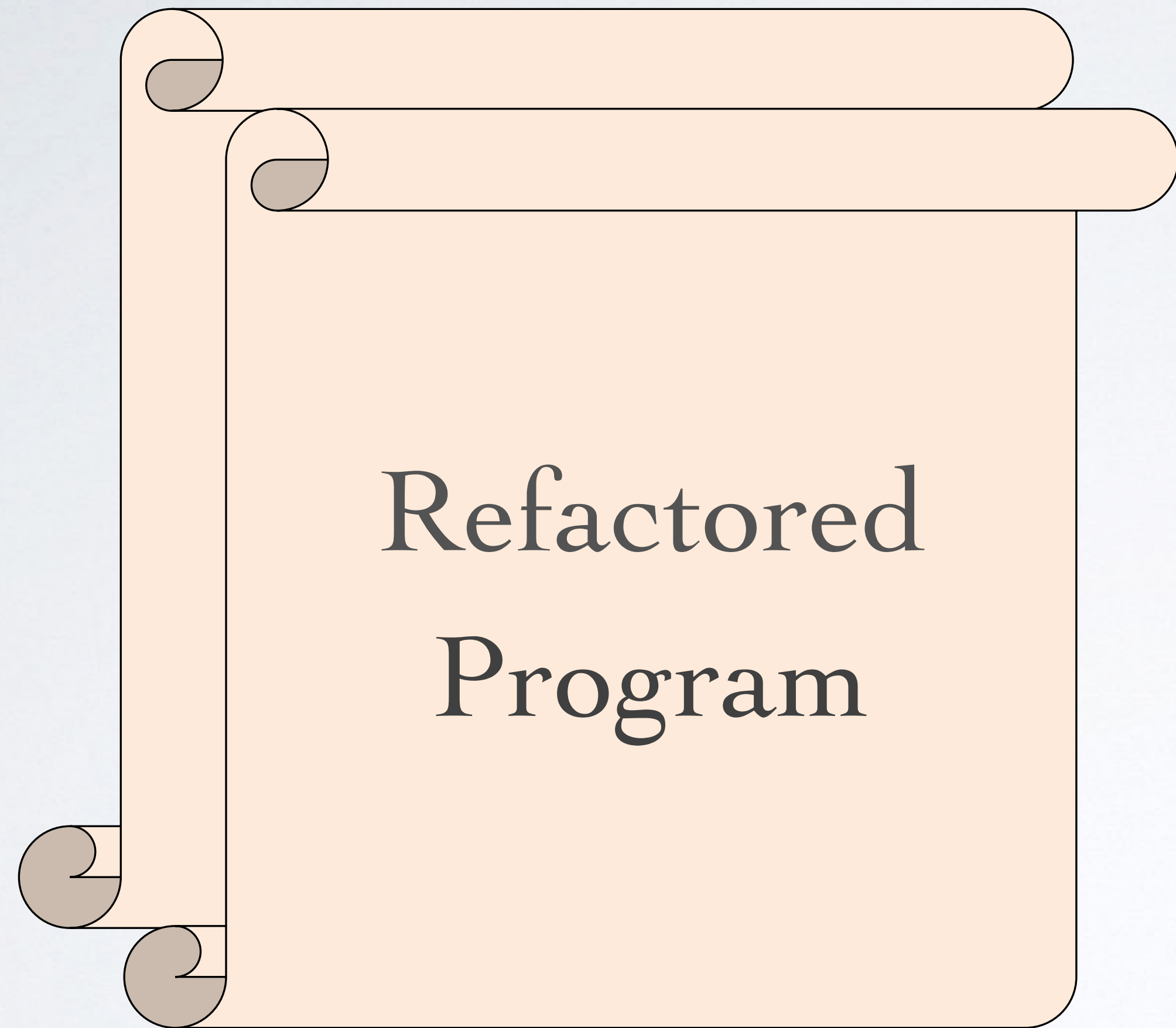
Direct Manipulation is Everywhere.



Programming



Programming + Direct Manipulation?



Ordinary, Text-Based Programming

+

Direct Manipulation on Output

=

Output-Directed Programming

Prior Output-Directed Programming

Hanna (2005)
Vital

Exercise!

- * First copy some number from the list.
- double-click in the list to select an element, then press the Copy button;
- double-click in a tree Tip to select an element, then press the Paste button.

* Now change the structure of the tree:

- double-click in a tree Node to select a subtree, then press the Copy button;

Automating Presentation Changes in Dynamic Web Applications via Collaborative Hybrid Analysis

Xiaoyin Wang¹, Lu Zhang¹, Tao Xie², Yingfei Xiong¹, Hong Mei¹
¹Key Laboratory of High Confidence Software Technologies (Peking University), MOE, China
²Department of Computer Science, North Carolina State University, USA
 {wangxy06,zhanglu,xiongyf04,meih}@sei.pku.edu.cn, xie@csc.ncsu.edu

ABSTRACT
 Web applications are becoming increasingly popular nowadays. During the development and evolution of a web application, a typical type of tasks is to change the presentation of the web application, such as correcting display errors, adding user-interface controls, or changing appearance styles. To change the presentation of a static web page, developers are able to modify the HTML text of the web page using a graphical web-page editor. However, to change the presentation of a dynamic web application, instead of using a graphical web-page editor to directly modify generated web pages, developers need to modify the code that generates the web pages. As manually performing such changes in dynamic web applications is tedious and error-prone, this paper introduces a tool based on collaborative hybrid analysis that enables static analysis and dynamic analysis to facilitate developers to perform presentation changes in dynamic web applications.

1. INTRODUCTION
 Recently, web applications are becoming increasingly popular due to easier access to the Internet. Various researchers have developed techniques to facilitate the development and evolution of web applications, such as testing web applications [4, 3, 19], static checking for bugs in web applications [10, 33], and refactoring web applications [31, 18]. A typical type of daily tasks during the development and evolution of web applications is presentation changes, which are modifications made to change the appearance of web pages. Typical presentation changes in web applications include correction of display errors, adding user-interface controls, and so on. According to our statistics, about 30% bug reports from three real-world web applications are about presentation changes. Bug reports are primarily occur in early evolution stages of the applications. On a static web page, it is straightforward to perform a presentation change by directly modifying the HTML text. However, in a dynamic web application, the presentation changes are generated by the server-side code. To change the presentation of a dynamic web application, developers need to modify the code that generates the web pages. As manually performing such changes in dynamic web applications is tedious and error-prone, this paper introduces a tool based on collaborative hybrid analysis that enables static analysis and dynamic analysis to facilitate developers to perform presentation changes in dynamic web applications.

Wang et al. (2012)

APX

McDermid (2015, 2016)

```

for y in 0 until 4:
  var p = 0
  val r = 0
  draw*circle(p, r, randColor)
  var v = [50, 0]
  val f = [0, 50]
  log y = p + v
  on tick
    val p1 = p + (v ÷ 50)
    val r1 = [r, r]
    val v1 = inflect1(p1, rr)
    val f1 = [x*v1, f ÷ 50]
  
```

Chugh et al. (2016)
Live Synchronization SnS

```

1 (def [x0 y0 w h sep amp] [50 120 20 90 30 60])
2 (def n 121{3-30})
3 (def boxi (v
4 (let xi (+ x0 (* i sep))
5 (let yi (- y0 (* amp (sin (* i (/ twoPi n))))))
6 (rect "lightblue" xi yi w h))))
7
8 (svg (map boxi (zeroTo n)))
  
```

Schuster & Flanagan (2016)

```

function keyup(evt) {
  str = evt.target.value;
}

function render() {
  return <div>
    <input value={str} onkeyup={keyup} />
    <p>
      {str.replace(/keyboard/g, "Leopard")}
    </p>
  </div>;
}
  
```

Kwok & Webster (2016)
Carbide Alpha

```

var str = ""
for(var i = 0; i < 30; i++){
  if(i % 3 == 0) str += "Fizz";
  if(i % 5 == 0) str += "Buzz";
  if(i % 3 && i % 5){
    str += i
  }
  str += ' '
}
str
  
```

Hempel & Chugh (2016)
Sketch-n-Sketch 2016

```

1 (def mug (outer_color stroke@idth320 color stroke@idth3
2 (def bounds [left top right bot]) (def rFrac 0.21)
3 (def [outer_right outer_left] [scaleBetween left right
4 (def [outer_x_radius (/ (- outer_right outer_left) 2)])
5 (def [rect2_top outer_bot outer_top] [scaleBetween top
6 (def [outer_ellipserY (/ (- outer_bot outer_top) 2)])
7
8 (def outer
9 (let [left top right bot] [(+ outer_left (* rFrac outer_
10 (let [bounds [left top right bot]
11 (let [strokeColor strokeWidth] [black stroke@idth3
12 (let [oval outer_color strokeColor strokeWidth bounds]
13 (oval outer_color strokeColor strokeWidth bounds)
14
15 (def inner
16 (let [left top right bot] [(+ outer_left (* rFrac outer_
17 (let [bounds [left top right bot]
18 (let [color strokeColor strokeWidth] [color "black" str
19 (let [oval outer_color strokeColor strokeWidth bounds]
20 (oval outer_color strokeColor strokeWidth bounds)
21
22 (def rect3
23 (let [left right bot] [scaleBetween left right 0] (+
24 (let [bounds [left rect3_top right bot]
25 (let [rectangle outer_color "black" 0 0 bounds]
26 (rectangle outer_color "black" 0 0 bounds)
27
28 (def steam (let [left top right bot]
29 (let [color strokeColor strokeWidth] [color "black" str
30 (let [oval outer_color strokeColor strokeWidth bounds]
31 (oval outer_color strokeColor strokeWidth bounds)
32
33 (let [group bounds] (concat [oval outer_color strokeColor strokeWidth bounds]
34 (let [loabs [
35 (with [16 82]
36 (with [220 11]
37 (with [72 40]
38 (with [385 493 397 484] (mug 160 0 435 0 86 5 0 48275802)
39 (with [385 493 397 484] (mug 160 0 435 0 86 5 0 48275802)
  
```

Schreiber et al. (2017)
Transmorphic

```

1 (defcomponent
2 digital-clock
3 IRender
4 (render
5 [self props submorphs]
6 (rectangle
7 (rectangle
8 {:id "rectangle",
9 :wants-hand-focus? true,
10 :border-color "black",
11 :fill "grey",
12 :border-radius 10,
13 :drop-shadow? false}
14 (text
15 {:rotation 5.502689183085366,
16 :drop-shadow? false,
17 :font-size 22.400000000000002,
18 :text-color "limegreen",
19 :allow-input false,
20 :text "April 4th",
21 :position {x 10, y 18},
22 :text-stroke "black",
23 :font-family "serif",
24 :font-size 22.400000000000002}
  
```

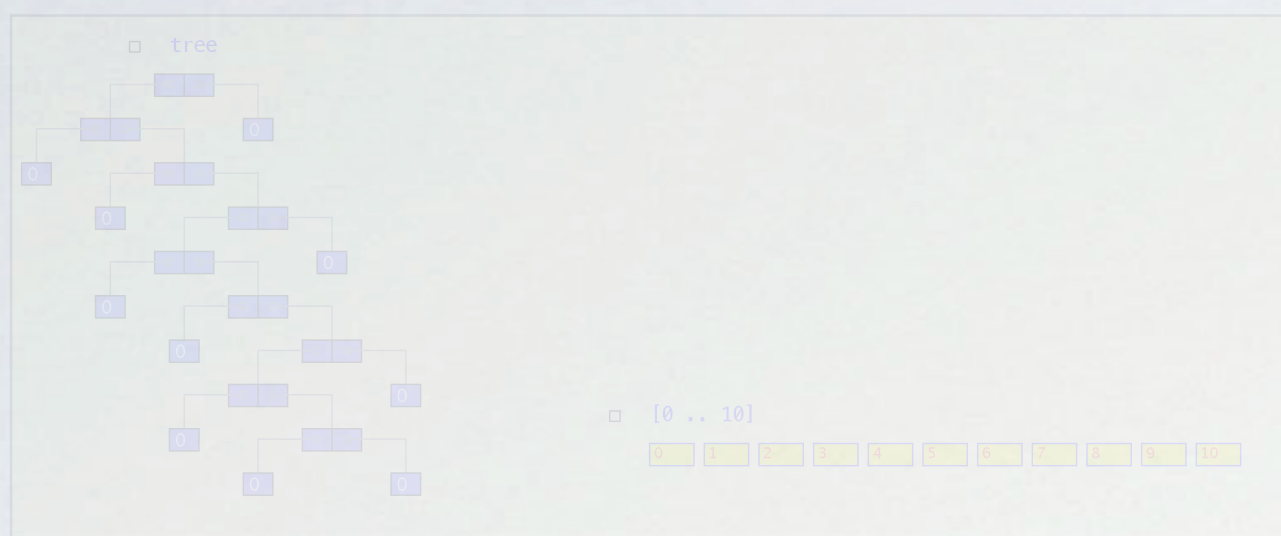
Mayer et al. (2018)
Bidirectional SnS

```

1 -- Like the previous example, but with the Tab
2 -- module included in this file.
3
4 TableWithButtons =
5 let wrapData rows =
6 let blankRow =
7 let numColumns =
8 case rows of
9 [] -> 0
10 row: _ -> List.length row
11 in
12 list.repeat numColumns "?"
13 in
14 Update.apply
15 { apply
16 List.map (fmap) (Update.Freeze
17 update
  
```

State	Capital
Alabama	Montgomery, AL
Alaska	Juneau, AK
Arizona	Phoenix, AZ
Arkansas	Little Rock, AR
California	Sacramento, CA
Colorado	Denver, CO
Connecticut	Hartford, CT

Prior Output-Directed Programming



Hanna (2005)
Vital

Exercise!

- First copy some numbers from the list into the Tip of the tree.
 - double-click in the list to select an element, then press the Copy button;
 - double-click in a tree Tip to select an element, then press the Paste button.
- Now change the structure of the tree:
 - double-click in a tree Node to select a subtree, then press the Copy button;

Automating Presentation Changes in Dynamic Web Applications via Collaborative Hybrid Analysis

Xiaoyin Wang¹, Lu Zhang¹, Tao Xie², Yingfei Xiong¹, Hong Mei¹
¹Key Laboratory of High Confidence Software Technologies (Peking University), MOE, China
²Department of Computer Science, North Carolina State University, USA
 {wangxy06,zhanglu,xiongyf04,meih}@sei.pku.edu.cn, xie@csc.ncsu.edu

ABSTRACT
 Web applications are becoming increasingly popular nowadays. During the development and evolution of a web application, a typical type of tasks is to change the presentation of the web application, such as correcting display errors, adding user-interface controls, or changing appearance styles. To change the presentation of a static web page, developers are able to modify the HTML text of the web page using a graphical web-page editor. However, to change the presentation of a dynamic web application, instead of using a graphical web-page editor to directly modify generated web pages, developers need to modify the code that generates the web pages. As manually performing presentation changes in dynamic web applications is tedious and error-prone, we propose a novel approach based on *collaborative hybrid analysis* that combines static analysis and dynamic analysis to facilitate developers to perform presentation changes.

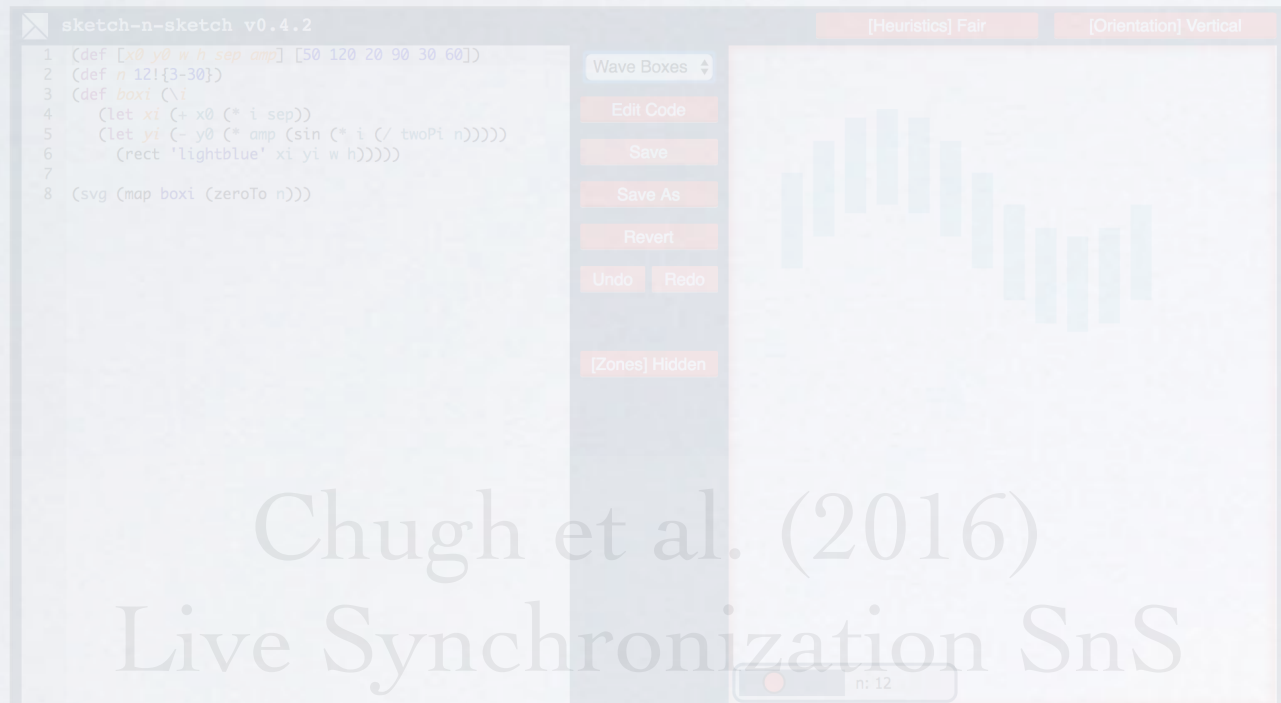
1. INTRODUCTION
 Recently, web applications are becoming increasingly popular due to easier access to the Internet. Various researchers have developed techniques to facilitate the development and evolution of web applications, such as testing web applications [4, 3, 19], static checking for bugs in web applications [10, 33], and refactoring web applications [31, 18]. A typical type of daily tasks during the development and evolution of web applications is presentation changes, which are modifications made to change the appearance of web pages. Typical presentation changes in web applications include correction of display errors, adding user interface controls, etc. According to our investigation of 100 bug reports from three real-world web applications, about 7% of the bug reports are presentation changes, and these presentation change bug reports primarily occur in early evolution stages of the applications. On a static web page, it is straightforward to perform a presenta-

```

for y in 0 until 4:
  var p = 0
  var r = 0
  draw*circle(p, r, randColor)
  var v = [50, 0]
  var f = [0, 50]
  log(v + p + v)
  or
  = p + (v + 50)
  = [r, r]
  = infect1(p, rr)
  = r * v1 + f + 50
  
```

McDermid (2015)
 APX

0	(172.1, 275.49)	(50, 75)
1	(255.89, 195.48)	(50, 75)
2	(333.41, 219.4)	(50, 75)
	(282.02, 241.98)	(50, 75)



Chugh et al. (2016)
 Live Synchronization SnS



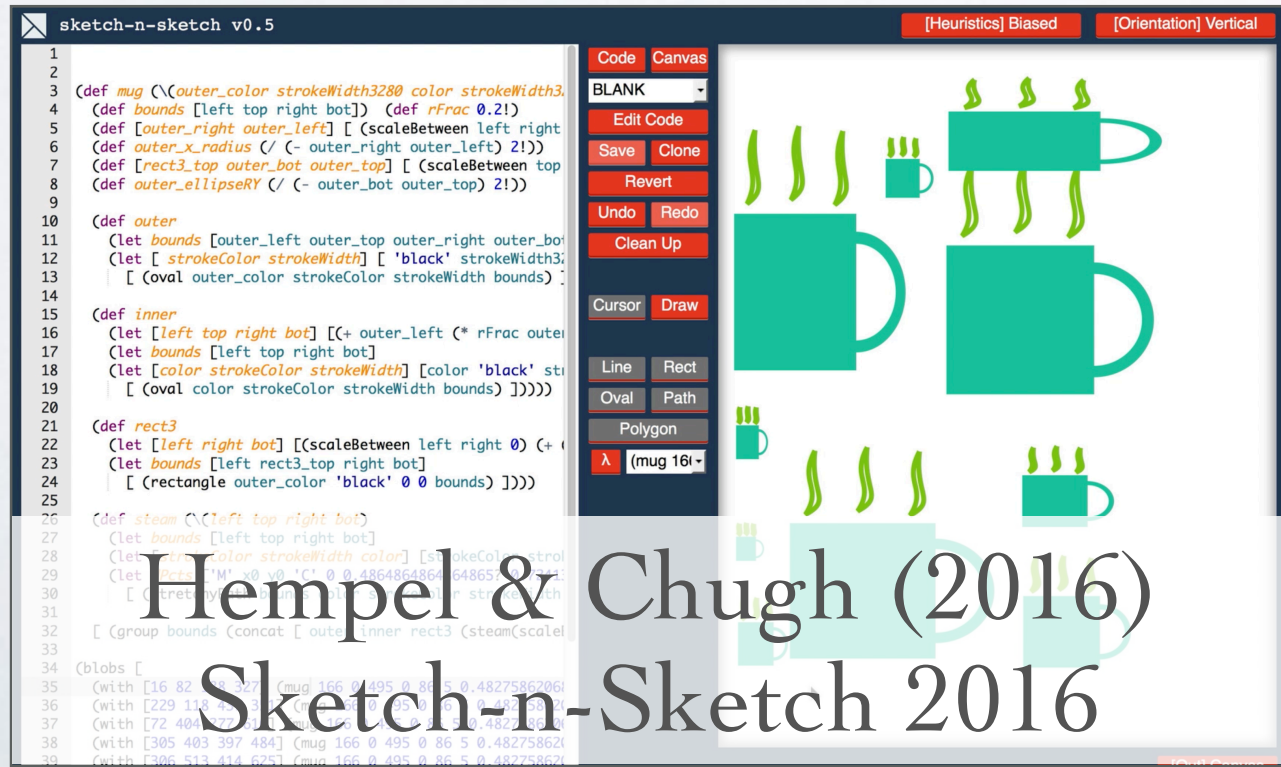
Schuster & Flanagan (2016)

```

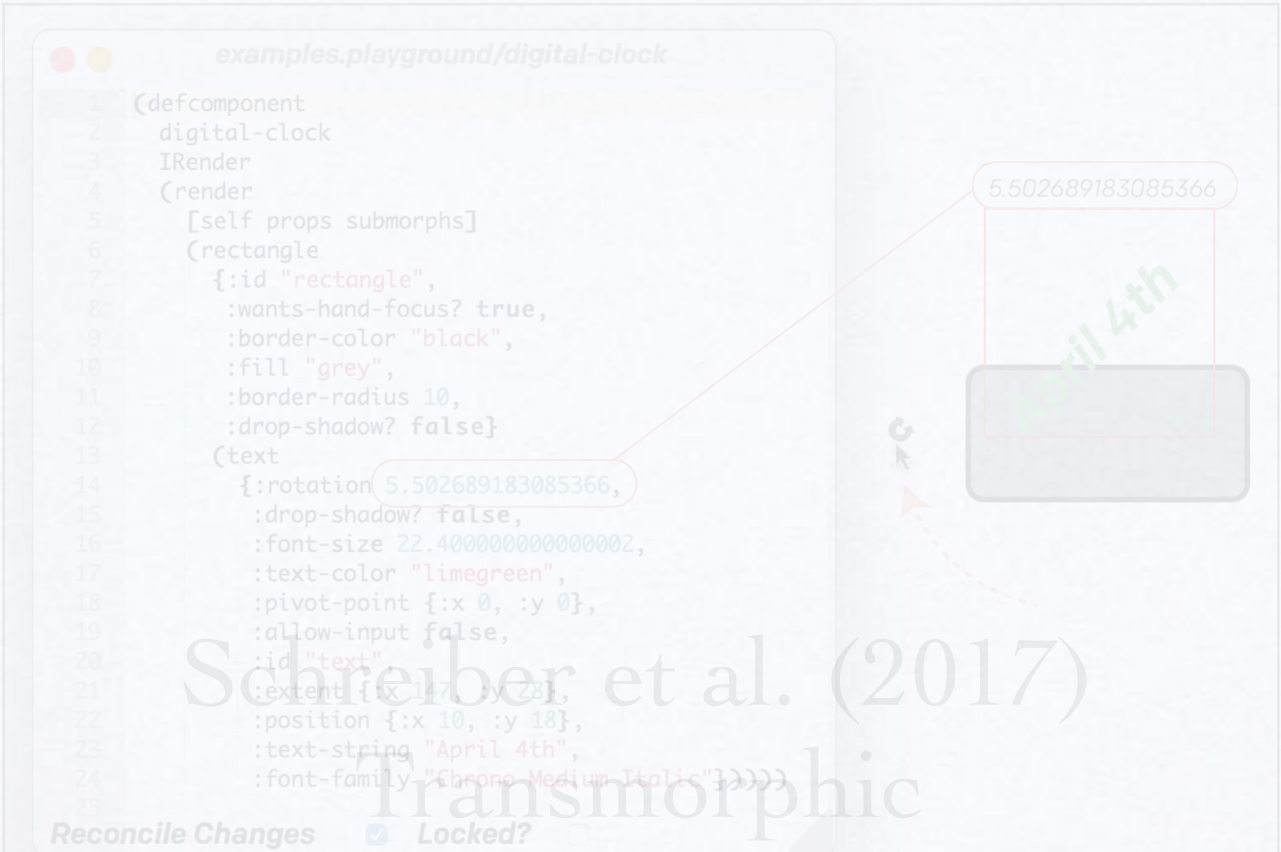
var str = ""
for(var i = 0; i < 30; i++){
  if(i % 3 == 0) str += "Fizz";
  if(i % 5 == 0) str += "Buzz";
  if(i % 3 && i % 5){
    str += i
  }
  str += ' '
}
str
  
```

Kwok & Webster (2016)
 Carbide Alpha

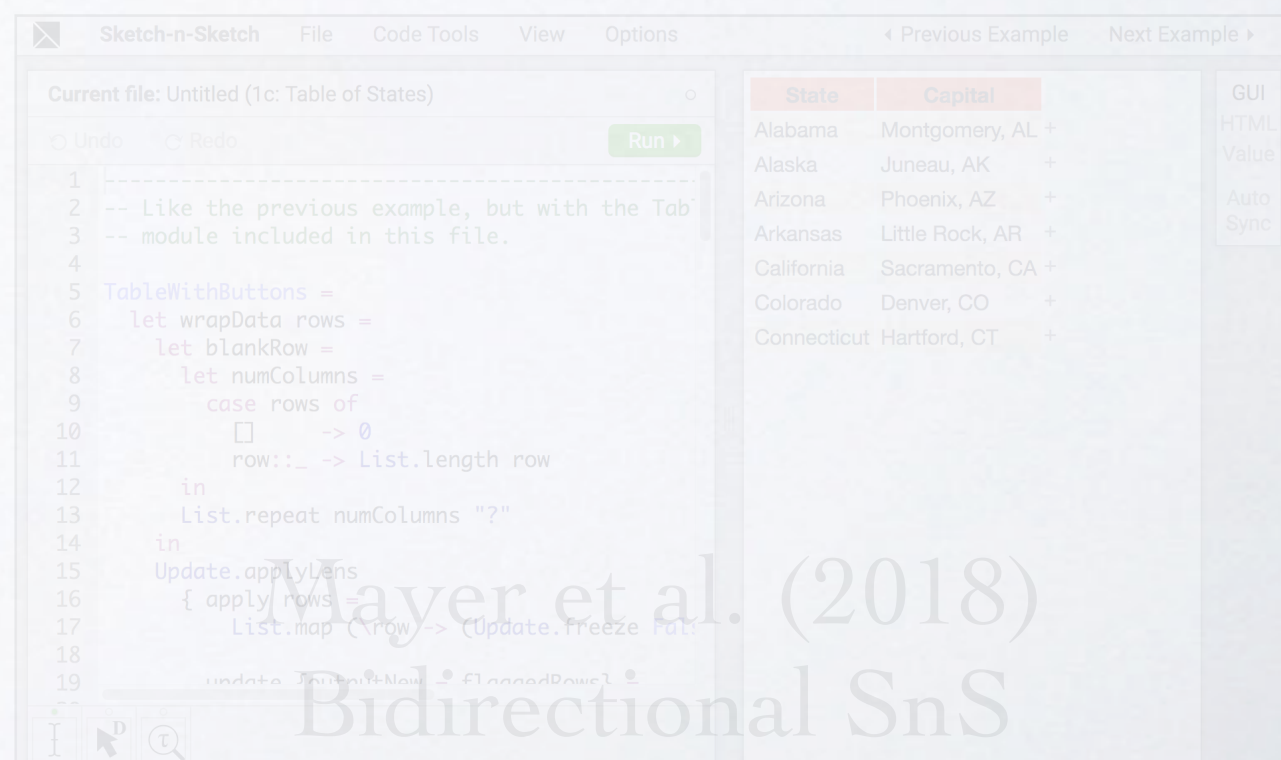
FizzBuzz 1 2 Fizz 4 Buzz Fizz 7 8
 Fizz Buzz 11 Fizz 14 Fizz Buzz 16
 Buzz 19 Fizz 22 23 Fizz
 Buzz 26 Fizz 28 29



Hempel & Chugh (2016)
 Sketch-n-Sketch 2016



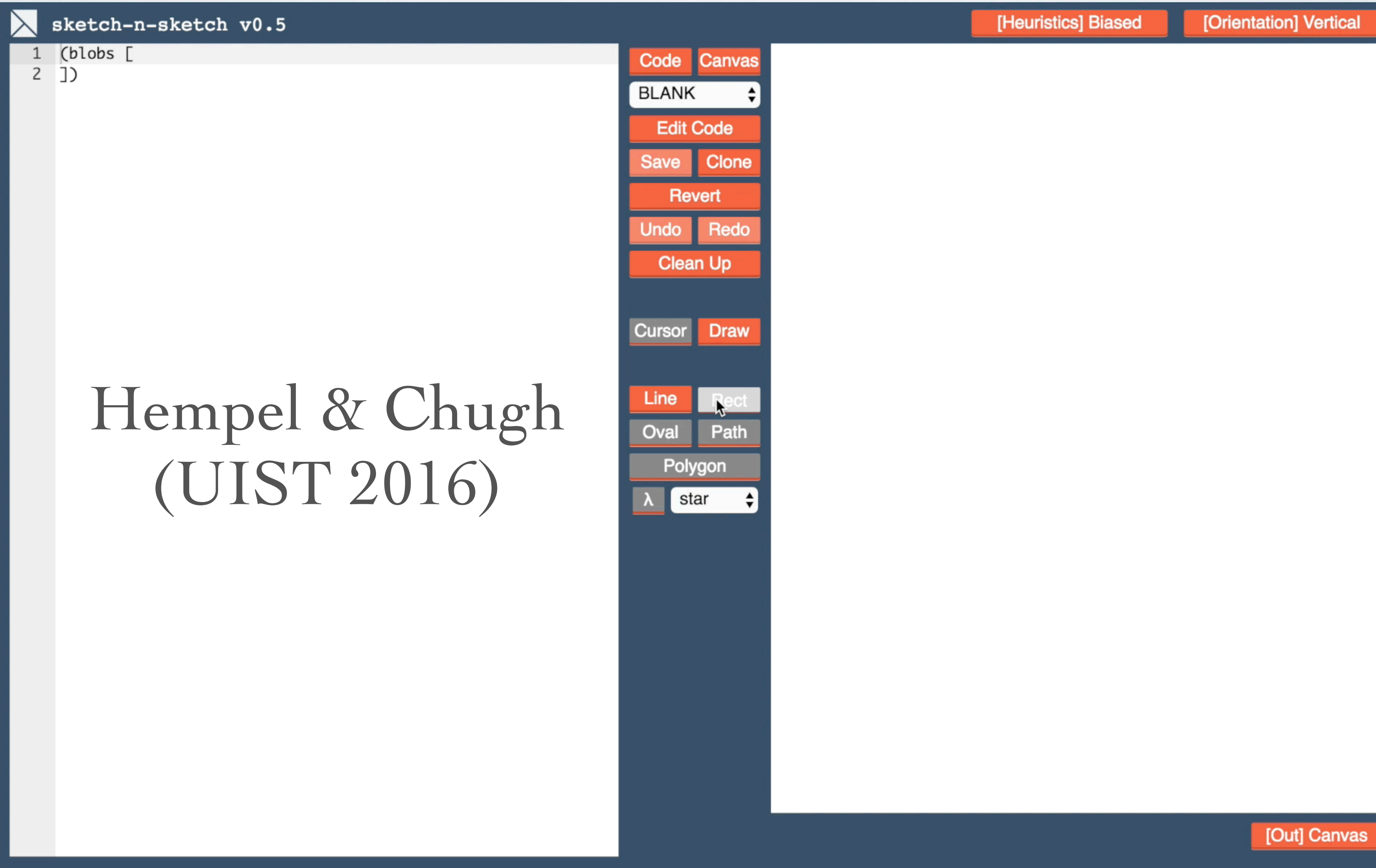
Schreiber et al. (2017)
 Transmorphic



Mayer et al. (2018)
 Bidirectional SnS

State	Capital	GUI HTML Value
Alabama	Montgomery, AL	+
Alaska	Juneau, AK	+
Arizona	Phoenix, AZ	+
Arkansas	Little Rock, AR	+
California	Sacramento, CA	+
Colorado	Denver, CO	+
Connecticut	Hartford, CT	+

Building on Sketch-n-Sketch 2016



Building on Sketch-n-Sketch 2016

```
top right bot] [107 147 290 318]  
s [left top right bot]  
371  
ngle color 'black' 0 0 bounds) ])))))
```

```
l x2 y2] [112 117 274 275]  
r width] [294 5{0-40}]  
color width x1 y1 x2 y2) ])))))
```

```
l x2 y2] [58 280 170 208]  
r width] [10 5{0-40}]  
color width x1 y1 x2 y2) ])))))
```

BLANK

Edit Code

Save

Clone

Revert

Undo

Redo

Clean Up

Cursor

Draw

Line

Rect

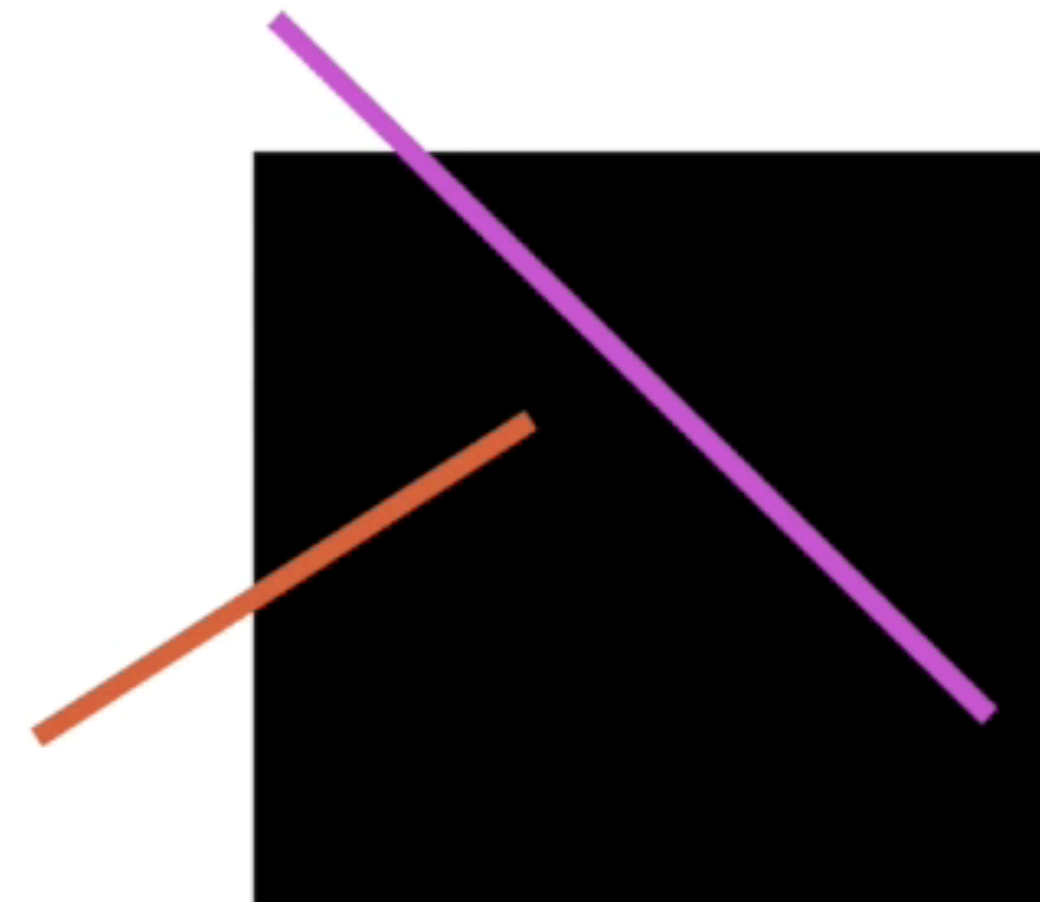
Oval

Path

Polygon

λ

star



Building on Sketch-n-Sketch 2016

```
bot]
0 0 bounds) ])))))

284 315]
-40}]
x2 y2) ])))))

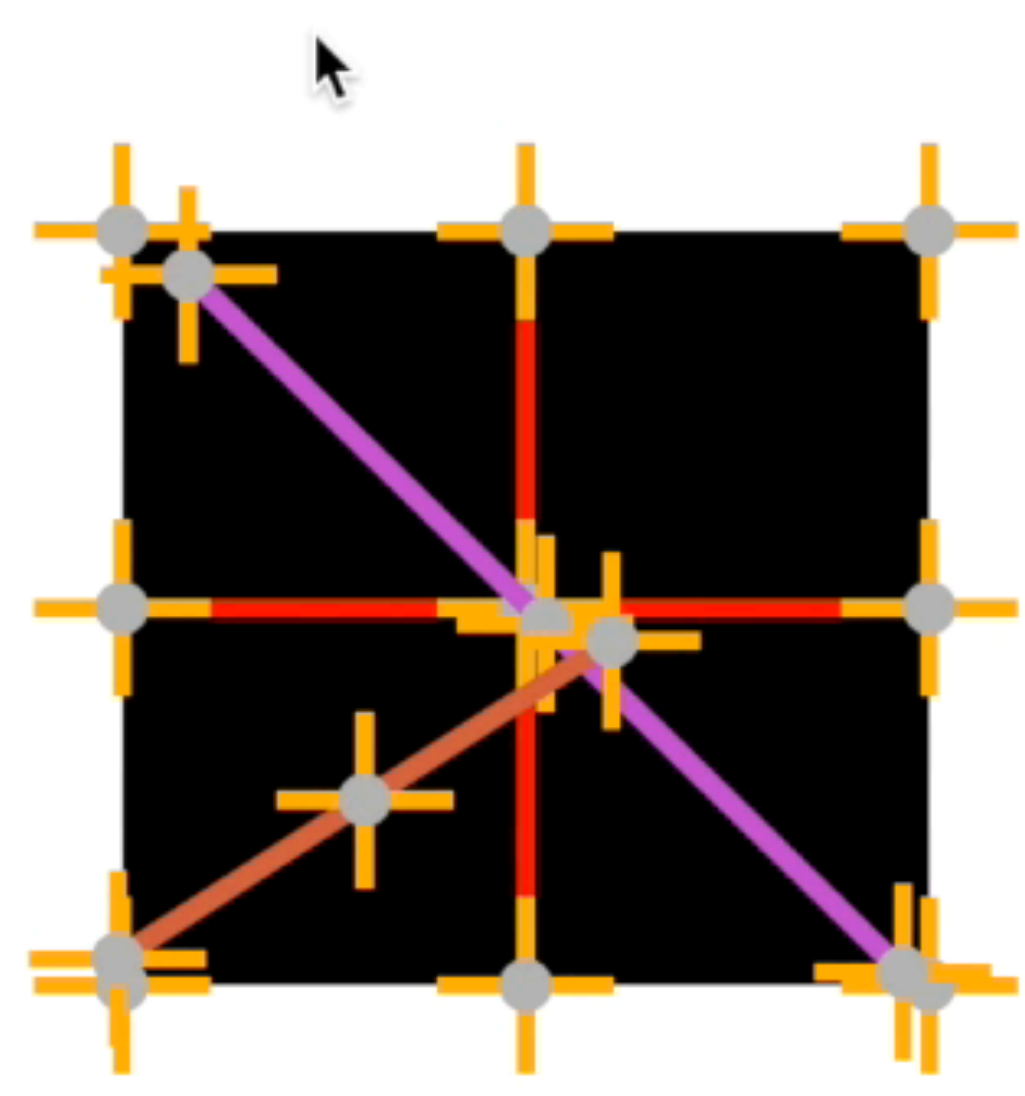
218 240]
40}]
x2 y2) ])))))
```

Edit Code
Save Clone
Revert
Undo Redo
Clean Up

Cursor Draw
Zones +
- Widgets +
- Ghosts

Click and Drag
Select Features
Select Blobs

Dig Hole
Make Equal



Building on Sketch-n-Sketch 2016

```
1  
2  
3 (def rect1  
4   (let [left top right bot] [107 147 290 318])  
5     (let bounds [left top right bot]  
6       (let color 371  
7         [ (rectangle color 'black' 0 0 bounds) ]))))  
8  
9 (def line2  
10  (let [x1 y1 x2 y2] [122 157 284 315])  
11    (let [color width] [294 5{0-40}])  
12      [ (line color width x1 y1 x2 y2) ])))  
13  
14 (def line3  
15  (let [x1 y1 x2 y2] [106 312 218 240])  
16    (let [color width] [10 5{0-40}])  
17      [ (line color width x1 y1 x2 y2) ])))
```


Building on Sketch-n-Sketch 2016

sketch-n-sketch v0.5

[Heuristics] Biased [Orientation] Vertical

```
1  
2 (def [rect1_right rect1_left] [306 107])  
3 (def [rect1_bot rect1_top] [344 147])  
4  
5 (def rect1  
6   (let bounds [rect1_left rect1_top rect1_right rect1_bot]  
7     (let color 403  
8       [ (rectangle color 'black' 0 0 bounds) ])))  
9 (def line2_color 210)  
10 (def line2_width 22{0-40})  
11  
12 (def line2  
13   [ (line line2_color line2_width rect1_left rect1_top rec  
14  
15 (def line3  
16   (let [ x2 y2] [ (* 0.5! (+ rect1_left rect1_right)) (* 0.5  
17     [ (line line2_color line2_width rect1_left rect1_bot x2  
18  
19 (blobs [  
20   rect1  
21   line2  
22   line3  
23 ])
```

Code Canvas
BLANK
Edit Code
Save Clone
Revert
Undo Redo
Clean Up
Cursor Draw
Zones +
- Widgets +
- Ghosts
Click and Drag
Select Feature
Select Blobs
Group Abs
Dupe Merge



Building on Sketch-n-Sketch 2016

sketch-n-sketch v0.5

[Heuristics] Biased [Orientation] Vertical

```
1
2
3 (def newGroup4
4   (def [left top right bot] [107 147 306 344])
5   (def bounds [left top right bot])
6   (def [rect1_right rect1_left] [right left])
7   (def [rect1_bot rect1_top] [bot top])
8   (def line2_color 210)
9   (def line2_width 22{0-40})
10
11  (def rect1
12    (let bounds [rect1_left rect1_top rect1_right rect1_bot]
13      (let color 403
14        [ (rectangle color 'black' 0 0 bounds) ])))
15
16  (def line2
17    [ (line line2_color line2_width rect1_left rect1_top r
18
19  (def line3
20    (let [ x2 y2] [ (* 0.5! (+ rect1_left rect1_right)) (* 0
21      [ (line line2_color line2_width rect1_left rect1_bot x
22
23    [ (group bounds (concat [ rect1 line2 line3 ])) ])
24
25  (blobs [
26    newGroup4
27  ])
```

Code Canvas

BLANK

Edit Code

Save Clone

Revert

Undo Redo

Clean Up

Cursor Draw

Zones +

Widgets +

Ghosts -


Click and Drag

Select Features

Select Blobs

Group Abs

Dupe Merge



Building on Sketch-n-Sketch 2016

```
1  
2  
3 (def newGroup4 (\(line2_color line2_width color [left top ri  
4   (def bounds [left top right bot])  
5   (def [rect1_right rect1_left] [right left])  
6   (def [rect1_bot rect1_top] [bot top])  
7  
8   (def rect1  
9     (let bounds [rect1_left rect1_top rect1_right rect1_bot]  
10       [ (rectangle color 'black' 0 0 bounds) ]))  
11  
12   (def line2  
13     [ (line line2_color line2_width rect1_left rect1_top r  
14  
15   (def line3  
16     (let [ x2 y2] [ (* 0.5! (+ rect1_left rect1_right)) (* 0  
17       [ (line line2_color line2_width rect1_left rect1_bot x  
18  
19   [ (group bounds (concat [ rect1 line2 line3 ])) ]))  
20  
21 (blobs [  
22   (with [107 147 306 344] (newGroup4 210 22{0-40} 403))  
23 ])
```

[Heuristics] Biased [Orientation] Vertical

Code Canvas

BLANK

Edit Code

Save Clone

Revert

Undo Redo

Clean Up

Cursor Draw

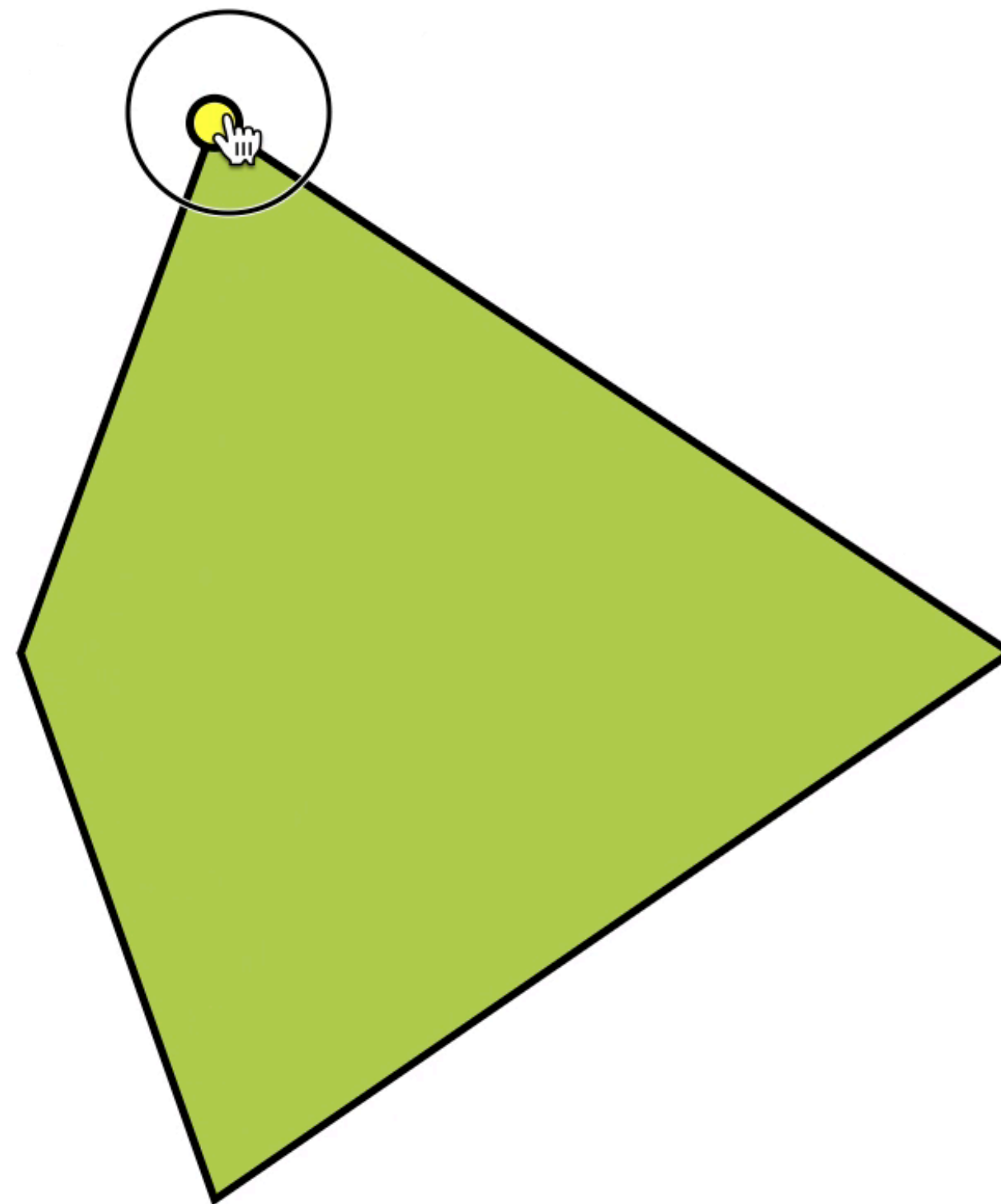
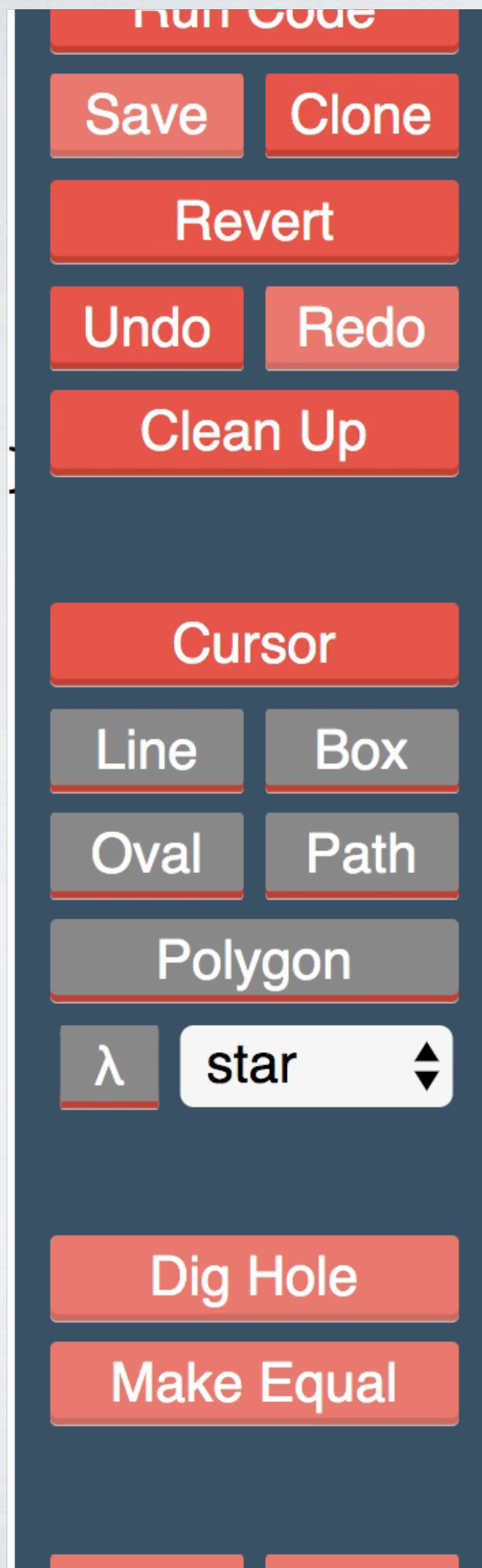
Line Rect

Oval Path

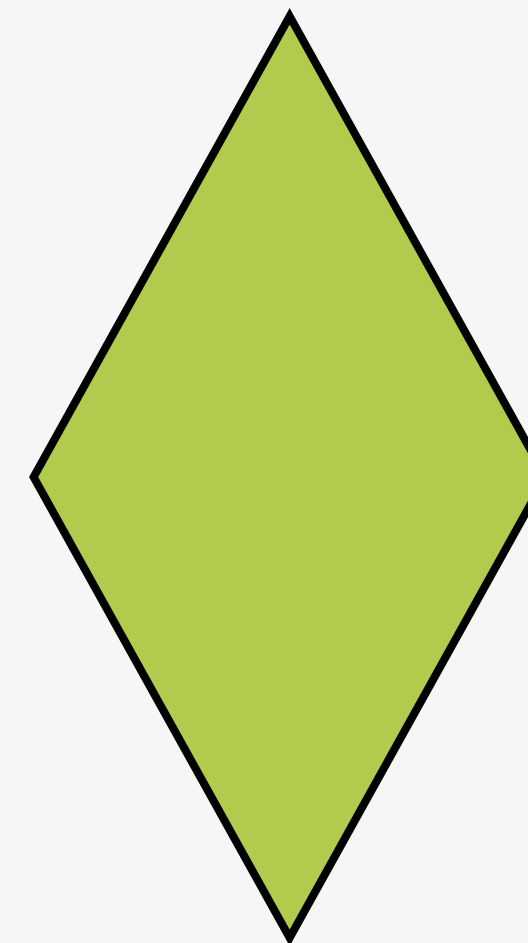
Polygon

λ (newGr

Building on Sketch-n-Sketch 2016



Want a
rhombus:



But tools
overly rigid!

Big Q

What kinds of programs can be constructed *entirely* through output manipulations?

Contribution

UI Insight

DM on More Than Output!

Intermediate Value
Widgets

Expression Focusing

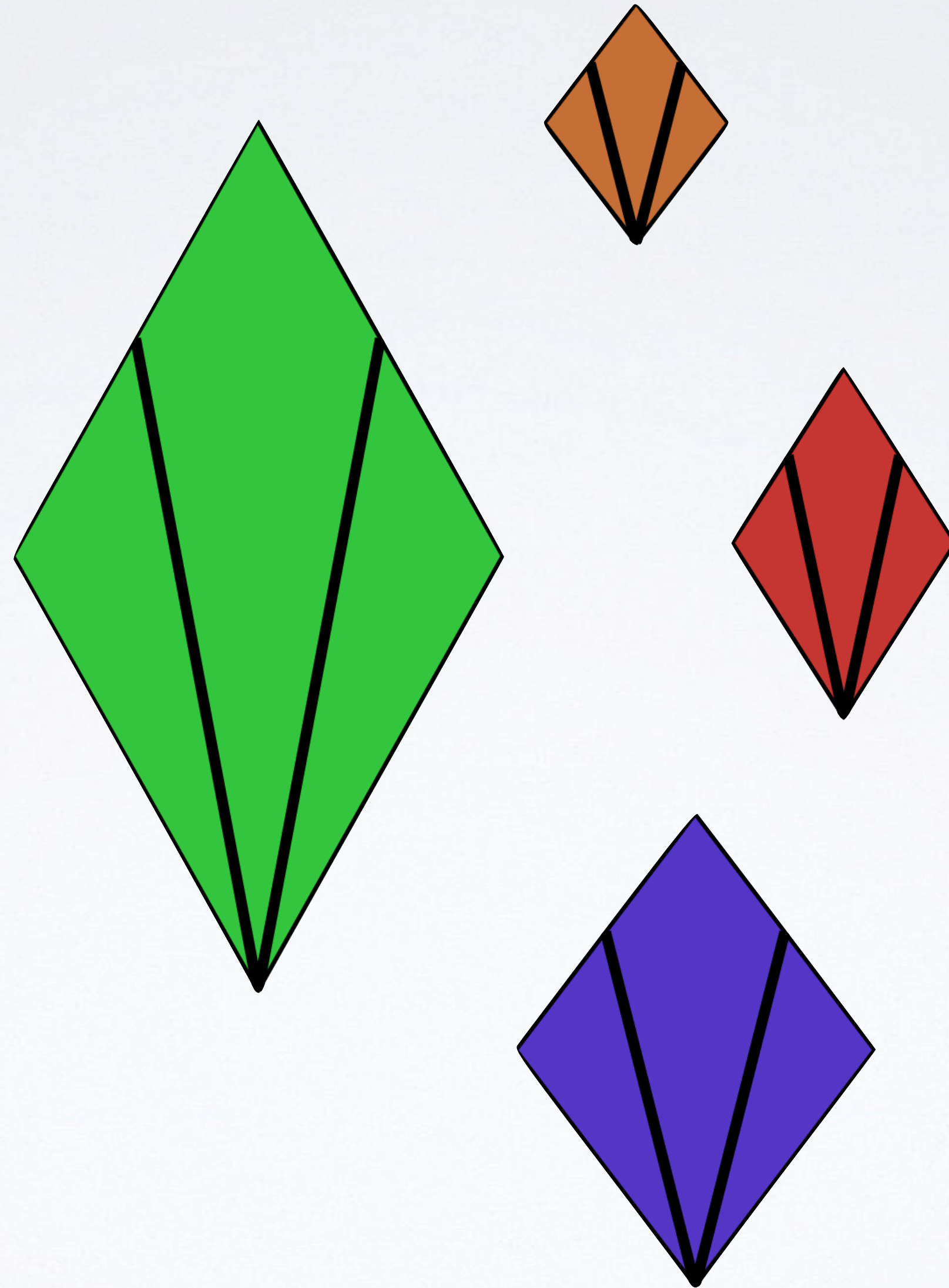
PL Insight

Generic Tools, Too!

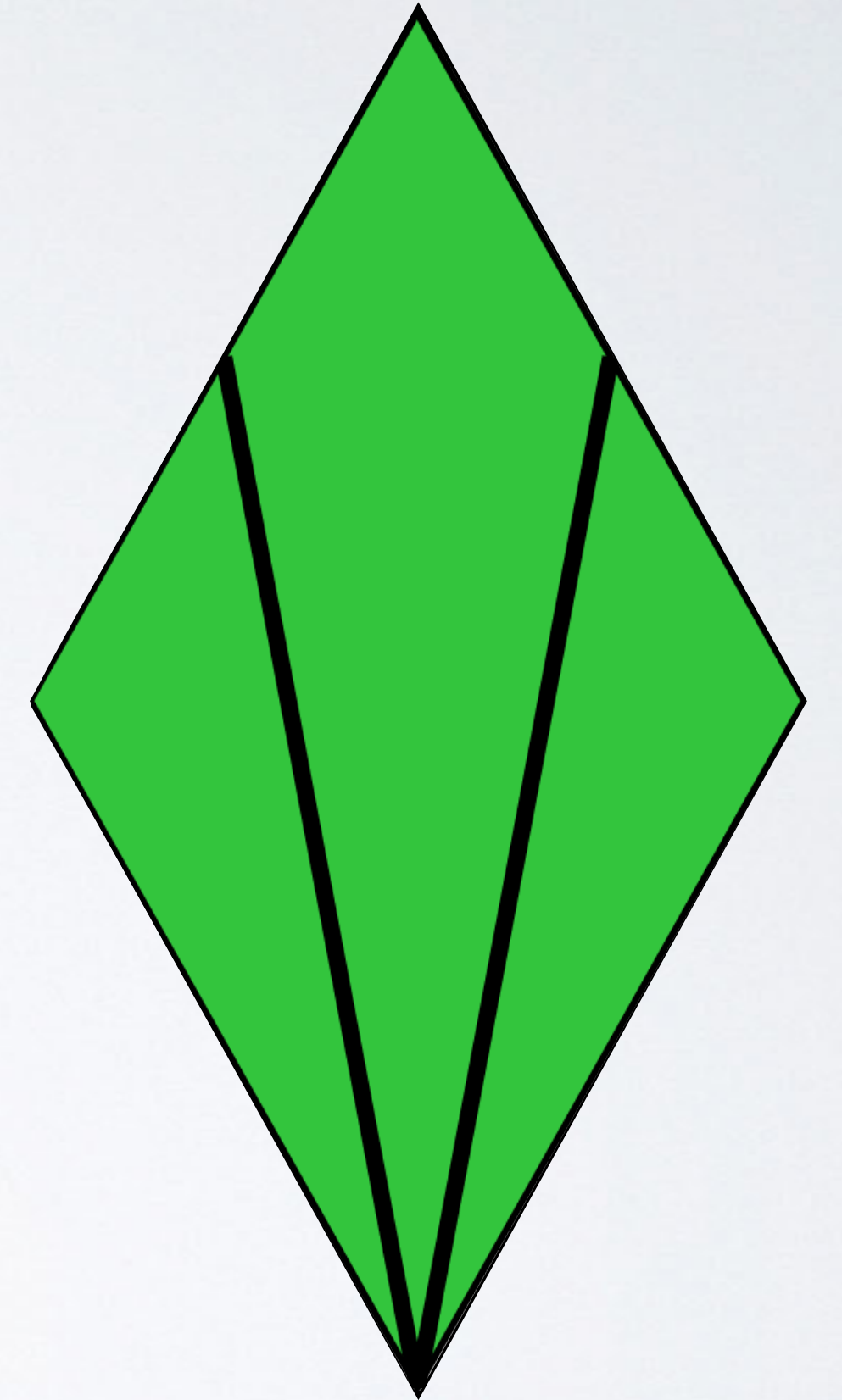
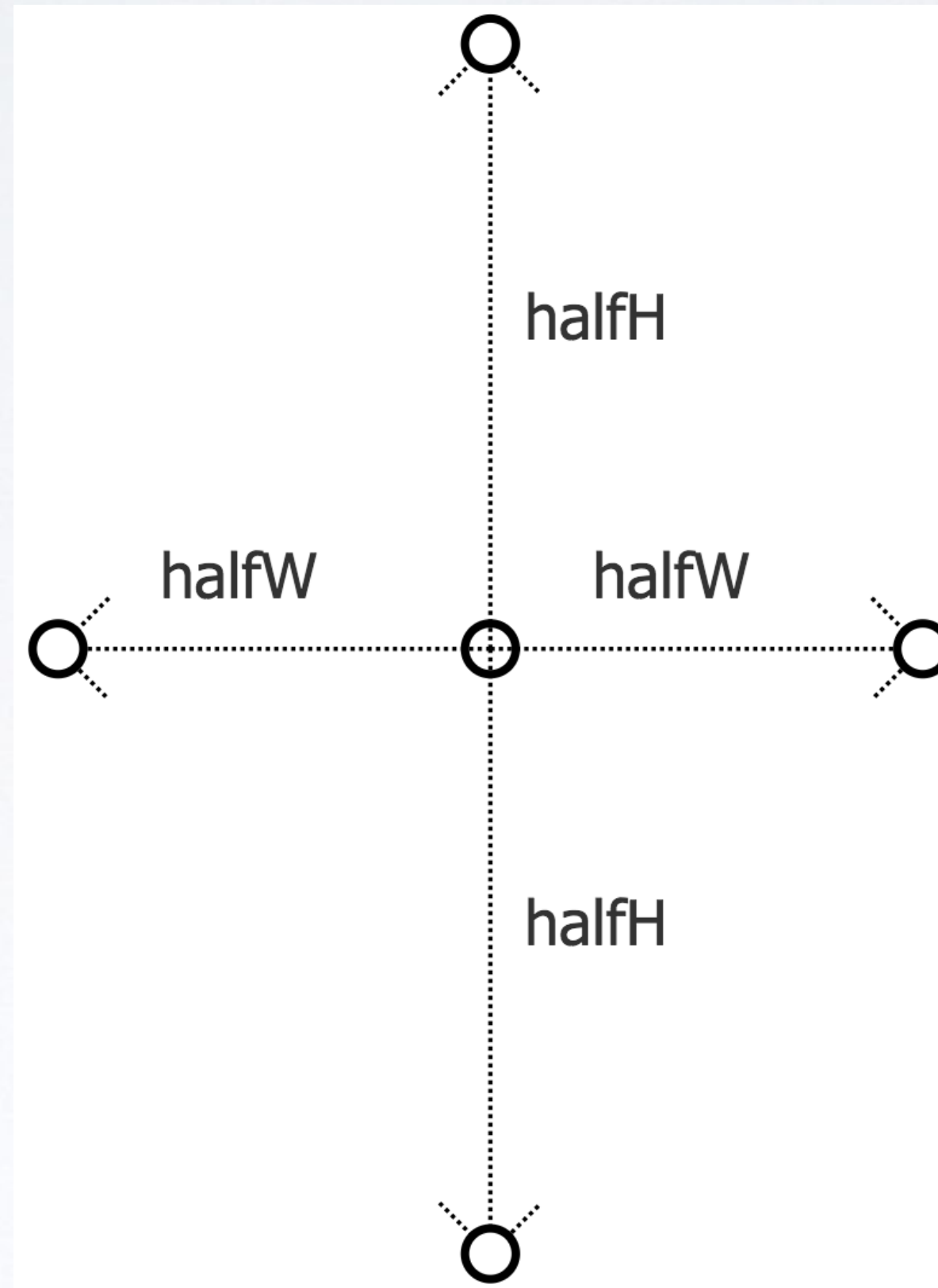
See Paper
Provenance
(but not SVG-specific)

Generic Refactorings

Demo



Rhombus with Veins



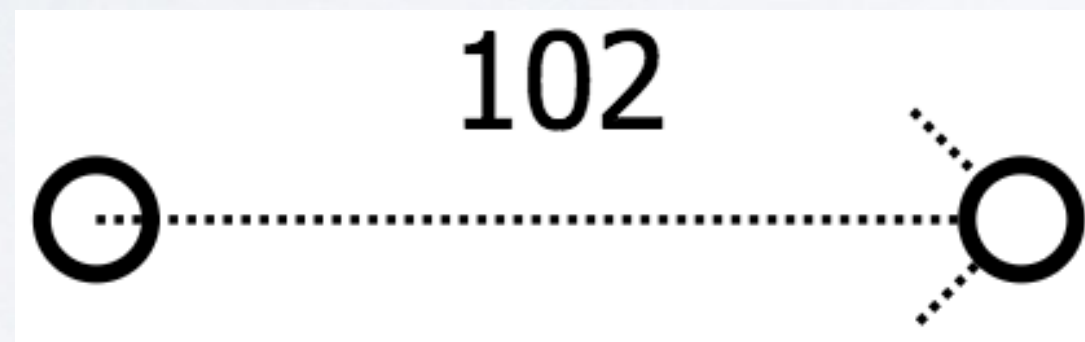
Widgets for Intermediate Values

Points



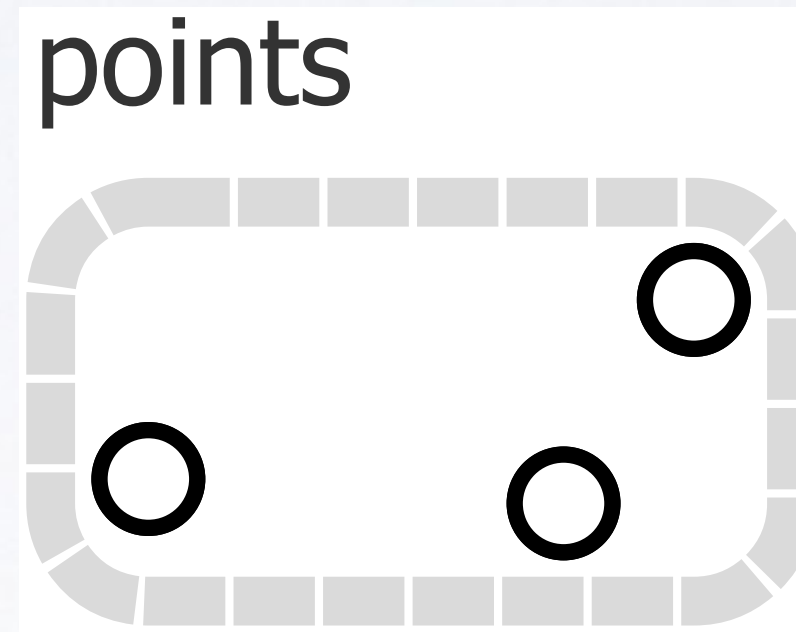
[79, 89]

Offsets



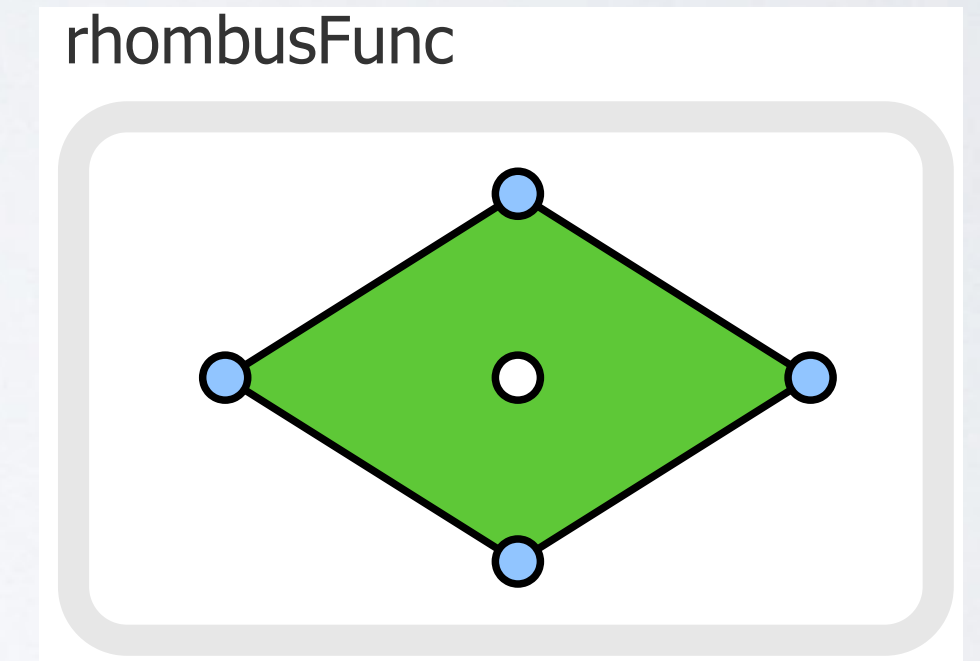
$x + 102$

Lists



[pt1, pt2, pt3]

Calls



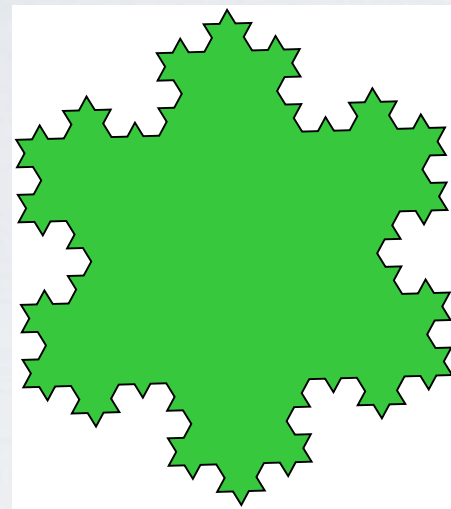
rhombusFunc [79, 89] 49 78

Expression Focusing + **Generic Refactorings**

Big Q

What kinds of programs can be constructed *entirely* through output manipulations?

Examples



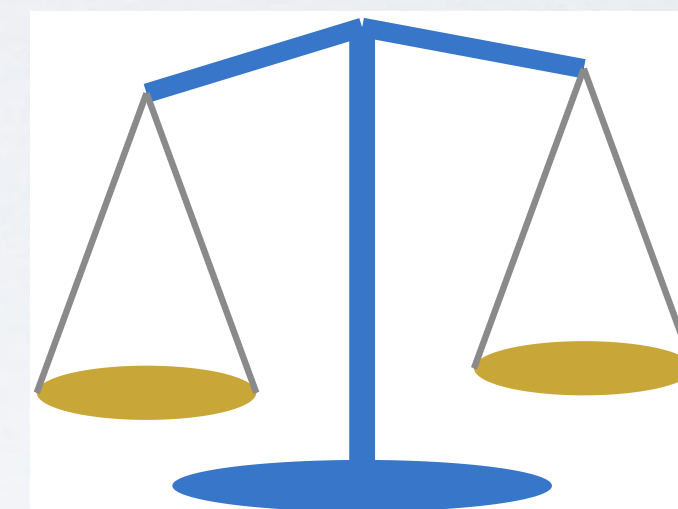
(i) Koch Snowflake



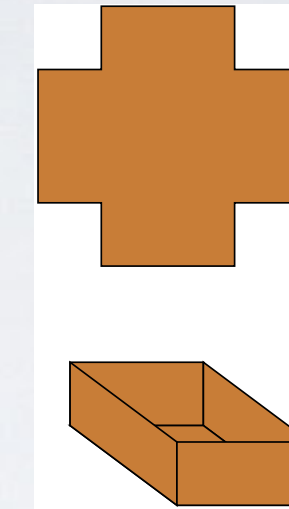
(ii) Precision Floor Plan



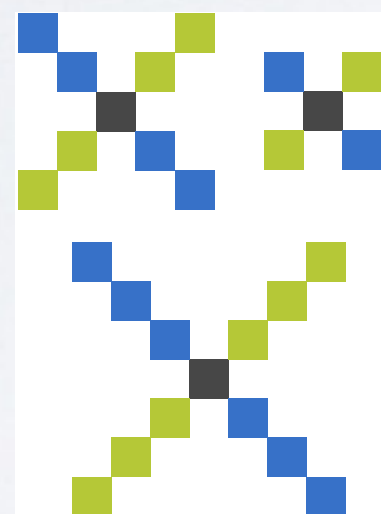
(iii) Mondrian Arch



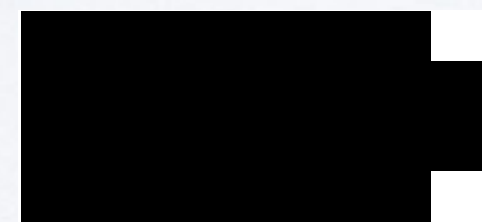
(iv) Balance Scale



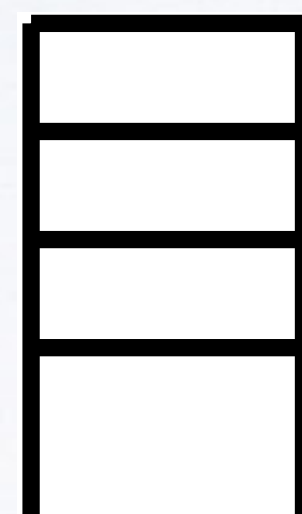
(v) Box Volume



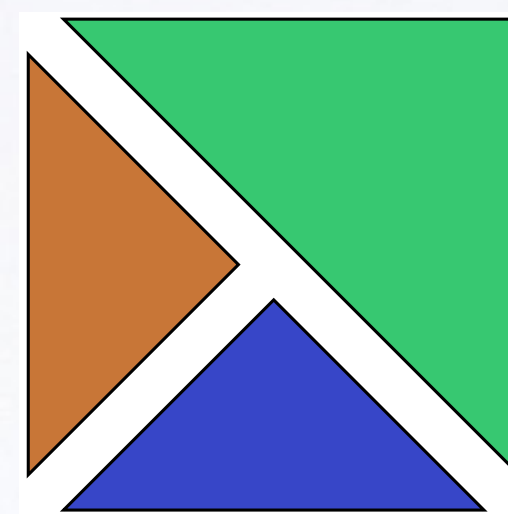
(vi) Xs



(vii) Battery



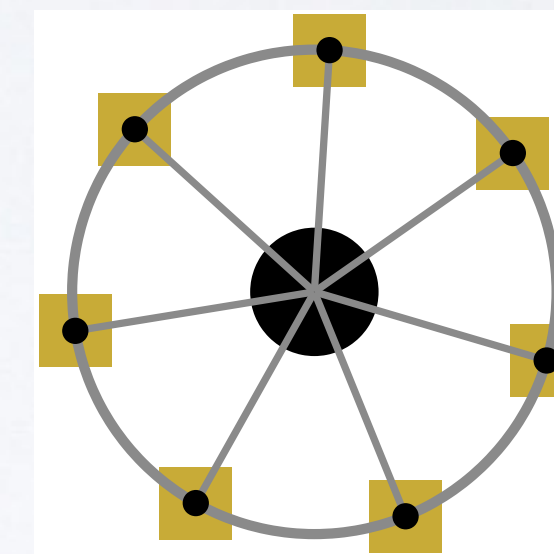
(viii) Ladder



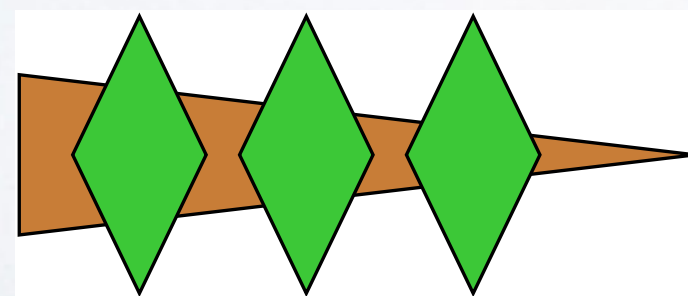
(ix) Logo (via Three Tris)



(x) N Boxes



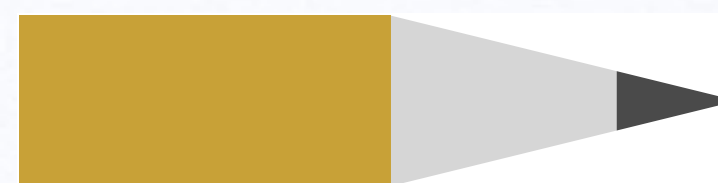
(xi) Ferris Wheel



(xii) Tree Branch



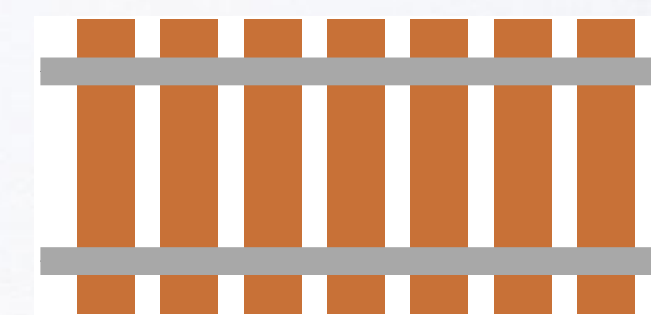
(xiii) Target



(xiv) Pencil Tip

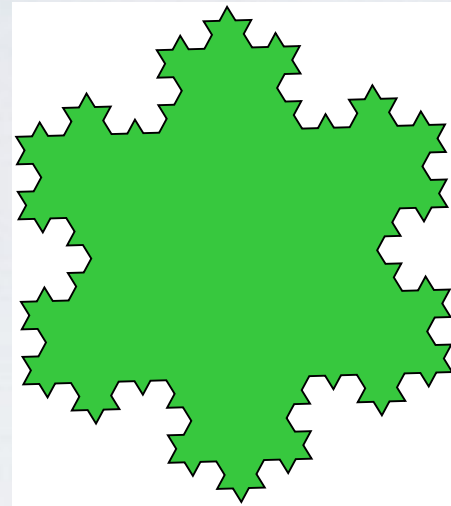


(xv) Arrows



(xvi) Rails

WWID: PBD Benchmarks



(i) Koch Snowflake



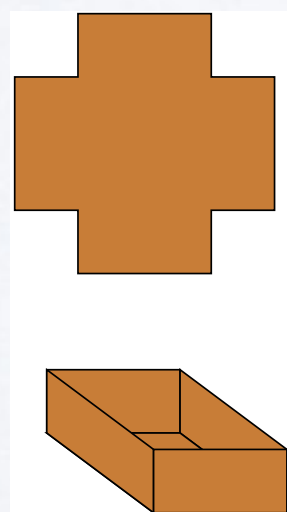
(ii) Precision Floor Plan



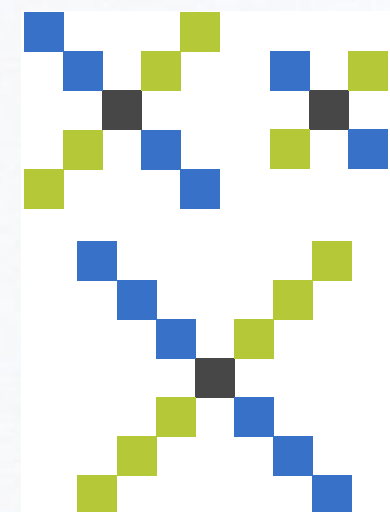
(iii) Mondrian Arch



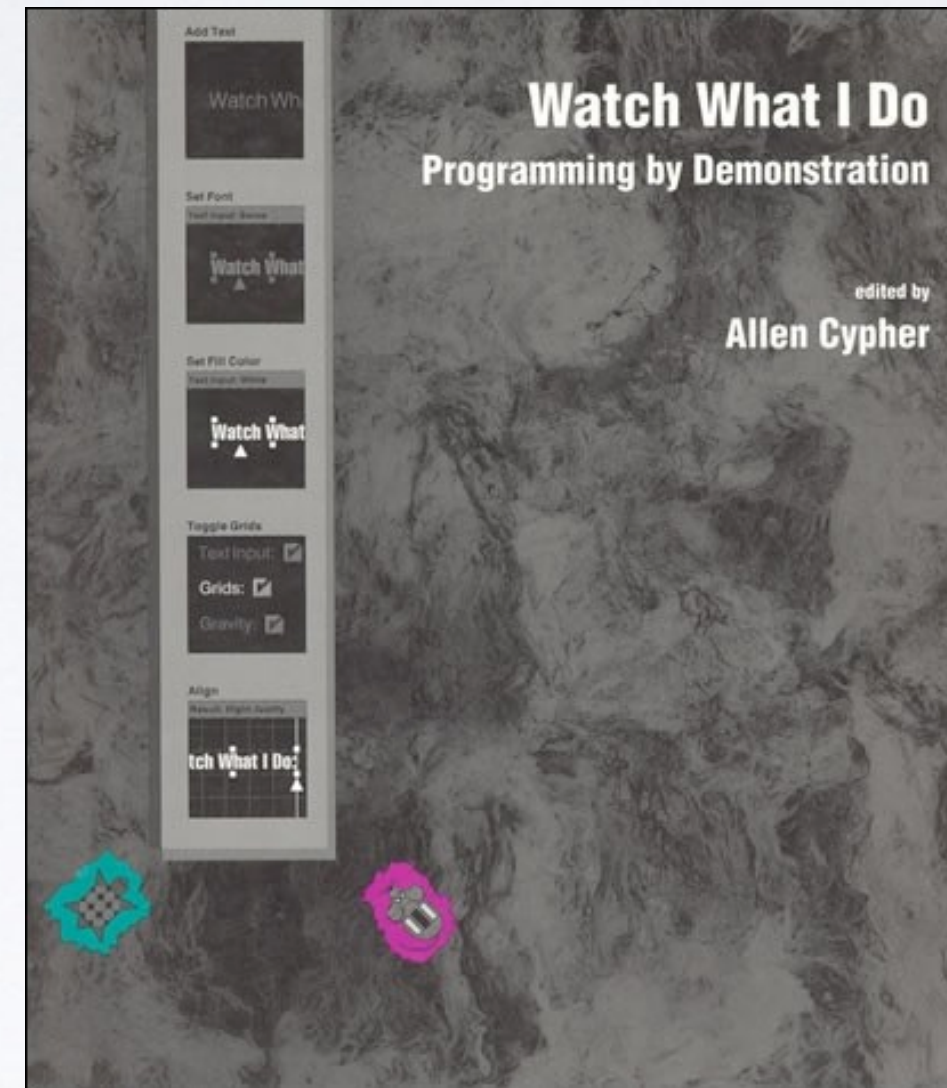
(iv) Balance Scale



(v) Box Volume



(vi) Xs



Watch What I Do:
Programming by Demonstration
Ed. Allen Cypher, 1993

Features needed for 9 remaining tasks:

- Text boxes
- list operations
- intersections of lines with edges
- overlapping & containment constraints
- multiple constraint solving
- arbitrary if-then-else branches

Future Work

Widget Visibility

Soooo many!

Contextual
visibility only
helps a little.

Change Explanation

Multiple results.
Necessary, but 😞

Better change
descriptions?

ODP for Novcies

ODP is
tantalizing.

But we haven't
shown it's easy.

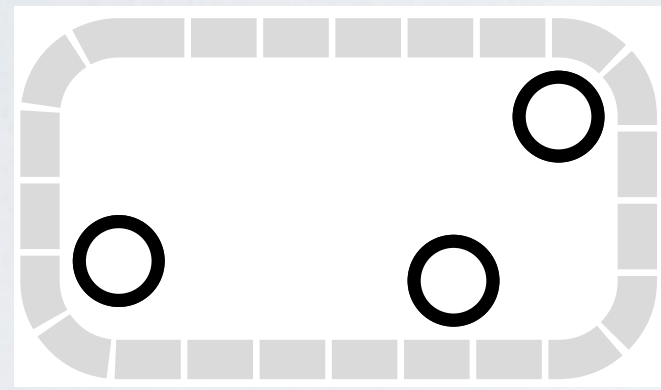
Points



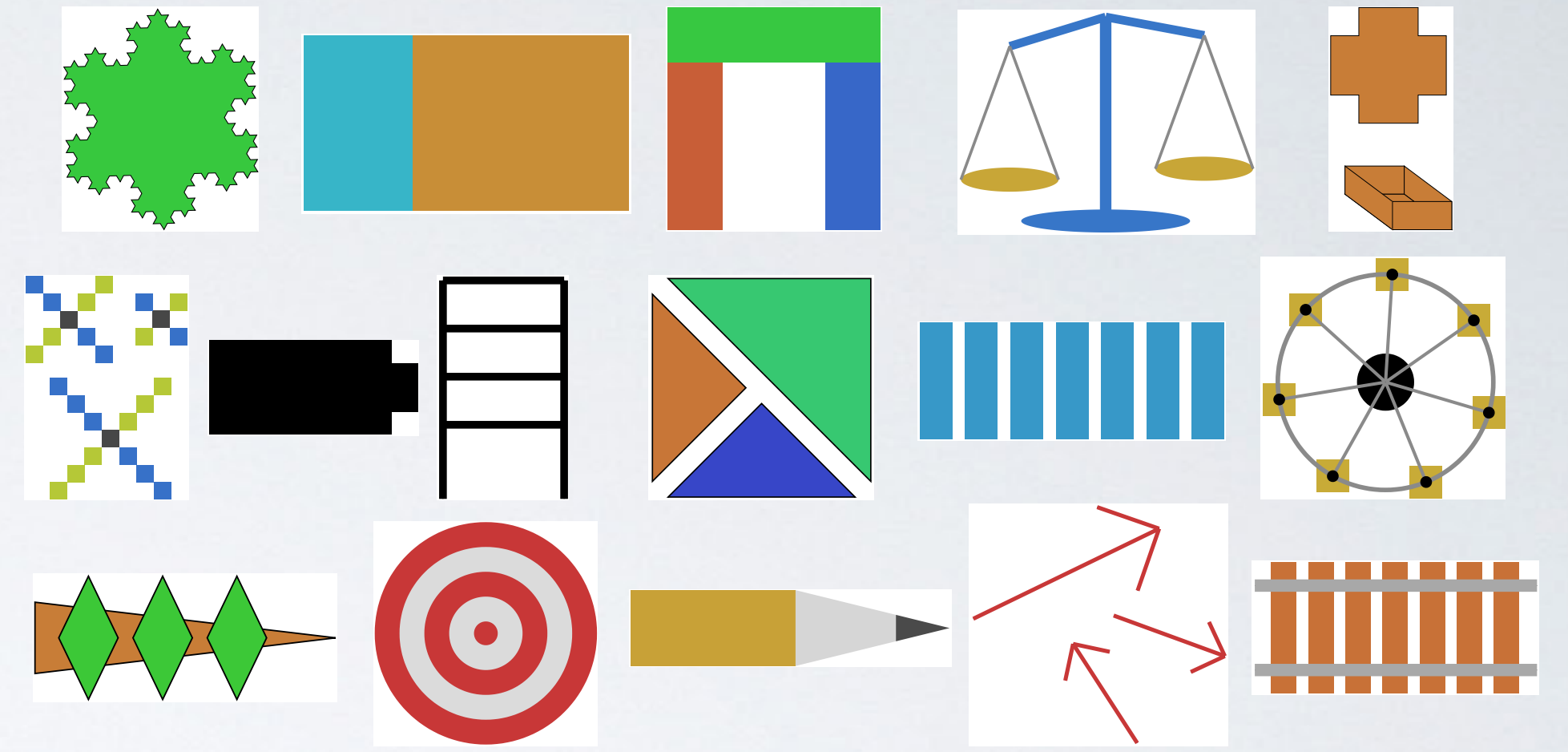
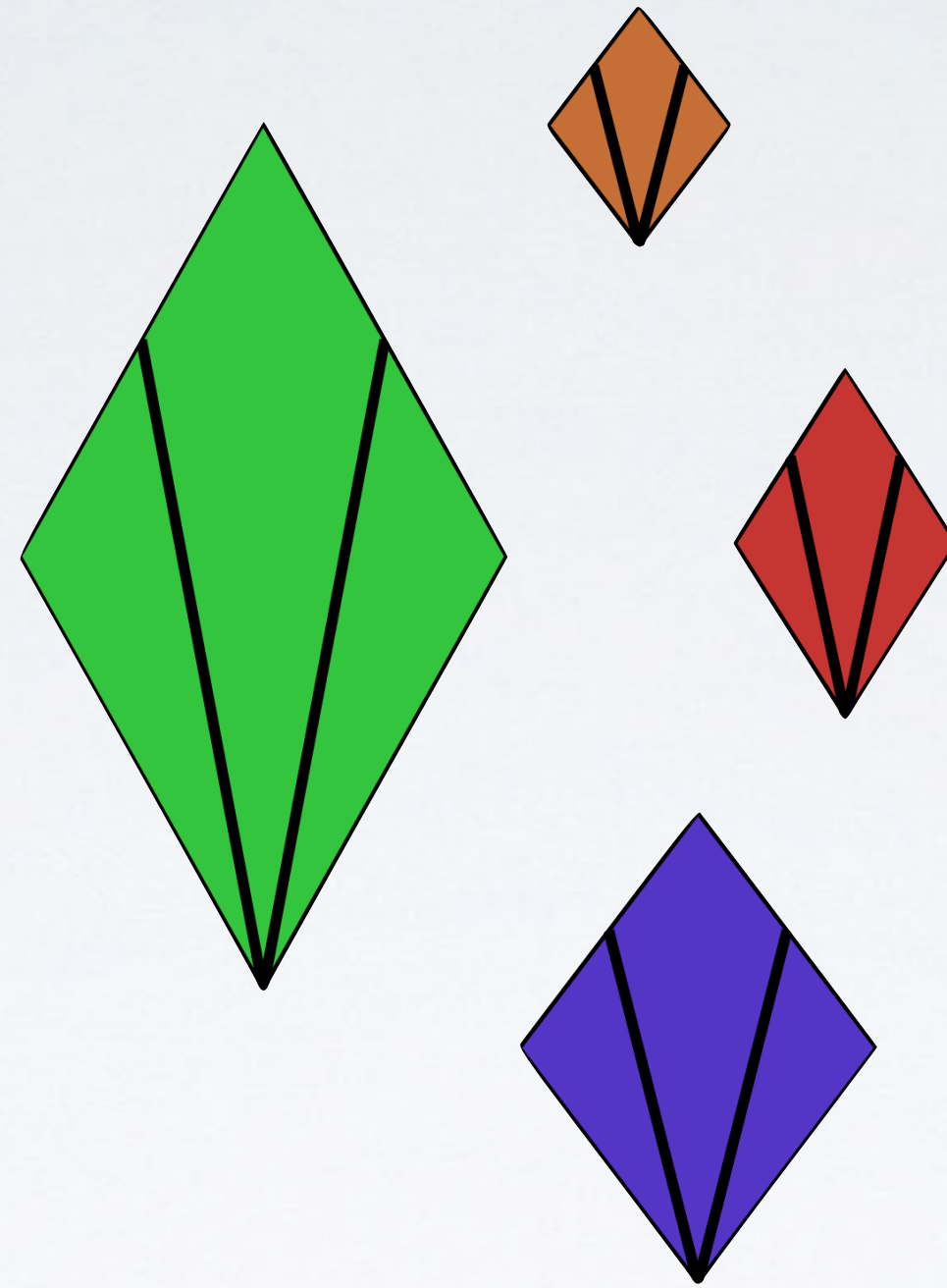
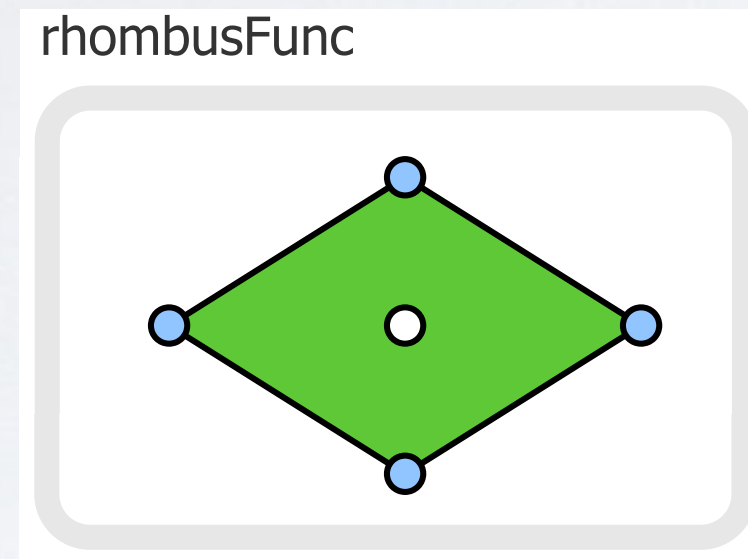
Offsets



Lists



Calls



DM on More Than Output!

Intermediate Value Widgets
Expression Focusing

Generic Tools, Too!

Generic Refactorings
(via generic tracing)

search online for “sketch n sketch”

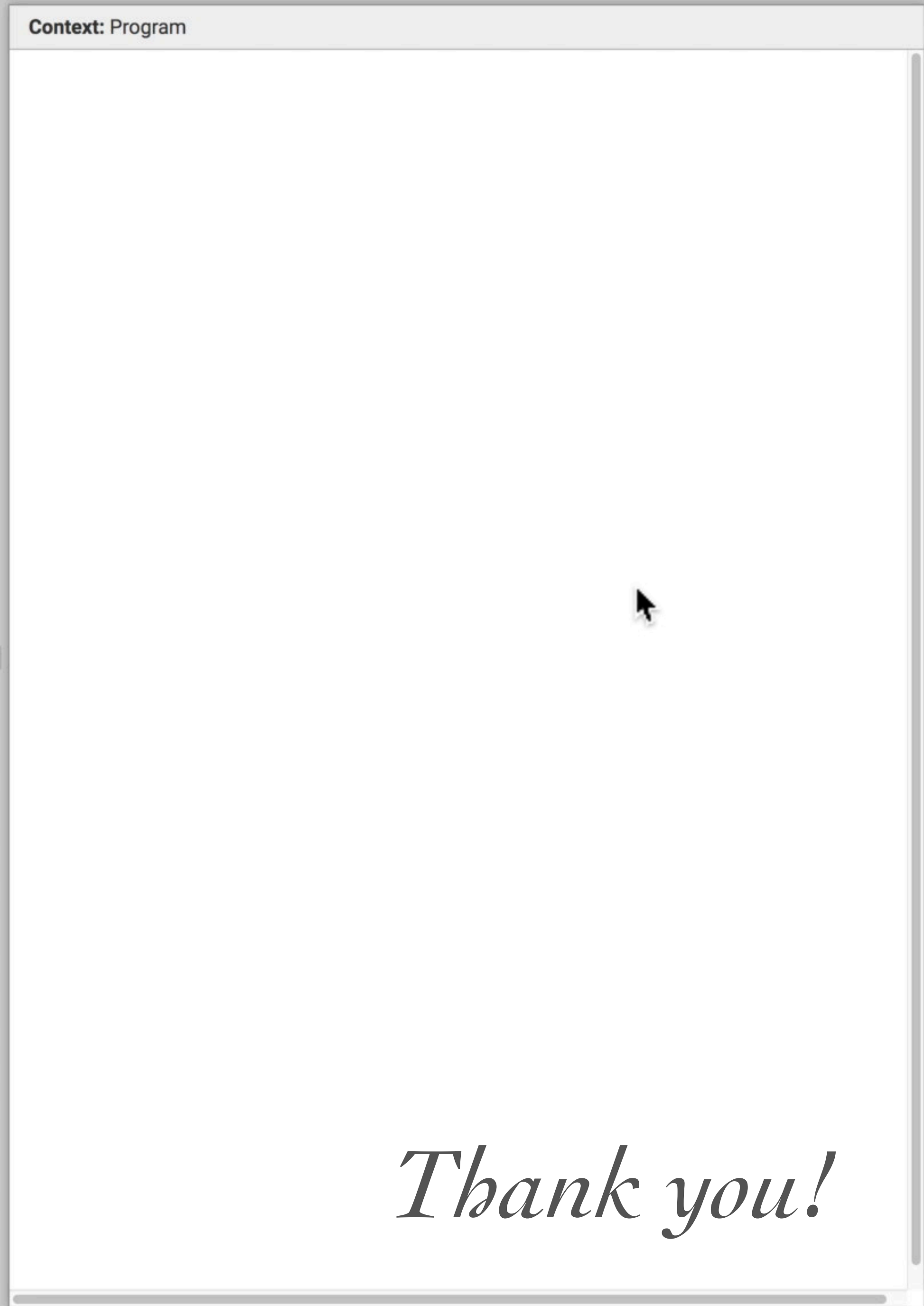
Thank you!

Current file: *Untitled **

Undo Redo Clean Up Run ▶

```
1 svg (concat [  
2 ])
```

Context: Program



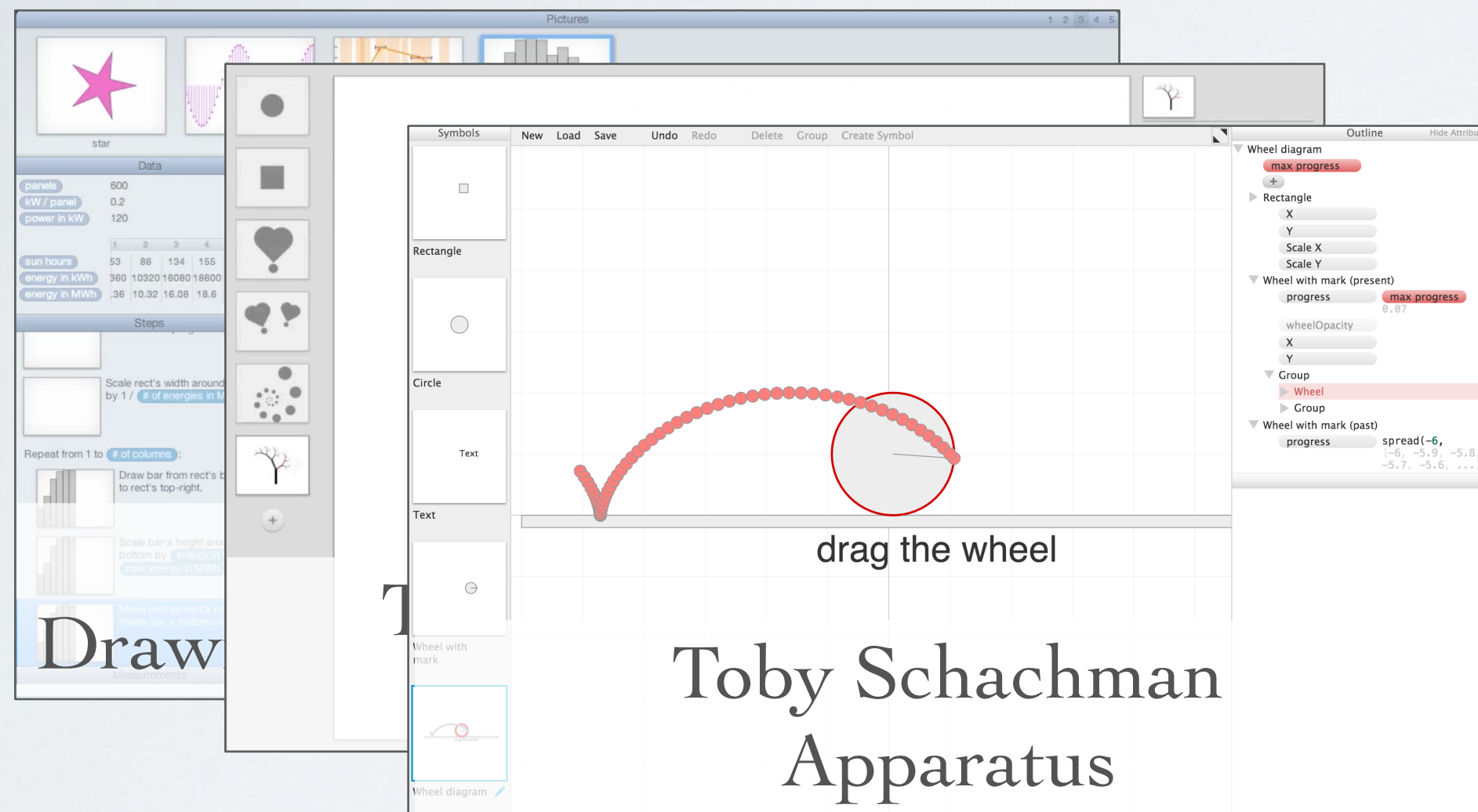
- Built-In Tools**
- Cursor
- Point or Offset
- Polygon
- User-Defined Tools**
- Standard Library Tools**
- vec2DPlus
- vec2DLength
- circle
- ring
- ellipse
- rect
- square
- line
- rectByCenter
- squareByCenter
- nPointsOnCircle
- nPointsOnSegment
- nPointsSepBy
- nHorizontalPointsSepBy
- nVerticalPointsSepBy
- pointsBetweenSepBy
- midpoint
- onLine
- onPerpendicularLine

Thank you!

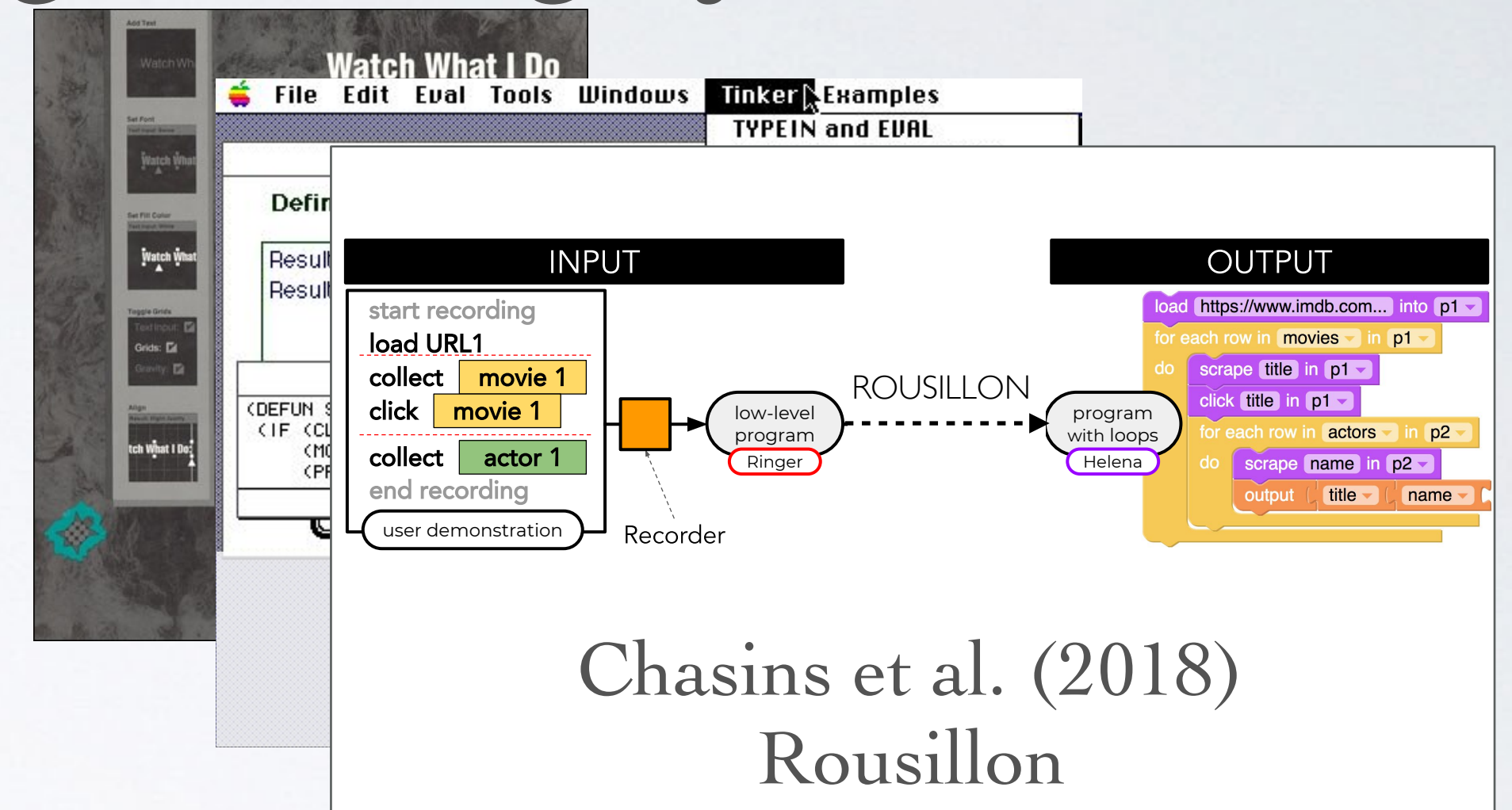


Related Work: Non-standard Programs

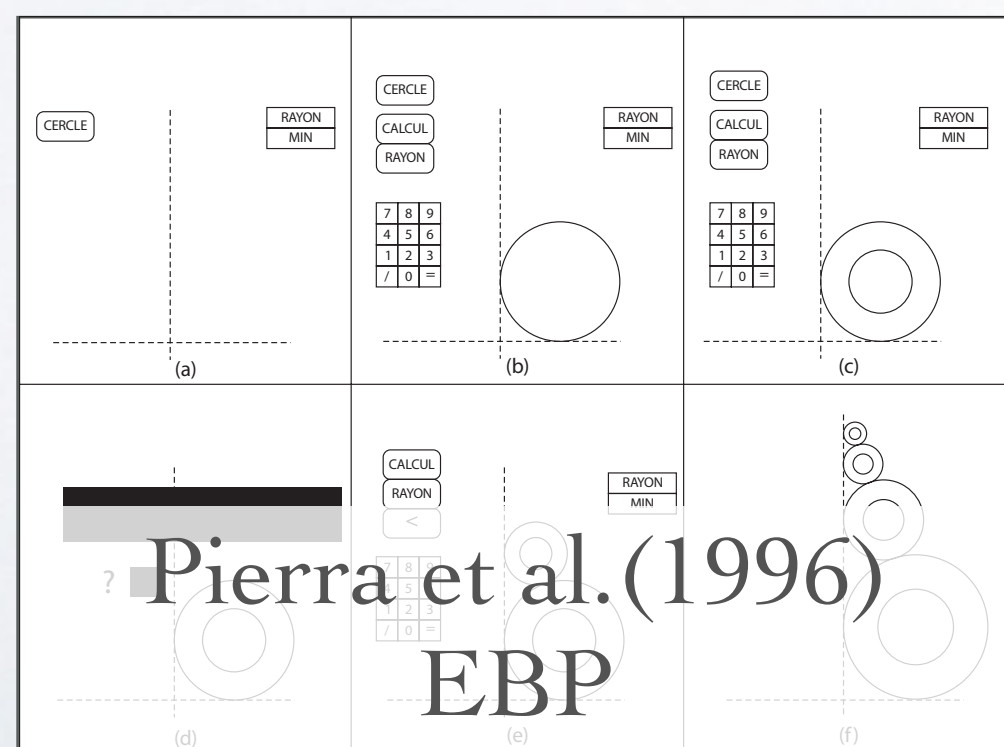
Drawing with Constraints



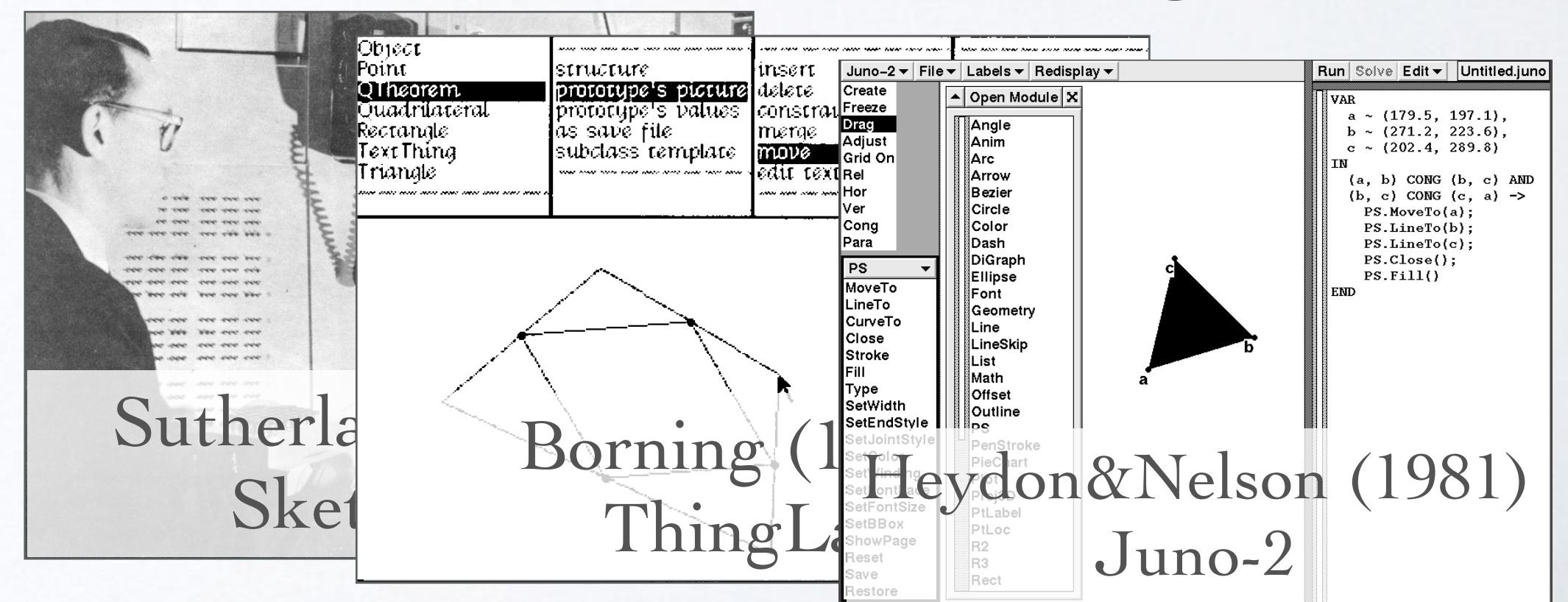
Programming by Demo (PBD)



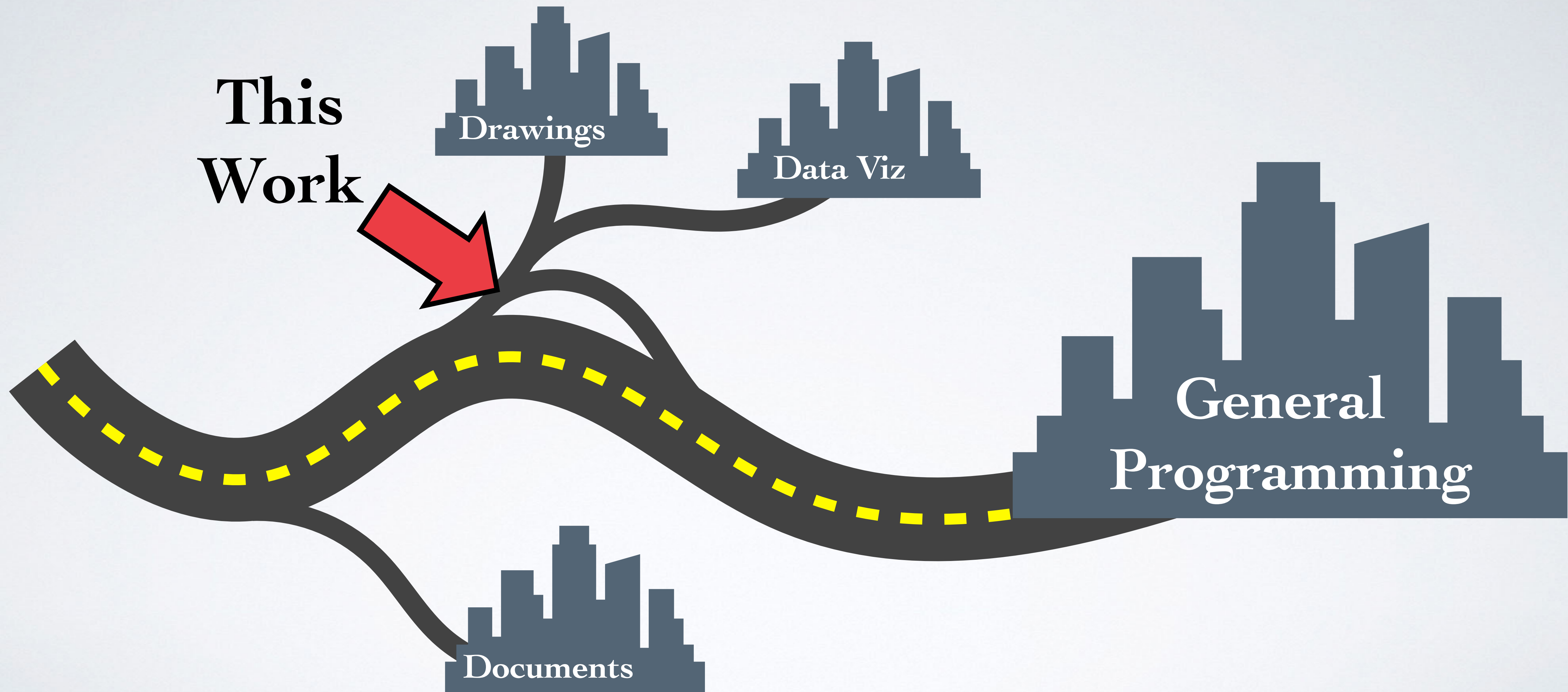
Parametric CAD



Constraint-Oriented Programming



Research Roadmap



DRAW SHAPE

1. Inserts function call, assigns it to a variable.
2. Attempts to add `newVar` and `[newVar]` to the list literals in the program.
3. Succeeds when number of shapes in the output increases by the expected amount.

MAKE EQUAL

1. Use numeric traces (Chugh et al. PLDI '16) to set up an equation:

$$114_{\text{lineX1}} = 245_{\text{rectCX}} - 80_{\text{rectHalfW}}$$

Numeric Traces (Chugh et al. PLDI '16)

let a = 3 in

let b = 5 in

a + b

⇓

8

Numeric Traces (Chugh et al. PLDI '16)

```
let a = 3a in
```

```
let b = 5 in
```

```
  a + b
```

⇓

```
8
```

Numeric Traces (Chugh et al. PLDI '16)

let a = 3_a in

let b = 5_b in

a + b

⇓

8

Numeric Traces (Chugh et al. PLDI '16)

let $a = 3$ in

let $b = 5$ in

$a + b$

↓

8

MAKE EQUAL

1. Use numeric traces (Chugh et al. PLDI '16) to set up an equation:

$$114_{\text{lineX1}} = 245_{\text{rectCX}} - 80_{\text{rectHalfW}}$$

2. Choose a constant to solve for & remove. Solve. (External solver: REDUCE).

$$114_{\text{lineX1}} \rightsquigarrow 245_{\text{cxRect}} - 80_{\text{halfWRect}}$$

$$80_{\text{halfWRect}} \rightsquigarrow 245_{\text{cxRect}} - 114_{\text{lineX1}}$$

$$245_{\text{cxRect}} \rightsquigarrow 114_{\text{lineX1}} + 80_{\text{halfWRect}}$$

3. If a needed constant is not bound to a variable, insert a new `let` binding at a scope visible to its usages.

4. Ranking heuristic:

1. Smallest AST (often all the same size).

2. Shortest distance between constants removed (measured in lines).

3. Prefer removing constants later in the program (less like to cause a dependency inversion).

ABSTRACT

1. Interpret the selection as a late (“proximal”) set of program expressions. (Probably could be looser.)
2. Choose one of those expressions to be the return expression of the function.
3. Iteratively find let bindings that (a) have free variables and (b) are only used in the function body and add those bindings to the function body.
4. Any remaining free variables become arguments.

REPEAT OVER FUNCTION CALL

1. Set up an expression filter: Find $[x, y]$ pair values in provenance (execution history) of selected shapes and thereby identify relevant x expressions, y expressions, and *point* expressions in the program.
2. Interpret the programmer's selections to a single expression that contains either (a) one of the above *point* expressions, or (b) both an x and y expression from above. Use `ABSTRACT` to make this single expression a function over a single point.
3. Map that new function over the point list.

SNAP DRAWING VIA *VALUE HOLES*

1. Internally: Insert template code with *value holes* in place of the snaps.
(A value hole is a temporary expression that contains a value.)

```
[x, y] = [123, 456]
```

```
rect1 = rect ... [??123, ??456] ...
```

2. Examine the provenance of the value in each to fill the hole by either:
 1. Using an existing variable (from the execution environment or from the static scope, possibly moving an existing binding into scope).
 2. Introducing (and using) a new variable for an existing expression.
 3. Deconstructing some variable in the environment with a pattern match to expose a needed value (and using the introduced variable).

```
[_, y] = somePoint
```


DRAW CUSTOM FUNC VIA *ROLES*

1. Functions that take two points, or a point and a distance, are drawable.
2. Types may be tagged with a set of *roles*, explaining the type's semantic meaning. (E.g. "This number is a *width*. This number is a *color*.") Called "brands" in APX. Similar to measure types, but not type-checked.

3. Roles are introduced by type aliases.

```
type alias Color = Num
```

```
rect :: ... → Color → ...
```

4. Roles propagate during the unification step of type inference.

5. Addition domain-specific rules for propagation, e.g.:

$$\mathbf{a}_{\text{Num}:\{X\}} + \mathbf{b}_{\text{Num}:\{\}} \Rightarrow \mathbf{a}_{\text{Num}:\{X\}} + \mathbf{b}_{\text{Num}:\{\text{HorizontalDistance}\}}$$

6. Roles also determine the defaults for arguments.

Provenance



Four Kinds of Provenance

Numeric Traces (Chugh et al. PLDI '16)

Offsets (numbers tagged with other coordinate)

“Based On” Provenance

“Parents” Provenance

“Based On” Provenance

What expressions are associated with a value selected in the output?

For a particular value, what other values at other execution steps were used to produce it?

$$\Gamma \vdash e \Downarrow v^e, \{v_1, \dots, v_n\}$$

Could you hide the code?

Fundamental limitations?

Other Limitations?

Will the techniques generalize?

Future Work

Could you hide the code?

Maybe for simpler cases.

Can you represent the computation visually? (VPLs 😞)

Code only → simulate computer.

Output only → simulate code.

Consider the hover-to-preview interaction today.

(Later APX demos did hide the code)

Fundamental Limitations?

So far: “Select and Act” in small steps.

Good for mouse, because that’s all a mouse can do.

Generally avoided large inference steps: ambiguity.

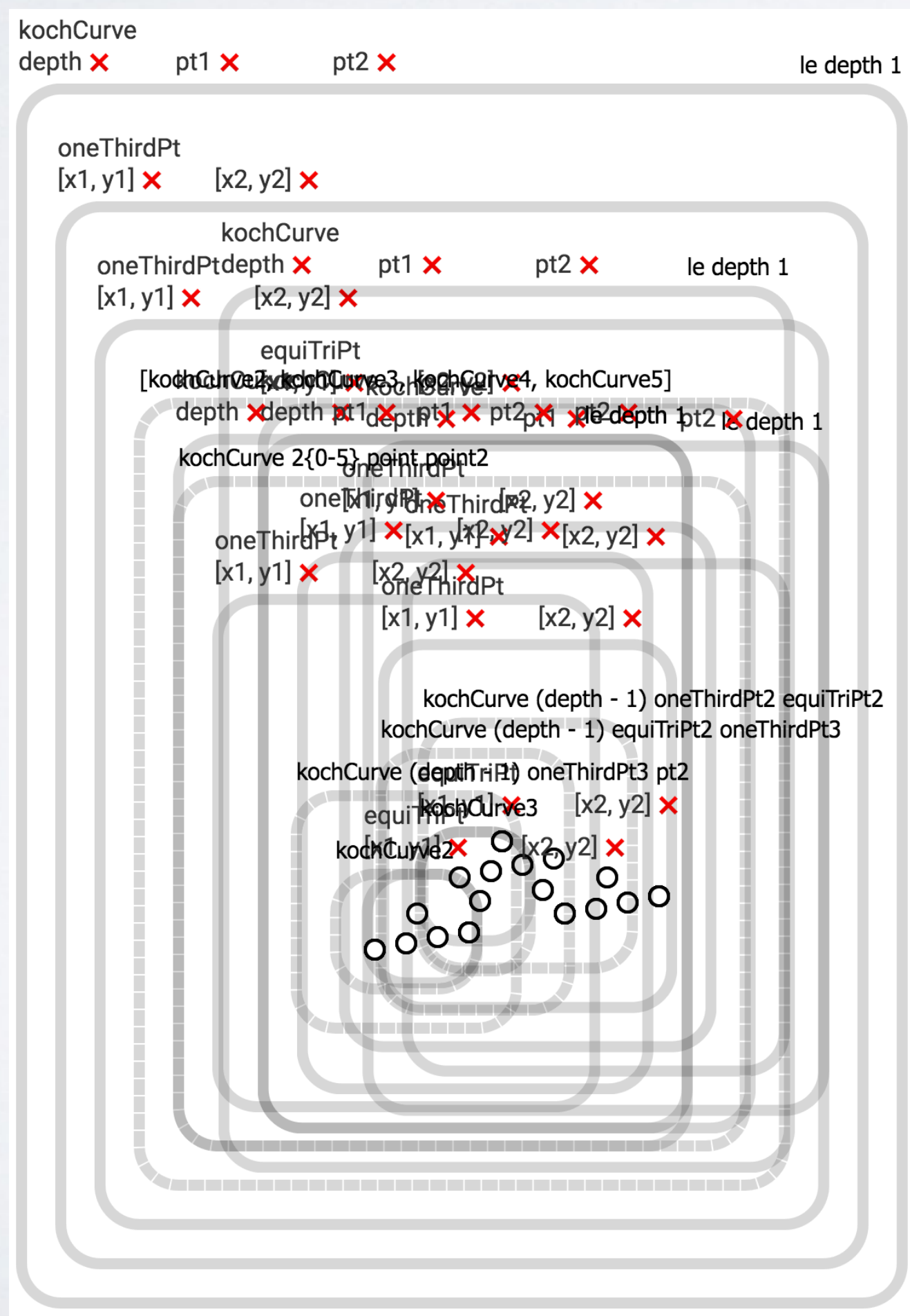
(exceptions: RELATE, REPEAT BY INDEXED MERGE)

$\text{bandwidth}_{\text{keyboard}} > \text{bandwidth}_{\text{mouse}}$

...voice input?

Fundamental Limitations?

Impossible to display all intermediates.



Solution so far: *contextual visibility*.

But this is fundamental:
#intermediates >>>>> screen space

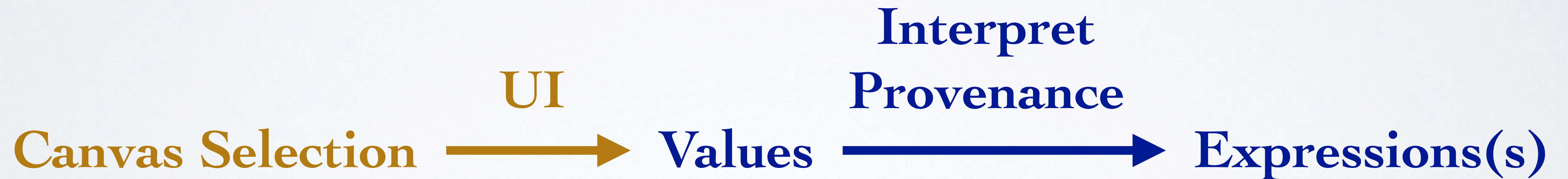
Other Limitations?

Not much work on breaking relationships.
(Edit history?)

More details need to be worked
out so tools compose reliably.
(Syntactic binding locations, e.g. Xs example.)

Will the techniques generalize?

“Select & Act”



Future Work

Transform DSL over value selections

Unified provenance

Visualize non-visual code