Task:

Q: What Would You Choose?

Programming Language
(e.g. JavaScript, Processing)

Direct Manipulation System
(e.g. Illustrator, PowerPoint)
Task:

Q: What Would You Choose?

Programming Language (e.g. JavaScript, Processing)  Direct Manipulation System (e.g. Illustrator, PowerPoint)
Imagine Using a **Direct Manipulation** System…
Imagine Using a Direct Manipulation System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System...
Imagine Using a **Direct Manipulation** System...
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Direct Manipulation** System…
Imagine Using a **Programming** System...

circle1 = ...

"Draw"
Imagine Using a **Programming** System...

circle1 = ...
line1 = ...
line2 = ...

"Draw"
Imagine Using a **Programming** System...

circle1 = ...

pt1 = left(circle1)

line1 = ...

line2 = ...
Imagine Using a **Programming** System...

circle1 = ...

pt1 = left(circle1)
pt2 = right(circle1)
line1 = ...

line2 = ...

"Relate"
Imagine Using a **Programming** System…

circle1 = ...
pt1 = left(circle1)
pt2 = right(circle1)
line1 = ...
pt3 = top(circle1)
pt4 = bottom(circle1)
line2 = ...

"Relate"
Imagine Using a **Programming** System…

circle1 = …

pt1 = left(circle1)
pt2 = right(circle1)
line1 = …

pt3 = top(circle1)
pt4 = bottom(circle1)
line2 = …

rectArray = …
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
) {
  ...
}
ferrisWheel(4, ...);
Imagine Using a **Programming** System…

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
)
{
  ...
}
ferrisWheel(4, ...);
```

"Tweak"
Imagine Using a **Programming** System...

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
)
{
  ...
}

ferrisWheel(8, ...);
```

"Tweak"
Imagine Using a **Programming** System...

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
)
{
  ...
}

ferrisWheel(8, ...);
```

"Tweak"
Imagine Using a **Programming** System…

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
)
{
  ...
}

ferrisWheel(8, ...);
```

"Tweak"
Imagine Using a **Programming** System…

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
) {
  ...
}

ferrisWheel(8, ...);
```

"Tweak"
Imagine Using a **Programming** System…

```javascript
function ferrisWheel(
  numSpokes,
  spokeLen,
  rotAngle,
  sizeCar,
  radiusCenter,
  cx,
  cy
) {
  ...
}
ferrisWheel(16, ...);
```

"Tweak"
Imagine Using a **Programming** System...

```javascript
function ferrisWheel ( numSpokes, spokeLen, rotAngle, sizeCar, radiusCenter, cx, cy ){
  ...
}
ferrisWheel(16, ...);
```
<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Direct Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td><img src="image" alt="Ai" /></td>
</tr>
<tr>
<td>Presentations</td>
<td><img src="image" alt="P" /></td>
</tr>
<tr>
<td>Documents</td>
<td><img src="image" alt="W" /></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td><img src="image" alt="X" /></td>
</tr>
<tr>
<td>Web Apps</td>
<td><img src="image" alt="d" /></td>
</tr>
</tbody>
</table>
General-Purpose Languages

Domain-Specific Languages or Libraries

Domain-Specific GUI Tools

<table>
<thead>
<tr>
<th>2D Graphics</th>
<th>JS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentations</td>
<td>JS</td>
<td></td>
</tr>
<tr>
<td>Documents</td>
<td>JS</td>
<td></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>JS</td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td>JS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming Language</td>
<td>Direct Manipulation</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>2D Graphics</td>
<td>JS</td>
<td><img src="image" alt="Sketch-n-Sketch" /></td>
</tr>
<tr>
<td>Presentations</td>
<td>JS</td>
<td>![???]</td>
</tr>
<tr>
<td>Documents</td>
<td>JS</td>
<td>![???]</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>JS</td>
<td>![???]</td>
</tr>
<tr>
<td>Web Apps</td>
<td>JS</td>
<td>![???]</td>
</tr>
</tbody>
</table>
Our Approach

$\lambda$-calculus + Direct Manipulation
Sketch-n-Sketch Demo

```
sketch-n-sketch v0.5.2

1 (def spokes (\(cx cy n r rot\)
2   (let [points (nPointsOnCircle n rot cx cy r)
3       (map \([x y]\) (line "lightgrey" 5 cx x y) points))]
4   (b! (obj spokes 183 159 10 100 0))
5 )

```

Run Code  Revert  Undo  Redo  Clean Up

Cursor
Line  Text
Rect  BB
Ellipse  BB
Polygon  BB
Path  BB
A  bounces.

Dig  A = B
Dupe  Merge
Group  Add
Repeat Flight
Repeat To
Repeat Around


Sketch-n-Sketch Demo
Sketch-n-Sketch Demo

[Image of Sketch-n-Sketch GUI]

Sketch-n-Sketch v0.5.2

```
(defun spokes (cx cy n r rot)
  (let points
    (PointsOnCircle n rot cx cy r)
    (map
     (lambda ([x y]) (line 'lightgray 5 cx x y)) points)))

(defun newGroup4 ((rect1.w rect1.stroke rect1.strokeWidth fill)
  (def rect1
    (rawRect fill rect1.stroke rect1.strokeWidth xAnchor yAnchor))

(defun line2
  (let [[x2 y2] [x1 y1] x2 y2]
    (Line rect1 w rect1.stroke rect1.strokeWidth x1 y1 x2 y2))

(defun line3
  (let [[x1 x2 y2] [x2 y2] x1 y1 x2 y2]
    (Line rect1 w rect1.stroke rect1.strokeWidth x1 y1 x2 y2))

(defun newGroup7
  (let circle6.cx 259
    (let circle6.cy 265
      (def radius 176))

(defun circle6
  (let r 28
    (let color 469
      (rawCircle color 360 0 circle6.cx circle6.cy r)))

(let spokes_k4768 12{1-20}
  (let spokes_k4766 -0.052640000000000019{3.14-3.14}
    (spokes circle6.cx circle6.cy spokes_k4768 radius spokes_k4766)))
```

[Code Editor]
Semi-Automated Programming by Manipulating Output
Draw  Relate  Group  Tweak

e ⇓ v
Constraint on Original $e$: 

\[ v = t \]

Constraint on Updated $e'$: 

\[ v' = t \]
Constraint on Original $e$:

$\mathbf{e} \Downarrow_{t} \mathbf{v}$

$\mathbf{v} = t$

Constraint on Updated $e'$:

$\mathbf{e}' \Downarrow_{t} \mathbf{v}'$

$\mathbf{v}' = t$
Dataflow-Only Traces

\[ t ::= x \]
\[ t ::= t_1 + t_2 \]
\[ t ::= t_1 \times t_2 \]
\[ t ::= \sin t \]
\[ t ::= \text{pow} \ t_1 \ t_2 \]
\[ ... \]
let a = 1
b = 2

in ...

a+b

a+b*b

(\x.x) a

true ? a : b
Tweak

\[ 110 = x_0 + 2 \times \text{sep} \]
\[ \uparrow \]
\[ 155 = x_0 + 2 \times \text{sep} \]

Change \(x_0\) or \(\text{sep}\)? Or both? Or more?

Change exactly one constant.
Heuristics to automatically choose.
Annotations to freeze (\(n!\)) or thaw (\(n?\)).
Can solve for $x$ if $x$ only occurs once.

- $5\sin(x) = n$ ✓
- $x(y+y^2) = n$ ✓
- $x^2 = n$ ✓
Can solve for $x$ if $x$ only occurs once.

- $5 \sin(x) = n \quad \checkmark$
- $x(y + y^2) = n \quad \checkmark$
- $x(x + y^2) = n \quad \times$
- $x^2 = n \quad \checkmark$
- $x^x = n \quad \times$

Good enough for 80% of equations.
"Raw" Shape

(x1, y1)

(x3, y3)

(x2, y2)
"Anchored" Shape

(x1, y1)

(x1+2, y1+8)

(x1+10, y1+10)
Draw
Relate
Group
Tweak

\[(x_1, y_1)\]

\[(x_1 + 0.20(x_2 - x_1), y_1 + 0.80(y_2 - y_1))\]

\[(x_2, y_2)\]

"Bounded"
Shape
Representation affects grouping operations.

(def rawPoly
  (let [x1 y1] ... 
    (let [x2 y2] ... 
      (let [x3 y3] ... 
        ... )))))

(def anchoredPoly
  (let [x1 y1] ... 
    (let [dx2 dy2] ... 
      (let [dx3 dy3] ... 
        ... )))))

(def boundedPoly
  (let [x1 y1] ... 
    (let [x2 y2] ... 
      (let [dx3 dy3] ... 
        (let [px3 py3] ... 
          ... )))))
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group

Draw
Relate
Group
Tweak
"Anchored" Group

"Bounded" Group
(def rect1
  (let x 52 ... ))

(def line2
  (let x1 50 ... ))

(def line3
  ... )
(def rect1 (let x 52 ...) ))

(def line2 (let x1 50 ...) ))

(def line3 ...) 

Features in Canvas
New Hole in Code
Hole with Relationship
New Code
(def rect1
  (let x 52 ... ))

(def line2
  (let x1 50 ... ))

(def line3 ...
  ...)
(def rect1
  (let x 52 ... ))

(def line2
  (let x1 50 ... ))

(def line3
  ... )
(def rect1_x 52)
(def line2_x1 50)
(def [rect1_x' line2_x1'] [rect1_x line2_x1])

(def rect1
  (let x rect1_x' ... ))
(def line2
  (let x1 line2_x1' ... ))
(def line3 ...

LIFT Constants and DIG New Hole
FILL Hole w/ Desired Relationship
(def rect1_x 52)
(def line2_x1 50)
(def [rect1_x' line2_x1'] [rect1_x rect1_x])

(def rect1
  (let x rect1_x' ... ))

(def line2
  (let x1 line2_x1' ... ))

(def line3
  ... )
(def rect1_x 52)

(def rect1
  (let x rect1_x ...
))

(def line2
  (let x1 rect1_x ...
))

(def line3
  ...
)
(def rect1_x 52)

(def rect1
 (let x rect1_x ... ))

(def line2
 (let x1 rect1_x ... ))

(def line3 ...
  ... )
Which Constants to Abstract?

**Heuristic: Unfrozen and Named**

```
(def foo
  (let [a b] [1 2!]
    [a b 3]))
foo
```
Which Constants to Abstract?

Heuristic: Unfrozen and Named
Programming in 2015

Sketch-n-Sketch

Less Keyboard.
More Mouse.
Draw  Relate  Group  Tweak
Draw
Relate
Group
Tweak

Sketch-n-Sketch

\[ e_1 \quad \Downarrow \quad e_2 \quad \Downarrow \quad e_3 \quad \Downarrow \]

\[ \text{Orange triangles} \]
Less Reliance on Syntactic Structure
; Top-LevelDefs
(def rect1 ...)
(def line2 ...)
(def line3 ...)

; MainExpression
[ rect1 ... ]
Sketch-n-Sketch

Less Reliance on Syntactic Structure

(def group1
  (def rect1 ...)
  (def line2 ...)
  (def line3 ...)
  [ rect1 ... ]
) group1
(def polygon7_bot (+ (+ (* 0.5!...)
(def k3105 // (- (+ (- polygon6...)
(def polygon7_top (- (* 0.5! (+...
(def [polygon5_right k3038] [(-...
(def k3061 // (- (+ polygon5_r1...
(def polygon6_bot (- (+ (- poly...
(def k3063 // (- (+ polygon6_bot...
(def polygon5_top (- polygon6_t...
(def k3103 // (- (+ (- polygon5...
(def [k3041 polygon5_bot] [(- p...
(def k3134 // (- (+ k3041 helpe...
(def k3141 // (- (+ k3038 helpe...
Draw

Relate

Group

Tweak

Sketch-n-Sketch

Sketch-n-Sketch ++
Draw

Relate

Group

Tweak

Sketch-n-Sketch

Sketch-n-Sketch ++

Smarter Algebraic Constraint Solver
Draw

Relate

Group

Tweak

Sketch-n-Sketch

Sketch-n-Sketch ++

Additional Interaction to Resolve User Intent
Draw | Relate | Group | Tweak

Sketch-n-Sketch | Sketch-n-Sketch ++

Limited, Syntactic Program Updates | Expressive, General Program Synthesis
Draw  Relate  Group  Tweak
Draw  Relate  Group  Tweak

Idea

Prototype; Repair; Refactor; Repeat
## Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th></th>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>General Purpose PL + General Program Synthesis</td>
<td>Domain Specific UI + Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Presentations</td>
<td></td>
<td>Sketch-n-Sketch</td>
</tr>
<tr>
<td>Documents</td>
<td></td>
<td>???</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
<td>???</td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
<td>???</td>
</tr>
</tbody>
</table>

85
Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>Domain Specific UI +</td>
</tr>
<tr>
<td>Presentations</td>
<td>Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Documents</td>
<td>Sketch-n-Sketch</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
</tr>
</tbody>
</table>
λ-Calculus + Direct Manipulation

Reuse effort across domains.

Enable experts and library writers to extend built-ins.

Smooth spectrum between "Experts" and "End Users".
\(\lambda\)-Calculus + Direct Manipulation of Output
\( \lambda \)-Calculus + Direct Manipulation of Code
λ-Calculus + Direct Manipulation of Code

Text editing is great, but...

Many mundane, low-level edits.

Especially with generated code.
Idea: Selectable AST Nodes

(defun foo
  (let [a b] [1 2!]
    [a b ?]))

foo

(defun foo (\a
  (let [b] [2!]
    [a b 3]))
  (foo 1)
Idea: Selectable AST Nodes

(def foo (let [a b] [1 2!] [a b ?]))

foo

( def foo ( \a c ( let [b] [2!] [a b c])))

(foo 1 3)
Idea: Selectable AST Nodes

(def rect1
  (let x 52 ...
  ))

(def line2
  (let x1 50 ...
  ))

(def line3
  ...
)
Idea: Selectable AST Nodes

(def rect1
  (let x 52 ... ))

(def line2
  (let x1 52 ... ))

(def line3
  ... )
Idea: Selectable AST Nodes

(def rect1
  (let x 52 ... ))
(def line2
  (let x1 50 ... ))
(def line3 ...
)

(def rect1
  (let x rect1_x ...
))
(def line2
  (let x1 rect1_x ...
))
(def line3 ...
)

A = B
Idea: Draggable AST Nodes

(defun rect1_x 52)
(funcall rect1
  (let x rect1_x ... ))
(funcall line2
  (let x1 rect1_x ... ))
(funcall line3
  )
Idea: Draggable AST Nodes

(def rect1_x 52)

(def line3 ...
  )

(def rect1 (let x rect1_x ...
  ))

(def line2 (let x1 rect1_x ...
  ))
Idea: Draggable AST Nodes

(def rect1_x 52)
(def line3
  (let x1 200 ... )
(def rect1
  (let x rect1_x ... ))
(def line2
  (let x1 rect1_x ... ))
Idea: Draggable AST Nodes

Copy-Paste,
Scope Manipulation,
Name Generation,
Substitution,
Formatting, ...
Idea: Auto Reformatting

(def [x1 y1] [0 0])
(def [x2 y2] [(+ x1 10) (+ y1 10)])
(def [x3 y3] [(+ x1 2) (+ y1 8)])
Idea: Auto Reformatting

Whitespace, Line Breaks, Single vs. MultiDefs, Alignment, Style Conventions, …

(def [x1 y1] [0 0])
(def x2 (+ x1 10))
(def y2 (+ y1 10))
(def x3 (+ x1 2))
(def y3 (+ y1 8))
Programming with

Less Keyboard.

Programming with

More Mouse.
\( \lambda \)-Calculus + Direct Manipulation of Code and Output
Related Work

DM + Code
Dynamic Drawing Apparatus

Code + DM
Wang et al. [FSE 2012]
McDirmid's Demos
Sketch-n-Sketch

Text + Structure Editors
Barista
Code Bubbles
Sketch-n-Sketch++

Constraints
Sketchpad
ThingLab
Juno-2

Program Synthesis
(Sketching, PBE)

Program Repair
http://ravichugh.github.io/sketch-n-sketch/
$\lambda$-Calculus + Direct Manipulation of Code and Output
EXTRA SLIDES
<table>
<thead>
<tr>
<th></th>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>General Purpose PL + General Program Synthesis</td>
<td>Domain Specific UI + Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Presentations</td>
<td>Sketch-n-Sketch</td>
<td>???</td>
</tr>
<tr>
<td>Documents</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Web Apps</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Programs</td>
<td>PL</td>
<td>DM</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>2D Graphics</td>
<td>General Purpose PL + General Program Synthesis</td>
<td>Domain Specific UI + Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Presentations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documents</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs</td>
<td></td>
<td>DM Text</td>
</tr>
</tbody>
</table>

Sketch-n-Sketch

Interactive General Purpose IDE

Prototype; Repair; Refactor; Repeat
END