# Representation-Plurality in Multi-Touch Mobile Visual Programming for Music

Qi Yang
Computer Science & Engineering Division
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
yangqi@umich.edu

Georg Essl
Electrical Engineering & Computer Science and Music
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@eecs.umich.edu

## ABSTRACT

Multi-touch mobile devices provide a fresh paradigm for interactions, as well as a platform for building rich musical applications. This paper presents a multi-touch mobile programming environment that supports the exploration of different representations in visual programming for music and audio interfaces. Using a common flow-based visual programming vocabulary, we implemented a system based on the urMus platform that explores three types of touch-based interaction representations: a text-based menu representation, a graphical icon-based representation, and a novel multi-touch gesture-based representation. We illustrated their use on interface design for musical controllers.

## Author Keywords

NIME, proceedings, Interaction design and software tools, Mobile music technology and performance paradigms, Musical human-computer interaction

## ACM Classification

D.1.7 [Software] Visual Programming, H.5.5 [Information Systems] Sound and Music Computing, H.5.2 [Information Systems] User Interfaces — Interaction Styles

## 1. INTRODUCTION

The growing popularity and ubiquity of personal mobile devices enabled creative touch-based interactions for a wide audience. As the sensing and computational capacity of these devices become sufficient for rich, creative musical applications, many efforts also seek to make audio and music programming accessible on mobile (e.g. [5, 19]).

A variety of approaches to mobile music programming exists, some uses text-based programming on PC and web which are then distributed to device (e.g. [17, 21]). Multi-touch interface on mobile is different in many aspects from traditional mouse and keyboard input. For example, hand occlusion and lower pointing accuracy of finger requires larger tap targets and other adaptations [8, 14]. Some approach this by adapting text-based and visual block-based programming [19] for touch, borrowing from extensive works on visual programming for PCs. We present an on-device system that is based solely on visual programming.


*NIME'15,* May 31-June 3, 2015, Louisiana State Univ., Baton Rouge, LA.


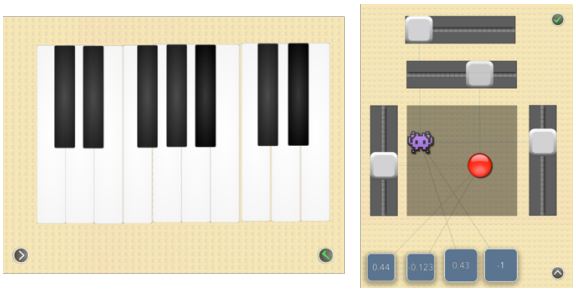

**Figure 1: (left) Touch-based piano keyboard interface and (right) music mixer interface built with our environment.**

Eaglestone et al. [4] found that electroacoustic musicians' preference for musical tools correlates highly with varying cognitive styles, suggesting that no single interaction style in musical programming is best-suited for everyone. We explored this design space of visual representation on mobile music programming by building an environment where simple dynamic interfaces can be interactively assembled, and implemented three different visualization and interaction paradigm using the same underlying visual language.

## 2. RELATED WORKS

The design of our system draws from previous work in visual programming, mobile music programming, and visualization for multi-touch.

### 2.1 Visual Programming

General purpose visual programming has being explored extensively ever since graphical interface became prevalent on PCs, often with the aim of making programming accessible to a wider audience. Visual programming are popular in the interactive media domain, often using a data-flow visual metaphor such as Pure Data [16] or Max/MSP. The visual aspect of programming is important in live coding, where programming-as-performance is closely coupled with audio-visual media. Some live coding languages are designed with a predominantly visual component [2, 13], since the performances usually rely on exposing and conveying the programming process to the audience.

The design of our visual environment draws from a number of previous visual programming representations and vocabularies, from data-flow to interactive or live programming (there is no distinction between editing and running of the program in our system).

## 2.2 Mobile Music Programming

General mobile programming environments such as Hopscotch [1] adapts a block-based visual metaphor. Others use block or scripting languages on mobile with a mixed text, iconic based interface (e.g. LiveCode[2], [22, 12]). Closest to our approach in mobile programming is Pong Designer [11], which combines a 2D physics engine with directly manipulatable objects in a sandbox. Programming logic and causal relations can be inferred by events to build simple games.

The common graphical representation of our system also uses a visual sandbox environment for assembling elements, eschewing text-based coding. Unlike most environments where a program is built and then ran, our system is fully interactive with no distinction between assembling a program versus running one. For now our environment is not general purpose but enables construction of simple music interfaces.

In music and audio domain, programming for mobile devices have received growing attention in recent years. Multiple approaches use text-based or visual language for audio and interface programming and distribute it remotely to mobile devices, using frameworks such as Max/MSP [17]. urMus provides self-contained on-device text-based programming framework accessible through web browser [5]. While not designed for building graphical music interfaces, ChucK audio programming environment has also been adapted for touch interface with some block-based visual elements [19]. Our work seeks to provide a platform on which musical interfaces can be created directly on device without text-based programming.

## 2.3 Visualization and Multi-Touch Gestures

Visual feedback is an important part of multi-touch interfaces. For example visual feedback is found to lead to significant improvement on accuracy of pointing/crossing tasks on touch-screens [10]. One main criticism by Donald Norman on gestural interfaces in consumer software is the lack of feedback to guide learning as well as execution of gesture commands [15].

There has been many approaches to address this using visual feedback for multi-touch gesture interactions. Many have focused the ease of recognition and recall of multi-touch gestures, but mapped arbitrarily to commands [18, 6]. Using simulated physical objects for visual affordance has also being explored [3]. Many works used continuous visualization to show possible commands given the current state of interaction [1, 9, 20], showing potential gestures either at the site of the interaction (near the finger touch location), or mirrored at a more visible location.

We explored three different visual representations in our system. The menu mode uses only single tap gestures. The icon and gesture mode make use of more complex drag selection or drag-and-drop gestures, and use different visuals to guide the gesture interactions continuously.

# 3. REPRESENTATION DESIGN

Our system is built on the urMus [5] platform. urMus provides a set of Lua API for mobile phone sensors and audio and graphics programming, using which a block-based drag-n-drop interface can be implemented. For us, urMus serves as a general purpose platform for prototyping representations and interactions.

[1] `http://www.gethopscotch.com`
[2] `http://livecode.com`

## 3.1 Shared Grammar

The basic grammar of the visual environment starts with a *full-screen* canvas where user can create basic elements called regions and arrange them spatially, as well as other elements that are discussed below. This is where a user creates a dynamic interface (see Figure 1 for example). Next we present the general programming and interaction elements that will be shared between all representations.

**Region:** The most basic building block is a Region (Figure 2 (a)). Each region is visually represented by a rectangle on screen and can be directly manipulated via dragging and resized via pinching gesture. Regions can be created with a simple tap gesture on the canvas, and can be arbitrarily arranged on canvas. Their visual appearance can also be modified, and they can be pinned to the canvas to prevent movement.

Regions possess characteristics of a generic variable or object in the context of programming, in that they can be used both as abstract containers for values such as a variable, and as a visual object in the interface. All regions and can send and receive events (similar to function calls). Regions can be configured to have other functionalities as well, such as playing an audio sample (which may be useful for building musical instruments) or moving its position on screen. Similar to how class can be instantiated as many times as needed, regions can be duplicated while retaining its properties and links (which defines how it interacts with other regions). In all representation modes regions are created by tapping on the empty canvas, and its appearance is changed using a common texture picker interface (Figure 3).

**Links:** Following the concepts of node and edges in event-driven data-flow languages, each region can receive *events* through links and respond to them with *actions*. The routing of these events between regions constitute the main mechanism of building interactions in the visual environment. Links are visually shown as lines connecting one region to another, similar to visual patching interfaces. Each link is directional and stores an event type that is generated from the sender of the link and an action that is to be taken by the receiver when the event is detected.

In the example in Figure 2 (c), a region on the left is linked to send its vertical position value to the region on the right (which displays a numerical value), acting as a vertical slider.

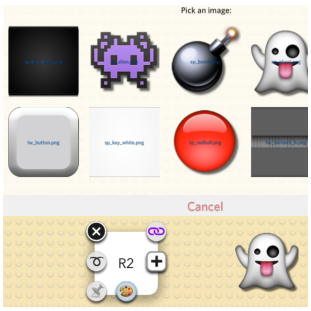


**Figure 3: Interface for changing a region's texture**

This mechanism can be used for building more complex interactions, a region can respond to multiple events from multiple sources, as well as respond to each event with multiple actions. Events can include touch-based events (fired when the region is dragged, tapped, or held etc.) and conditional on its properties. The actions that can be triggered by events can changing the spatial properties of the region (size, position or movement), or passing the event to another region. When a region is duplicated, all the incoming and outgoing links are also duplicated, preserving its interaction between other regions.

We implemented a basic set of touch-based events and actions. The move event which are evoked when the region is dragged and sends the current position of the region; this

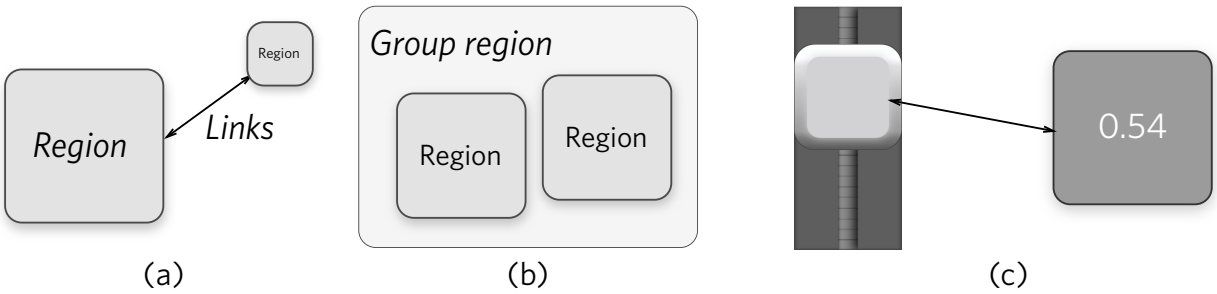


**Figure 2: (a) Regions and links (b) groups, (c) an example slider**

event can be responded by move action which move the receiving region in the same direction and distance, or display one of the coordinates of the received position and send it to a music synthesizer.

**Grouping:** We use groups to encapsulate and organize set of regions spatially. A group region spatially constrains the movement of its child regions, so they cannot be moved out of the group. It can be used to create common interface widgets such as slider (see Figure 1 and Figure 2 for examples). Because each group is also treated as another region, it can receive and respond to events, as well as being duplicated.

## 3.2 Visual Representation Modes

Given the above shared mechanisms we implemented three types of representations for manipulating and interacting with them: (1) Menu-driven, (2) Icon-driven, and (3) Gesture-driven.

**Menu-Driven Mode:**

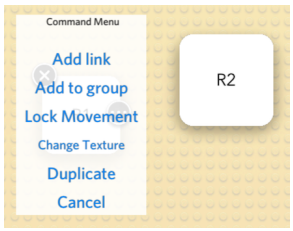


**Figure 4: The text-based contextual menu**

We created a menu representation mode which uses text menu that has similarity to traditional desktop UI. Except for region and link deletions, and the shared gestures mentioned above, every other command such as linking or grouping are activated through a contextual text-based menu (Figure 4) that is associated with one region. Deletion buttons shown for each region whenever the menu is activated by a single tap.

For commands that require a second region (for example, creating a link between two regions), the user activates the command and then is prompted to select the second region (region to create link to, or in case of creating a group, the region to be used as the container) by tapping. After the selection the command is executed on the two regions. Creation of groups uses the identical steps.

Due to its widespread use on different platforms, we expect this representation/interaction mode to be most familiar to average users who have some prior experience with PCs and commercial touchscreen operating systems such as iOS or Android, where text-based menu or list interface is common.

**Icon-Driven Mode:**

In icon-driven mode, most commands are accessed through a contextual graphic symbolic menu (Figure 5). The icons are arranged similar to a radial layout surrounding the region, which borrows from previous work on radial menu design for touch-screens ([7] for example).

Two types of gesture interactions exist in the icon-driven

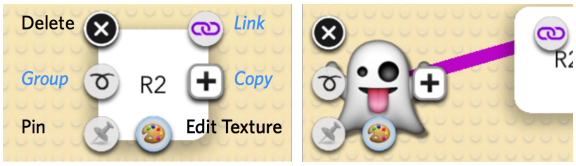


**Figure 5: Left: Icon mode contextual menu with labels, right: draggable icon handle for linking**

menu. Static buttons can be activate by tapping (for example, the Delete button on the top left corner), while draggable buttons acts like handles for more complex drag and drop gestures. These drag gestures activates different types of semantically related commands. For linking, the link icon can be dragged and dropped on the region to link to, the potential link is visually shown using a cable-like metaphor (Figure 5). The grouping gesture handle is used as a lasso selector to select other regions to add to the parent group region. The copy handle is visually represented as a smaller version of the region, can be tapped and produces a copy of the parent region. It can also be dragged into any other area on screen and produces a copy of the parent region at that location when released (Figure 5). To visually differentiate draggable gesture handles from the static buttons, the draggable handles are animated periodically when not used, moving slightly away and back to their original position, as visual affordance suggesting that they can be dragged as opposed to responding only to taps.

**Gesture-Driven Mode:**

The last representation mode is designed to be almost entirely driven by direct manipulation gestures that act on the regions. In our approach, gestures are designed for directly manipulating the on screen elements (regions in this cases). The mapping between gestures and the associated command is not arbitrary, but semantically related to or reinforce the command being triggered.

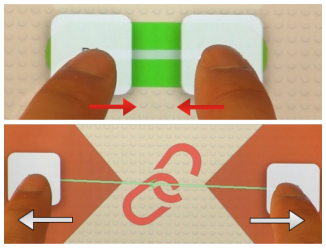

**Figure 6: Gesture mode uses pinch gestures for creating and deleting links.**

For linking and unlinking two regions, a pinch or stretch gesture is used (Figure 6). A pinch gesture where two regions are dragged concurrently and moved towards each other is used for linking, while the opposite, moving away from each other is used for unlinking, reinforcing the concept of linking and unlinking. For each gesture, the regions have to be moved past a threshold (which is visually

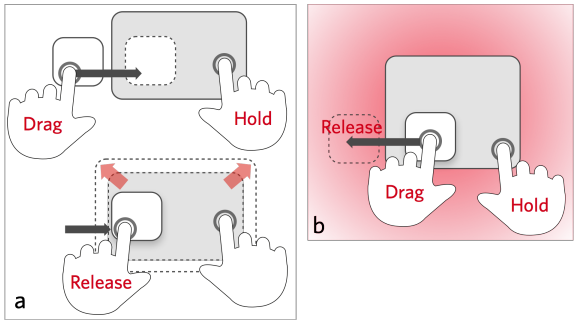


**Figure 7: A drag-n-drop gesture is used to add a region to a group (a), the reverse for removal (b).**

shown when reached) for the action to be completed, this acts as a confirmation for each action to prevent mistakenly activating gestures. If the threshold is not crossed then regions are automatically restored to their previous positions and the action is cancelled.

A background colour guide is displayed faintly after the initiation of the gesture as a visual guide, and continuously updated depending which direction the user is performing the gesture (pinch or stretch), the colour area corresponding to the potential command is displayed in increased intensity while the colour corresponding to the opposite command is faded (see Figure 6 for the visual guide). The large colour background is meant to alleviate the problem of occlusion by the user's hand.

For grouping regions, a drag and drop gesture is used. When two regions are dragged, and one is released over another one with a larger size, the smaller or released region is added to the group associated with the larger region (Figure 7). To remove a region from a group, the reverse gesture is used: the user simply drags both the region and its parent group region, and then move the child region outside of the group region and release. The movement restriction of child regions within their group region is temporarily lifted while this gesture is performed, and restored after the remove-from-group action is either executed or cancelled. Similar to the linking gesture, visualization provides guidance in enlarging the potential group region or showing a drop-zone for the removing gesture with background colour.

The region creation gesture of a single tap on the empty canvas (shared among all three interaction modes) is modified to support copying. While holding onto a source region to be copied, tapping the empty space on the canvas creates instead a copy of the source region. Since there is no need to release the hold on the source region, this allows for efficient creation of copies.

To pin a region, a double tap gesture is used to toggle between restricting or allowing movements (see Figure 8 (c)). For region deletion and modifying a region's texture, a hold-and-slide gesture menu is used. The gesture menu is displayed after holding on the region for a short time and while no other commands (moving, resizing etc.) are activated, and its two commands are activated by maintaining the hold gesture while sliding towards the command icon (similar symbol as the icon mode) and releasing the finger. Figure 8(b) shows the visual feedback for activating each commands after the hold-and-slide gesture.

Continuous visual feedback is also used for interaction modes that relies on drag gestures (e.g. pinch, lasso selection), in previous works continuous visual feedback has been found to be beneficial in visual programming context [23]. In icon mode, the cable and lasso selection visualization are continuously animated, and in gesture mode the

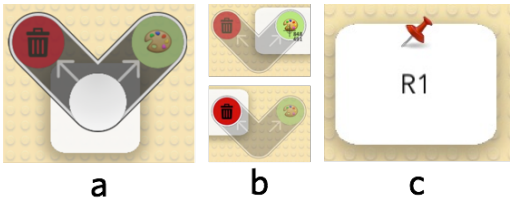


**Figure 8: (a, b) hold-gesture-based menu and its different activation states, (c) visual feedback for pinning a region**

colour background is continuously updated during the linking gesture (Figure 6).

# 4. SYSTEM IMPLEMENTATION

Our system was implemented on top of the urMus framework using Lua scripting language. The system consists of the functional language components (which include classes that represent the basic entities mentioned earlier: Region, Link and Group) and visual interface components that enables the interaction and feedback.

In the language components, `region` is an extension (subclass) of the built-in visual region class in urMus, as it retains appearance properties and drag-n-drop interactions. In our environment, `region` class also stores interaction states as part of the gesture state-machine, as well as mechanism for dispatching and responding to events (those created by user interaction, and sent from other linked entities), managing links, and managing associated contextual interfaces for text and icon-based menus.

`Group` is a subclass of the `region` class, which acts simply as a container for other `region` objects, providing methods for managing its child regions.

`Link` object is a utility class for connecting two `region` objects, and managing the flow of events from object to object in a single direction (specified sender and receiver). Since `group` is a subclass of `region`, they can also be connected by `links` as either a sender. Visual appearance of link objects on screen is handled centrally by a separate class which draws and updates all links on screen.

## 4.1 User Interface

Other than the visual representation of Regions and Links, the rest of the user interface are managed by text and icon menu classes, and two utility singleton classes for gesture recognition and visual gesture guides.

Text menu and icon menu classes are implemented similarly using textured urMus regions and text labels. They are designed to be reusable and are instantiated and configured each time they are needed. Callback methods/functions are used for specifying each commands in the menus.

Gesture interactions are mediated by two singleton classes, `gesture manager` which handles the actual multi-touch interaction state-machine, and `gesture guide` which draws the visual guides for gestures on screen. The multi-touch gesture state-machine recognize each gesture such as pinch or drag-and-drop-onto by listening for low-level touch events from all regions in the environment, and triggering the assigned actions (grouping, linking) when action states are reached. The modular design of the `gesture manager` allows a variety of gestures to be swapped in by specifying different state-machines within the manager class, without changing the region objects which are involved in the actual gesture.

`Gesture guide` is called to update onscreen visual feedback by the manager class, allowing the visualization of ges-

tures to be configured separately from the recognition.

Other utilities including logging which records each touch events and user-triggered actions in a detailed log on device, and notification class providing simple onscreen help messages.

In addition to visual interface, a sound module currently in development serves as a conduit to some of urMus's sound synthesis API, allowing user interaction from the programming environment to be fed into sound synthesis parameters, enabling the building of musical controllers. In the example in Figure 1, each slider controls a different parameter of a simple sine oscillator.

## 5. CONCLUSION

We presented a visual programming environment for creating music interfaces. Our system allows a variety of representation and interaction modes using a basic grammar that supports the construction of musical controller interfaces. All paradigms are visual and we explore the potential of using both familiar and novel gesture-based multi-touch paradigms to construct representations. Ongoing and future work include evaluation of the different representations with respect to their usability in musical end-user programming. We are currently in the process of conducting and analyzing a user study comparing the different representations. Furthermore we plan to create an accessible API helping musicians embed their preferred multi-touch interaction paradigm in their own performance interfaces.

## 6. REFERENCES


[1] O. Bau and W. E. Mackay. OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets. In *the 21st annual ACM symposium*, pages 37–46, New York, New York, USA, 2008. ACM Press.

[2] A. Blackwell and N. Collins. The Programming Language as a Musical Instrument. *Proceedings of PPIG05 (Psychology of Programming Interest Group)*, pages 120–130, 2005.

[3] A. Bragdon, A. Uguray, and D. Wigdor. Gesture play: motivating online gesture learning with fun, positive reinforcement and physical metaphors. In *ITS 2010*, pages 39–48, Saarbru ÌĹcken,Germany, 2010.

[4] B. Eaglestone, N. Ford, and P. Holdridge. Are Cognitive Styles an Important Factor in Design of Electroacoustic Music Software? *Journal of New Music*, 2008.

[5] G. Essl and A. Müller. Designing Mobile Musical Instruments and Environments with urMus. In *Proceedings of the 2010 conference on New Interfaces for Musical Expression*, Sydney, Australia, 2010.

[6] E. Ghomi, S. Huot, O. Bau, M. Beaudouin-Lafon, and W. E. Mackay. Arpège. In *the 2013 ACM international conference*, pages 209–218, New York, New York, USA, 2013. ACM Press.

[7] D. Kammer, F. Lamack, and R. Groh. Enhancing the expressiveness of fingers: multi-touch ring menus for everyday applications. In *AmI'10: Proceedings of the First international joint conference on Ambient intelligence*. Springer-Verlag, Nov. 2010.

[8] S. Lee and S. Zhai. The Performance of Touch Screen Soft Buttons. In *Proceedings of the SIGCHI conference on Human ...*, 2009.

[9] S. Lundgren and M. Hjulström. Alchemy : dynamic gesture hinting for mobile devices. In *the 15th International Academic MindTrek Conference*, pages 53–60, New York, New York, USA, Sept. 2011. ACM.

[10] Y. Luo, D. Vogel, and Y. Luo. *Crossing-based selection with direct touch input*. ACM, New York, New York, USA, Apr. 2014.

[11] M. Mayer and V. Kuncak. Game programming by demonstration. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 75–90. ACM, 2013.

[12] S. McDirmid. Coding at the speed of touch. In *the 10th SIGPLAN symposium*, pages 61–76, New York, New York, USA, Oct. 2011. ACM.

[13] A. McLean, D. Griffiths, N. Collins, and G. Wiggins. Visualisation of live code. In *EVA'10: Proceedings of the 2010 international conference on Electronic Visualisation and the Arts*. British Computer Society, July 2010.

[14] M. A. Nacenta, P. Baudisch, H. Benko, and A. Wilson. Separability of spatial manipulations in multi-touch interfaces. In *GI '09: Proceedings of Graphics Interface 2009*. Canadian Information Processing Society, May 2009.

[15] D. A. Norman and J. Nielsen. Gestural Interfaces: a Step Backward in Usability. *Interactions*, 17(5), Sept. 2010.

[16] M. Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[17] C. Roberts, M. Wright, J. Kuchera-Morin, and T. Höllerer. Rapid creation and publication of digital musical instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2014.

[18] J. s, E. Lecolinet, and T. Selker. Multi-finger Chords for Hand-held Tablets: Recognizable and Memorable. In *CHI 2014*, 2014.

[19] S. Salazar and G. Wang. miniAudicle for iPad: Touchscreen-based Music Software Programming. In *International Computer Music Conference*, 2014.

[20] R. Sodhi, H. Benko, and A. Wilson. LightGuide: Projected Visualizations for Hand Movement Guidance. In *the 2012 ACM annual conference*, pages 179–188, 2012.

[21] B. Taylor, J. Allison, W. Conlin, and Y. Oh. Simplified Expressive Mobile Development with NexusUI, NexusUp and NexusDrop. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2014.

[22] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich. TouchDevelop: programming cloud-connected mobile devices via touchscreen. In *ONWARD '11: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, 2011.

[23] E. M. Wilcox, J. W. Atwood, M. M. Burnett, J. J. Cadiz, and C. R. Cook. Does continuous visual feedback aid debugging in direct-manipulation programming systems? In *CHI '97: Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 258–265, 1997.