Direct Manipulation Programming Systems

Ravi Chugh
Brian Hempel
Jacob Albers
Grace Lu
Justin Lubin
Mitch Spradlin

February 20, 2017
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 = ...

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

```
circle1 = ...
p1 = left(circle1)
p2 = 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 = ...

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

```
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="image1" alt="Ai" /></td>
</tr>
<tr>
<td>Presentations</td>
<td><img src="image2" alt="P" /></td>
</tr>
<tr>
<td>Documents</td>
<td><img src="image3" alt="W" /></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td><img src="image4" alt="X" /></td>
</tr>
<tr>
<td>Web Apps</td>
<td><img src="image5" alt="d" /></td>
</tr>
</tbody>
</table>
Programming Languages

- 2D Graphics
- Presentations
- Documents
- Spreadsheets
- Web Apps

Domain-Specific Languages or Libraries

- JavaScript
- LaTeX
- SQL
- Markdown

Domain-Specific GUI Tools

- Adobe Illustrator
- PowerPoint
- Word
- Google Apps
- Google Web Designer
<table>
<thead>
<tr>
<th></th>
<th>Programming Language</th>
<th>Direct Manipulation</th>
<th>Direct Manipulation Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td>JS</td>
<td>AI</td>
<td>Sketch-n-Sketch</td>
</tr>
<tr>
<td>Presentations</td>
<td>JS</td>
<td>PPT</td>
<td>???</td>
</tr>
<tr>
<td>Documents</td>
<td>JS</td>
<td>W</td>
<td>???</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>JS</td>
<td>SQL</td>
<td>???</td>
</tr>
<tr>
<td>Web Apps</td>
<td>JS</td>
<td>CSS</td>
<td>???</td>
</tr>
</tbody>
</table>
Our Approach

- Reuse effort across domains.
- Enable experts and library writers to extend built-ins.
- Smooth spectrum between "Experts" and "End Users".

\( \lambda \text{-calculus} + \text{Direct Manipulation} \)
Sketch-n-Sketch Demo
Sketch-n-Sketch Demo

Programming with:
Less Keyboard.
More Mouse.
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 $e$:

$\quad v = t$

Constraint on Updated $e'$:

$\quad v' = t$
Dataflow-Only Traces

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

a+b

a+b*b

(\x.x) a

true ? a : b
110 = x0 + 2 * sep

\[ \downarrow \]

155 = x0 + 2 * sep

Change \textit{x0} or \textit{sep}? Or both? Or more?

Change exactly one constant.
Heuristics to automatically choose.
Annotations to freeze (\textit{n}!) or thaw (\textit{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.

\[
\begin{align*}
5\sin(x) &= n & \checkmark \\
(x+y^2)x &= n & \checkmark \\
(x+y^2)x &= n & \times \\
x^2 &= n & \checkmark \\
x^x &= n & \times
\end{align*}
\]

Good enough for 80% of equations.
Draw
Relate
Group
Tweak

"Raw" Shape

(x1, y1)
(x3, y3)
(x2, y2)
"Anchored" Shape

(x1, y1)

(x1+2, y1+8)

(x1+10, y1+10)
Draw

Relate

Group

Tweak

"Bounded" Shape

\[(x_1, y_1)\]

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

\[(x_2, y_2)\]
Draw | Relate | Group | Tweak

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] ... 
    ... ))))
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
"Anchored" Group

"Bounded" Group
"Anchored" Group

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

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

(def line3
  ... )
(let x1 50 ... ))

(def line3 ...
 ... )
(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])

(clean)

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

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

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

(def [rect1_x' line2_x1'] [rect1_x rect1_x])

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

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

(def line3
  ... )

CLEAN Redundant Vars
(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
Draw  Relate  Group  Tweak

Programming in 2015

Sketch-n-Sketch

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

Sketch-n-Sketch

e
e

⇓

⇓

⇓

\[ e_1 \]

\[ e_2 \]

\[ e_3 \]
Less Reliance on Syntactic Structure
; Top-Level Defs
(def rect1 ...)
(def line2 ...)
(def line3 ...)

; Main Expression
[ rect1 ... ]
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_bo... (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: Sketch-n-Sketch
Limited, Syntactic Program Updates

Relate: Sketch-n-Sketch ++
Expressive, General Program Synthesis

Group: Sketch-n-Sketch
Tweak: Sketch-n-Sketch ++
Draw  Relate  Group  Tweak
Draw  Relate  Group  Tweak

Idea

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</td>
<td>Domain Specific UI</td>
</tr>
<tr>
<td>Presentations</td>
<td>General Program Synthesis</td>
<td>+</td>
</tr>
<tr>
<td>Documents</td>
<td></td>
<td>Domain Specific Program Synthesis</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Web Apps</td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Prototype; Repair; Refactor; Repeat

Sketch-n-Sketch
Prototype; Repair; Refactor; Repeat

<table>
<thead>
<tr>
<th></th>
<th>PL</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Graphics</td>
<td><img src="image" alt="2D Graphics" /></td>
<td><img src="image" alt="Sketch-n-Sketch" /></td>
</tr>
<tr>
<td>Presentations</td>
<td><img src="image" alt="General Program Synthesis" /></td>
<td><img src="image" alt="Domain Specific UI" /></td>
</tr>
<tr>
<td>Documents</td>
<td><img src="image" alt="General Program Synthesis" /></td>
<td><img src="image" alt="Domain Specific Program Synthesis" /></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td><img src="image" alt="General Program Synthesis" /></td>
<td><img src="image" alt="Domain Specific Program Synthesis" /></td>
</tr>
<tr>
<td>Web Apps</td>
<td><img src="image" alt="General Program Synthesis" /></td>
<td><img src="image" alt="Domain Specific Program Synthesis" /></td>
</tr>
</tbody>
</table>
λ-Calculus + Direct Manipulation of Output

1: Mouse-Edit Output
\( \lambda \text{-Calculus} + \text{Direct Manipulation of Code} \)
Programming with
Less Keyboard.

Programming with
More Mouse.
Related Work

**DM + Code**
- Dynamic Drawing Apparatus

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

**Text + Structure Editors**
- Barista
- Drag-n-Drop Refactor
- 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

0: Text-Edit **Code**

1: Mouse-Edit **Output**

2: Mouse-Edit **Code**
EXTRA SLIDES
Sketch-n-Sketch

Draw
Relate
Group
Tweak
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>???</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>

Sketch-n-Sketch
## Prototype; Repair; Refactor; Repeat

### Programming Domains

<table>
<thead>
<tr>
<th>Domain</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>??</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>??</td>
</tr>
</tbody>
</table>

### Tools

- Sketch-n-Sketch
- Interactive General Purpose IDE
- DM Text
\(\lambda\)-Calculus + Direct Manipulation

Reuse effort across domains.

Enable experts and library writers to extend built-ins.

Smooth spectrum between "Experts" and "End Users".
Text editing is great, but…

Many mundane, low-level edits.

Especially with generated code.
Idea: Selectable AST Nodes

\[
\begin{aligned}
\text{(def foo (let [a b] [1 2!]} \\
&\text{[a b ?]]) foo}
\end{aligned}
\]
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 50 ... ))

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

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

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

(def line3 ...
  )

(def rect1_x 52)

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

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

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

(def rect1_x 52)
(def rect1
  (let x rect1_x ... ))
(def line2
  (let x1 rect1_x ... ))
(def 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, …

```scheme
(def rect1_x 52)

(def line3_x1 200)

(def line3
  (let x1 line3_x1 ... ))

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

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