Hacking with Multi-touch for Java (MT4j)

Craig Anslow
Department of Computer Science
Middlesex University
London, United Kingdom
c.anslow@mdx.ac.uk

Stuart Marshall, James Noble
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
{stuart,kjx}@ecs.vuw.ac.nz

Robert Biddle
School of Computer Science
Carleton University
Ottawa, Canada
robert.biddle@carleton.ca

Abstract

Developing applications for touch devices is hard. Developing touch based applications for multi-user input is harder. The *Multi-Touch for Java (MT4j)* toolkit supports developing touch based applications for multiple users. In this paper, we outline our experience using MT4j for developing a number of software applications to support developers working in co-located teams. Our experience using the toolkit will help developers to understand the nuances of the toolkit and design issues that can be applied to other toolkits for developing multi-user touch based applications.

Categories and Subject Descriptors H.5.2 [*User Interfaces*]: Input Devices and Strategies

General Terms Design

Keywords Programming, Multi-Touch, Java, Visualization

1. Introduction

Designing touch based applications for devices larger than phones or tablets (e.g. digital tabletops and walls) is challenging especially when you require multi-user collaboration (e.g. pairs or groups) [2]. Some programming languages have touch features but very few of them have been designed to support multi-user input. There are a number of toolkits in different programming languages that support multi-user interaction and multi-touch gestures [13]. A few surveys outline and compare some existing multi-touch toolkits and frameworks [4, 7]. We have explored a number of these toolkits for developing different applications.

The Simple Multi-touch Toolkit (SMT)[1] is based around the concept of touch-enabled zones, is built on top of the Processing toolkit [9], and is also available as a standalone Java library. SMT natively supports TUIO [6] and works on Windows Touch, Leap Motion, and SMART SDK, but also provides a multi-touch simulator to support non-touch enabled environments with a mouse. SMT was designed to support students and used in undergraduate HCI courses experimenting with multi-touch input but has also been used to develop several applications.

PyMT[2] is an open source multi-touch framework for Python. Kivy[3] is another Python based toolkit which evolved from PyMT to make the toolkit more robust. Kivy is an open source library for developing mobile apps and other multitouch application software with a natural user interface (NUI). Kivy can run on Android, iOS, Linux, OS X, and Windows.

lbavg[4] is a C++ touch library and uses OpenGL for display output. libavg runs on on Linux, Mac OS X and Windows, and is open source. Cinder[5] is also a C++, free, and open source library for professional-quality creative coding.

A number of JavaScript gesture based toolkits exist such as Hammer.js[6] for developing touch based applications in a web browser. However, the focus of these web based touch toolkits is on supporting basic atomic touch gestures (e.g. tap, press, pinch, swipe, pan, and rotate) and not necessarily large interactive devices that support multi-user interaction for group work. We are particularly interested in developing applications designed for multiple users and different hardware form factors. We describe our experience at using one of these toolkits, MT4j for developing collaborative software team applications.

[1] http://vialab.science.uoit.ca/smt/
[2] https://github.com/tito/pymt
[3] https://kivy.org
[4] https://www.libavg.de/
[5] https://libcinder.org/
[6] http://hammerjs.github.io/

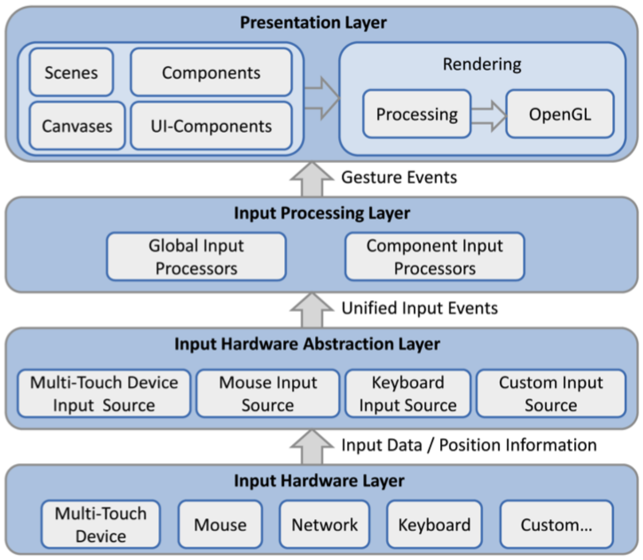


Figure 1. MT4j architecture overview [8].

2. Multi-touch for Java (MT4j)

Multi-touch for Java (MT4j) is an open source (LGPL) multi-touch application development toolkit [8]. MT4j provides high level functionality and aims at providing a toolkit for easier and faster development of multi-touch multiple user applications. The toolkit was developed by Fraunhofer researchers in Stuttgart in 2010. There are several requirements and challenges which MT4j addresses in a generic and reusable way:

Portability: supports different operating systems, hardware detection protocols, and input hardware (e.g. tablets, tabletops).

Input Abstraction: supports common atomic multi-touch gestures and the ability to customize gestures. The gesture recognition and gesture processing is flexible and extensible.

Performance and Graphics: uses high performance rendering capabilities such as OpenGL.

UI Integration: supports a range of UI components which allow extensibility and integration with other Java libraries to develop rich user interfaces.

Figure 1 shows the architecture of MT4j. The architecture contains different layers that communicate through events sent from one layer to the next. The emphasis on the input layers represents the importance of a flexible input architecture. Performance issues are mainly addressed by the presentation layer. The architecture resembles a reduced MVC design, with the input layer representing the controller and the presentation layer representing the view.

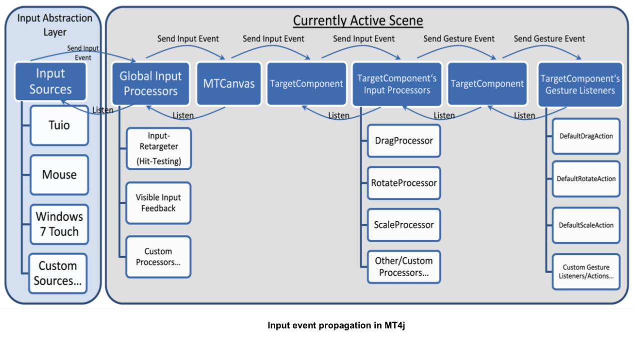


Figure 2. MT4j input processing [8].

Various input hardware is supported with only minimal adjustments required in the input hardware abstraction layer. In this abstraction layer, raw input data is converted into unified input events. A set of input providers exists including mouse, keyboard, and multi-touch input protocols such as WM_TOUCH and TUIO [6].

The input processing occurs at two stages in the event flow, see Figure 2. The first stage is global input processing where input processors can be registered which subsequently listen directly to the various input sources. This stage is used when all input has to be processed and allows modification of user input before it is passed up to the next layer. The second stage is component input processing which allows processing of input that was targeted at one component only (e.g. rotate or scale gestures). The action taken when a gesture event (e.g. tap, double tap, press and hold) is received is determined by the attached listeners which can modify the component's behaviour or appearance (e.g. location or colour).

The presentation layer contains components, scenes, and a canvas. Components range from graphical primitives (e.g. rectangle, polygon, ellipse, and line) to more complex user interface widgets (e.g. menu, keyboard). Scenes encapsulate and separate the input processing and presentation of one aspect of an application from another. An example of using scenes in an application is to have one main content scene and separate visualization scenes that contain different business logic and interfaces. A scene change can be triggered to navigate between scenes. The canvas component is the root component of every scene. The canvas acts as the link between the global input processing layer and the presentation layer. All input events pass through the canvas component, which then further propagates events to their destinations. The canvas also contains methods for checking which components are located at a specified screen position and it is responsible for recursively drawing the canvas with all its child elements. For rendering of components, the Processing toolkit is used [9].

3. Applications

We have developed a number of novel applications that utilize the MT4j toolkit in different ways. These applications range from software visualization, digital cardwalls, gestural interaction techniques, to strategy games. We present two of these applications followed by a discussion about the use of the toolkit and implications for future toolkits.

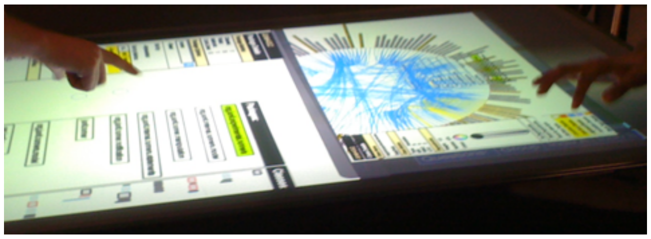
SourceVis [1, 3] is a collaborative application for visualizing the structure and evolution of software systems. The aim is to help developers working in co-located teams explore how a system has been structured by viewing metric and evolution based visualizations. These steps can help identify what parts of a system are large and likely need to be refactored. SourceVis is designed for use on various sized digital surfaces (tabletops and walls) within co-located environments, to support multiple users, and to display multiple visualizations at once. SourceVis is approximately 50K lines of code built on top of MT4j. All the visualizations were developed from scratch but influenced from existing literature, and some Java libraries were integrated to help improve the visualizations. SourceVis has visualized a number of Java systems from the Qualitas Corpus [12]. We evaluated SourceVis with professional developers working in pairs on a multi-touch tabletop to complete a range of software maintenance tasks. We found that the prototype was effective for identifying trends and outliers of classes and packages which highlighted potential refactoring opportunities. Figure 3(a) shows two developers exploring different visualizations about JHotDraw.

Story walls are common among Agile software development teams and are typically a paper artefact. Electronic versions of story walls exist to provide benefits over their paper based counter parts, such as distributed collaboration, content generation, or content archival, but take away from the traditional form of interaction a paper wall has. We developed e-Wall [5] which is a multi-touch Agile story wall designed for a multi-touch tabletop to maintain the original feel and interaction of a paper wall and to support team meetings. The prototype was evaluated with Agile practitioners and found to be effective for supporting Scrum team meetings.

4. Discussion

We now discuss MT4j based on the requirements by Cirelli and Nakamura [4] that a gesture recognition framework or toolkit should meet to support interactive touch applications and from our own experience at developing collaborative applications (§3).

Flexible and Extensible. We used existing atomic gestures and experimented creating some new ones such as drawing shapes for specific tasks. While we could extend the default gesture set the majority of the time participants in our user studies preferred basic atomic gestures for selection, manipulation, and navigation tasks.


(a) SourceVis – developers working individually in parallel synchronously, visualizing metrics and dependencies of a system within different moveable windows.

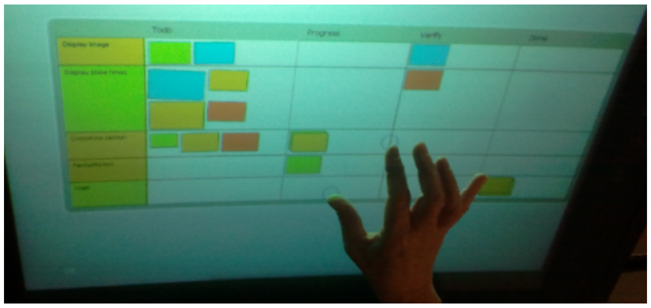

(b) eWall – a developer moving cards on the Agile cardwall.

Figure 3. Sample applications developed with MT4j.

Fast and Accurate. The gestures in general were fast at detecting the touch points from the participants. This was a combination of using the TUIO protocol and the prototype hardware developed by ourselves following existing guidelines [10]. The accuracy of the touch detection was not as precise as required due to calibration of the optical tracking and lighting conditions. The accuracy was also dependent on how participants interacted with their fingers.

Multi-touch and Multiple Users. Multi-touch was very well supported. At any one point in time we did not have more than 50 touch points being detected. There did not appear to be any limit on the number of touch points but we did not test for any extremes. It is difficult to develop multi-user interaction especially when it comes to associating touch points to different users. We have found with all multi-touch toolkits is that they can not distinguish between multiple users interacting simultaneously effectively. These issues can also be highly dependent on the hardware detection platform as well. We found that when participants were working in different areas on the tabletop and different scenes there were no issues. Occasionally when they were interacting with the same scene there were issues with the system being able to distinguish which touch points belonged to each user and subsequently this caused unexpected behaviour and confused the participants.

Spatial Invariance. Gestures could be performed at any location on the surface. For example, we had a global menu for selecting different visualizations which was displayed when a user performed a press and hold gesture on the main canvas component.

Continuous Feedback. A blue circle was displayed around each finger to identify touch points. When a touch and hold gesture was performed a timed red circle around the touch point would be displayed to indicate the start and end of the gesture, and upon completion an action was performed.

Easy Prototyping. A number of basic examples along with an extensions library were available for prototyping applications. A significant amount of effort was required to build something more substantial. Every object that was displayed had to be rendered as a geometric shape so all classes had to extend one of the existing built in shapes. There existed very few user interface controls and menus, hence it was time consuming to build our own widgets. Prototyping in MT4j is not as fast as toolkits like Processing [9].

Symbolic and Time-Constrained Gestures. We did not implement any symbolic gestures. Instead we had buttons that were used as hot keys for getting back to the main canvas if other windows were being display, and buttons to display a window at full screen or close a window. We believe, depending on the use cases symbolic gestures could be implemented as it is important for making smooth interactions. Determining what symbolic gestures to use for certain tasks and making them discoverable for the end-user can be problematic though. Tap and press and hold gestures are supported, but we did now explore other forms of time-constrained gestures.

Territories. We did not explore territoriality (e.g. personal and shared territories) in great detail [11]. The canvas was used as a shared territory. Separate windows could be displayed which facilitated personal territories, whereupon users could move windows to different areas of the surface but with no restrictions on who could interact with the windows.

Free-Form Gestures. Drag and lasso gestures for moving components around the screen and grouping of components is supported. Drag worked relatively well, but lasso for selection was not as effective when there were many objects to select. Then deselecting an object from a group (via another lasso gesture) or dragging the grouped objects was cumbersome and not very effective.

Sequential and Cooperative Gestures. We did not explore whether sequential and cooperative gestures were supported, but would like to do this as part of future work.

Ongoing Development Support. Despite providing a valuable toolkit for the community ongoing development efforts appear to have ceased (at least publicly), even though there are many developers building upon the toolkit. We have also noticed this with other toolkits. Ongoing support is important to sustain these toolkits.

Acknowledgments

This work was supported by the New Zealand Ministry of Science and Innovation, TelstraClear, Victoria University of Wellington, and the Canadian NSERC SurfNet network. Thanks to the original MT4j developers at Fraunhofer and to members of the NUI Community Group for support notably: Seth Sandler, Nolan Ramseyer, Memo Akten, Jordan Hochenbaum and Owen Vallis. Thanks to Roger Cliffe, Pippin Barr, and Rilla Khaled for technical help. Thanks to the students who developed software prototypes.

References

[1] C. Anslow. *Collaborative Software Visualization in Co-located Environments*. PhD thesis, Victoria University of Wellington, 2013.

[2] C. Anslow, P. Campos, and J. Jorge, editors. *Collaboration Meets Interactive Spaces*. Springer, 2016.

[3] C. Anslow, S. Marshall, J. Noble, and R. Biddle. SourceVis: Collaborative software visualization for co-located environments. In *Proc. VISSOFT*. IEEE, 2013.

[4] M. Cirelli and R. Nakamura. A survey on multi-touch gesture recognition and multi-touch frameworks. In *Proc. International Conference on Interactive Tabletops and Surfaces (ITS)*, pages 35–44. ACM, 2014.

[5] M. Crisp. e-Wall: a multi-touch agile story wall. Honours Report, Victoria University of Wellington, 2011.

[6] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. TUIO – a protocol for table based tangible user interfaces. In *Proc. International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005.

[7] D. Kammer, G. Freitag, M. Keck, and M. Wacker. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. In *Proc. Workshop on Engineering Patterns for Multitouch Interfaces*, 2010.

[8] U. Laufs, C. Ruff, and J. Zibuschka. MT4j – a cross-platform multi-touch development framework. In *Proc. Workshop on Engineering Patterns for Multi-Touch Interfaces at EICS*. ACM, 2010.

[9] Casey Reas and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, 2007.

[10] J. Schöning, P. Brandl, F. Daiber, F. Echtler, O. Hilliges, J. Hook, M. Löchtefeld, N. Motamedi, L. Muller, P. Olivier, T. Roth, and U. von Zadow. Multi-touch surfaces: A technical guide. Technical Report TUM-I0833, University of Munster, 2008.

[11] S. Scott, S. Carpendale, and K. Inkpen. Territoriality in collaborative tabletop workspaces. In *Proc. CSCW*, pages 294–303. ACM, 2004.

[12] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of Java code for empirical studies. In *Proc. APSEC*, pages 336–345. IEEE, 2010.

[13] J. Wobbrock, M. Morris, and A. Wilson. User-defined gestures for surface computing. In *Proc. CHI*, pages 1083–1092. ACM, 2009.