The Simple Multi-Touch Toolkit

Kalev Sikes, Zachary Cook, Erik Paluka, Mark Hancock, and Christopher Collins

Introduction

The popularity of mobile devices and large interactive displays has brought the touch input paradigm into the limelight. Individuals from various domains are eager to take advantage of the benefits of this interaction style. The problem is that the differences from mouse and keyboard input often create barriers for non-expert programmers to prototype their ideas. The lack of familiarity of the unique requirements for surface application development has inhibited the proliferation of this platform as a medium for research, design, and art. To mitigate this problem, surface computing education needs to be incorporated into the curricula of programs in computer science (CS), information systems, and digital media. In order for this to happen we need tools which can be successfully used by people of different programming skill levels, and which support the rapid prototyping of applications. Existing toolkits for surface development tend to be too complex for non-CS majors to use. In addition, the time required to create a prototype using these toolkits prevents them from being integrated into high paced human-computer interaction courses. To solve this dilemma, we have created the Simple Multi-Touch toolkit (SMT).

With a focus on education and interdisciplinary use, the main goal of our open source toolkit is to simplify the prototyping process for people from differing domains whose programming skill levels range from novice to expert. As a library for the Processing programming language (Reas and Fry, 2006), our toolkit has a simplified syntax and an accessible graphics model. Its high-level nature makes surface development a more inclusive activity and less daunting for beginners. Novices are able to take advantage of its features without knowing CS concepts such as object oriented and event-driven programming. The toolkit is also beneficial for expert programmers since it is highly customizable, efficient, and provides access to low-level input data and graphical primitives.

To further reduce the knowledge and time required to develop surface applications, SMT is device agnostic through the integration of many input bridges. People no longer have to spend a considerable amount of time customizing their application or use multiple toolkits to develop for different platforms. These design choices have resulted in a robust toolkit that has been used, with success, at multiple universities for developing research prototypes to full-fledged applications. The tool has also been integrated into HCI courses at two universities to facilitate the teaching of prototyping to non-programmers and multi-touch computing to CS students.

Our primary contribution is a simplified software toolkit that can reduce the amount of time required for prototyping by both programmers and non-programmers. We also briefly describe our experiences and resulting insights gained from using this toolkit over the span of two years in HCI courses, as well as for research and application development.

Related Work

With the advent of computer vision frameworks (NUI Group, 2013; Gokcezade et al., 2010; Kaltenbrunner, 2009), the creation of multi-touch systems has become increasingly prevalent. With this rising popularity, researchers have been working on ways to reduce the difficulty of developing for these platforms (Kammer et al., 2010). As a result, multi-touch toolkits for different programming languages have been designed (Hansen et al., 2009; Khandkar et al., 2010; Laufs et al., 2010; Leftheriotis et al., 2012; Luderschmidt et al., 2010; Nebeling and Norrie, 2012). While reducing development complexity is important, supporting rapid prototyping is equally so, as it allows the evaluation of design decisions with minimal effort (Tang et al., 2011) resulting in an improved design process (Olsen, 2007).

To support rapid prototyping in post-WIMP design, König et al. created Squidy, which uses semantic zooming and visual dataflow programming to make development accessible to novices with the ability to provide advanced features when needed (König et al., 2010). T3 is an interactive tabletop toolkit meant for prototyping high-resolution (multi-projector) applications (Tuddenham and Robinson, 2007). To facilitate prototyping interfaces for shared interactive displays, such as interactive tabletops, Shen et al. (2004) developed the DiamondSpin toolkit, which works exclusively with DiamondTouch tables. Specifically focusing on gaming, Marco et al. (2012) created a software toolkit to ease the prototyping of tangible games for vision-based interactive tabletops. Hasen et al. (2009) present the PyMT toolkit, with a specific focus on a new event model to support flexible and creative design of multi-touch widgets and interactions in a post-WIMP environment. Our SMT toolkit similarly supports rapid prototyping of surface applications, but we focus on the Processing model of coding as sketching, and designed it to support teaching multi-touch programming in classroom environments as well as enabling digital media expressivity and creativity.

Pedagogical software toolkits have ranged from teaching students skills related to art (Ariga and Mori, 2010) to more traditional computer science concepts and skills (Kobayashi et al., 2006; Murshed and Buyya, 2002). Toolkits have been shown to lower the skill barriers for entry and reduce development viscosity when creating user interface applications (Olsen, 2007). For example, Hornecker and Psik (2005) effectively used the ARToolKit to teach students how to prototype tangible interfaces. In this work, we target the Processing programming language to create a toolkit which is useful for both prototyping and education for multi-touch applications. Processing is a high level programming language and development environment designed to enable nontechnical people to use computational methods in the creation of their projects (Reas and Fry, 2006). Our toolkit augments Processing by providing the first comprehensive library of high level methods and features targeted at reducing the complexity of surface development and supporting educators in teaching the fundamentals of surface computing.

Design Goals

The ability to use one's hands and fingers to interact with digital information is a promising technology for a variety of creative applications and interfaces. Hardware supporting collaboration, in the form of tabletop and wall displays, is becoming more common and significant continued growth is expected (Jain, 2014). For a variety of reasons, including variable content orientation, multiple simultaneous inputs, the prevalence of direct manipulation, and a need to support co-located collaboration, traditional WIMP (Windows, Icons, Menus, and Pointers) interfaces are undesirable for many multi-touch usage scenarios. We have designed SMT for non-programmers and programmers alike to be able to rapidly prototype creative and novel interfaces and techniques that make use of multi-touch interaction.

We chose Processing as our target language for several reasons. Processing supports teaching the fundamentals of computer programming, and has been used for this purpose in many different educational contexts around the world, including high school, university, and online courses in visual arts and computer science, and has been downloaded over two million times (Reas and Fry, 2015). The Processing platform already has many powerful graphical libraries, which support the rapid prototyping of beautiful, creative sketches.

It has an easy deployment pathway for installation of libraries directly in the IDE, and a wide variety (e.g., sound, networking, data, math, etc.) of libraries are already available. Processing is built around a flexible programming model supporting three levels of development (Reas and Fry, 2003):

Simple: single line programs
Novice: hybrid procedural/object-oriented style
Expert: full object-oriented (Java) style

In addition to supporting this multi-level coding flexibility, we built the SMT toolkit using the following design objectives, derived from our experiences in teaching modules on multi-touch computing in HCI courses:

Multi-touch for the masses. The toolkit was designed to allow people to rapidly create prototypes with little knowledge of programming. We focussed specifically on allowing access to touch interaction and common multi-touch components, without the need for an understanding of object-oriented programming (OOP) or events.

Ability to sketch multi-touch ideas. The toolkit was designed to allow for the sketching of multi-touch interfaces and interaction techniques. Specifically, we focussed on minimizing code required to have a working multi-touch interface that enables the testing of design ideas, rather than on polishing the look and feel of interface components or developing a robust application ready for deployment.

Ability to code multi-touch in a one-hour lab session. The toolkit was also designed to enable students to go from no experience with multi-touch programming to creating a simple multi-touch interface in a one-hour lab session. Specifically, the toolkit was designed with the intent of allowing courses to focus content on the design aspect of multi-touch, rather than the in-depth programming understanding required to make working multi-touch systems.

Support for a variety of platforms and inputs. The toolkit is cross-platform, running on Windows, Mac, and Linux. An Android version is also available but requires a custom build of Processing to use it. SMT was designed to support native (e.g., Windows) touch events, as well as popular input providers such as the TUIO protocol (Kaltenbrunner, 2009). SMT also supports touch emulation using a mouse.

Support both novices and experts. The toolkit was designed for use in teaching of HCI courses where students range from students in programs such as visual design or management ("novices"), to fourth year CS students ("experts"). Similarly, the toolkit was designed to support quick sketching of small ideas (e.g., lab assignments) as well as development of large projects (e.g., graduate student research or interactive artwork). This was achieved through a flexible syntax in which there are multiple avenues for achieving the same result.

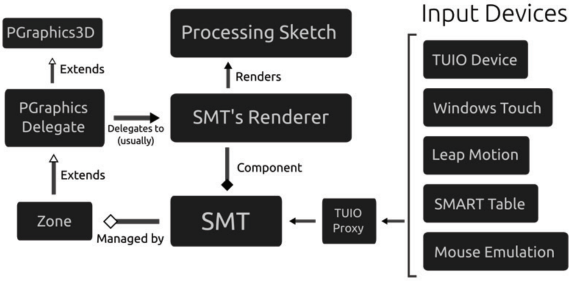


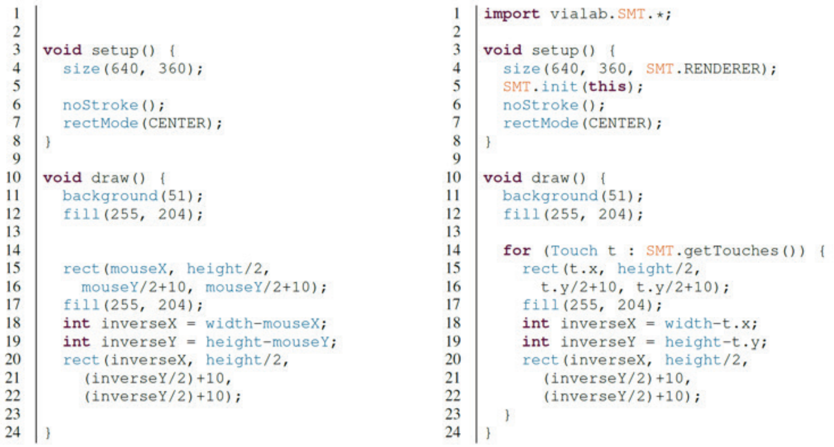
Figure 1. Overview of the architecture of the Simple Multi-Touch Toolkit. The Processing Sketch is written by the student or designer after importing the SMT library, which provides input handling and rendering capabilities.

The Simple Multi-Touch Tookit

Following our design guidelines, the SMT toolkit integrates with the styles of programming supported by Processing. The central construct of SMT is a new display and interaction primitive called the Zone. SMT also provides back-end support for a variety of input devices, handling touch events and providing them to applications using a common Touch construct. The accompanying website offers documentation, including complete JavaDoc and a full suite of tutorials and teaching materials.

Zones

The Zone is the central concept of the toolkit (Figure 1, bottom left). Zones are similar to Windows or Panels from other windowing toolkits, with the important difference that, as graphical primitives, they are not limited to assumptions such a predefined direction/shape/scale or interaction through a single mouse and keyboard. Moreover, they are designed to be understandable without an in-depth understanding of object-oriented programming, messaging, or callback functions. Each zone defines a drawable and touchable artifact in the programmer's sketch. Zones can be customized to accomplish a variety of interface goals. The most important and common modifications, changing how a zone draws and what happens when it is touched, have special support from the toolkit. Zones can be nested, which permits the creation of more complex user interface elements, such as toolbars and menus (which SMT also provides).



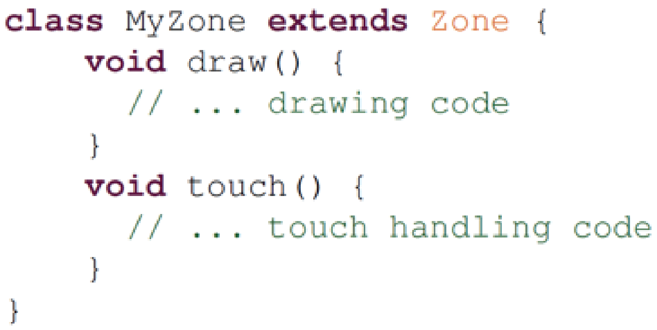
```


void setup() {
  size(640, 360);

  noStroke();
  rectMode(CENTER);
}

void draw() {
  background(51);
  fill(255, 204);


  rect(mouseX, height/2,
    mouseY/2+10, mouseY/2+10);
  fill(255, 204);
  int inverseX = width-mouseX;
  int inverseY = height-mouseY;
  rect(inverseX, height/2,
    (inverseY/2)+10,
    (inverseY/2)+10);

}
```

```
import vialab.SMT.*;

void setup() {
  size(640, 360, SMT.RENDERER);
  SMT.init(this);
  noStroke();
  rectMode(CENTER);
}

void draw() {
  background(51);
  fill(255, 204);

  for (Touch t : SMT.getTouches()) {
    rect(t.x, height/2,
      t.y/2+10, t.y/2+10);
    fill(255, 204);
    int inverseX = width-t.x;
    int inverseY = height-t.y;
    rect(inverseX, height/2,
      (inverseY/2)+10,
      (inverseY/2)+10);
  }
}
```

Figure 2. An example from Processing's website (https://processing.org/examples/mouse2d.html) to demonstrate mouse use (left),converted to support multiple touches using SMT (right).

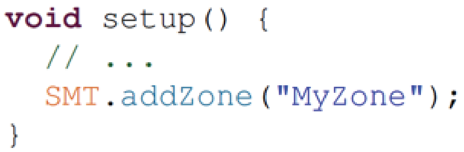
Zone Methods. There are two critical methods that must be implemented for each zone. These are the draw method (adapted from Processing) and the touch method (introduced in SMT). There are two different styles in which these methods can be written—procedurally and using object-oriented programming (OOP). These two styles mimic the approaches taken by

Processing and Java, respectively. We discuss how we incorporated both styles into SMT later in this report. To implement these methods using OOP, the traditional approach of overriding the methods in a class that inherits from the Zone class is used:



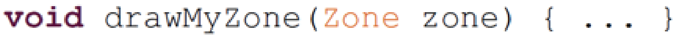
```
class MyZone extends Zone {
    void draw() {
      // ... drawing code
    }
    void touch() {
      // ... touch handling code
    }
}
```

To implement these methods procedurally, one would first create the zone with a string-based name:



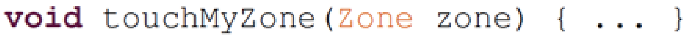
```
void setup() {
  // ...
  SMT.addZone("MyZone");
}
```

And then define a method in the processing sketch by appending the zone's name. For example, to implement the draw method, one would write the method:



```
void drawMyZone(Zone zone) { ... }
```

To implement the touch method for same zone, one would write the method:



```
void touchMyZone(Zone zone) { ... }
```

When both a procedural and object-oriented implementation are detected for the same zone name and method, the procedural one is selected and invoked by the toolkit.

Nesting. An important principle in user interface design is the nesting of elements. SMT supports this principle by permitting zones to be nested in parent-child relationships. This is done by having the child zones inherit their parent's transformation matrix. If the parent is rotated, scaled, or translated, the child will be rotated, scaled, or translated along with it.

Touch Input

SMT supports all the most common desktop touch input devices (Figure 1, right). This includes TUIO devices, Windows Touch, SMART Tables, and Leap Motion. Each of these touch event sources are optional and can be used in any desired combination. Since each of these devices provides events in a different way, they must be unified in some manner. SMT handles this by converting all input into the TUIO protocol. SMT then wraps the underlying TUIO cursor object with a convenient Touch class which provides the user with an abstract handle to touches that is both easy to understand and use. For example, to make any Processing sketch touch-capable, one need only add a few lines of code (Figure 2).

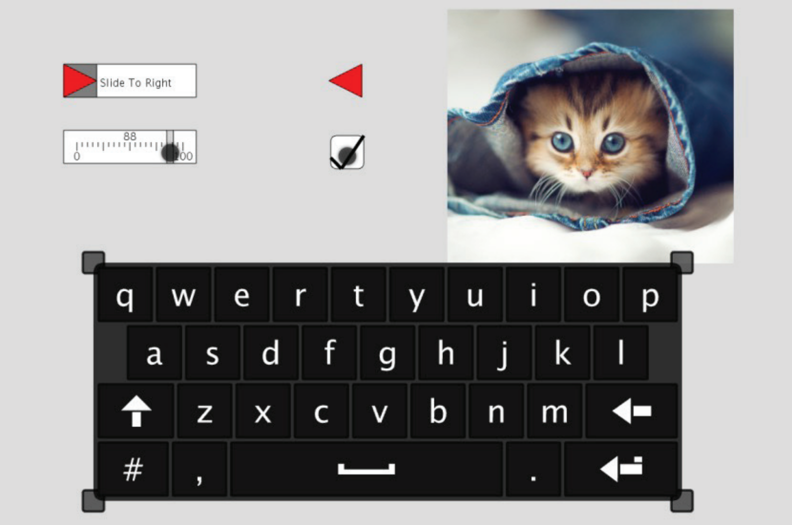


Figure 3. Various interface components provided as Zones in SMT.

While we have designed processing of touch events to closely resemble mouse handling in Processing, we have also provided several techniques for conveniently enabling common multi-touch interaction techniques, such as rotation, translation, and scaling. For example, to implement the common RST (rotate-scale-translate) method on any zone, one would write:

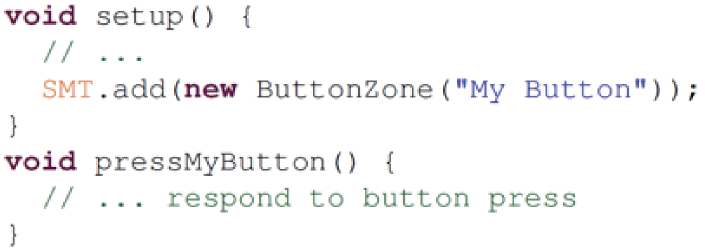
```
void touchMyZone(Zone zone) {
  zone.rst();
}
```

Implementation Details. After conversion into the TUIO format, touches are assigned to zones using standard colour picking. The definition of picking bounds is actually done with a zone method in the same form as the draw and touch methods previously discussed. Special care has been taken in the development of SMT to prevent colour calls and similar erroneous call from being made within this picking method. After touches have been assigned to their zones, a group of methods that correspond to the main types of touch events are invoked. These methods can also be defined in the procedural or object-oriented form. Finally, the touch method is invoked, within which there are a number of predefined standard gestures that can be used, such as drag, pinch, rotate, and scale.

Common Interface Components

In addition to providing support for programmer-drawn zones and low-level touch handling, we provide several common interface components that can be added in the same way as any other zone. For instance, we provide support for tabs (TabZone), buttons (ButtonZone), sliders (SliderZone, SlideRevealZone, PatternUnlockZone), checkboxes (CheckBoxZone), menus (PieMenuZone and LeftPopUpMenuZone), keyboards (KeyboardZone), and many other common interface components (for a total of 21 zones). Figure 3 shows several of these components rendered in an SMT sketch.

Many of these components are made interactive through methods that can again be overridden in a child class (OOP) or directly in the sketch through a named method (procedural). For example:



```
void setup() {
  // ...
  SMT.add(new ButtonZone("My Button"));
}
void pressMyButton() {
  // ... respond to button press
}
```

Development and Debugging Tools

Multi-touch Emulation. Not all development machines necessarily have touch input methods. In order to support the development and testing of SMT sketches on machines lacking such input devices, we implemented a convenient way of emulating multi-touch with just a mouse. The system is fairly simple: the left mouse button emulates a temporary touch, and the right mouse button emulates a touch that lingers. Touches created with the left mouse button will only stay as long as the mouse button is held down. Conversely, touches created with the right mouse button will remain after the mouse button is released. At this point, these lingering touches can either be moved around with the left mouse button, or removed by right-clicking them again. Any number of touches can be created, but only one can be moved at a time with the mouse.

Procedural Programming Warnings. The procedural-style zone methods must follow a fairly specific form in order to be detected and invoked properly. Since mistakes in following this form are easy to make, SMT provides a number of warnings to help guide the user. For example, when a method is detected with one of the zone method prefixes, but the rest of the method name does not match any known zones, it is likely that the user simply misspelled the name of one of their zones, so SMT prints a warning.

Documentation. In this vein, SMT's website covers most of the bases. In addition to recent release information, the website hosts SMT's JavaDocs as well as a full suite of tutorials, examples, and a Processing-style reference page. The tutorials start with the basic concepts, then covers all the important more advanced concepts, including various visual customizations, how to make viewports, and how to transition code from the procedural style to the object-oriented style.

Programming with SMT

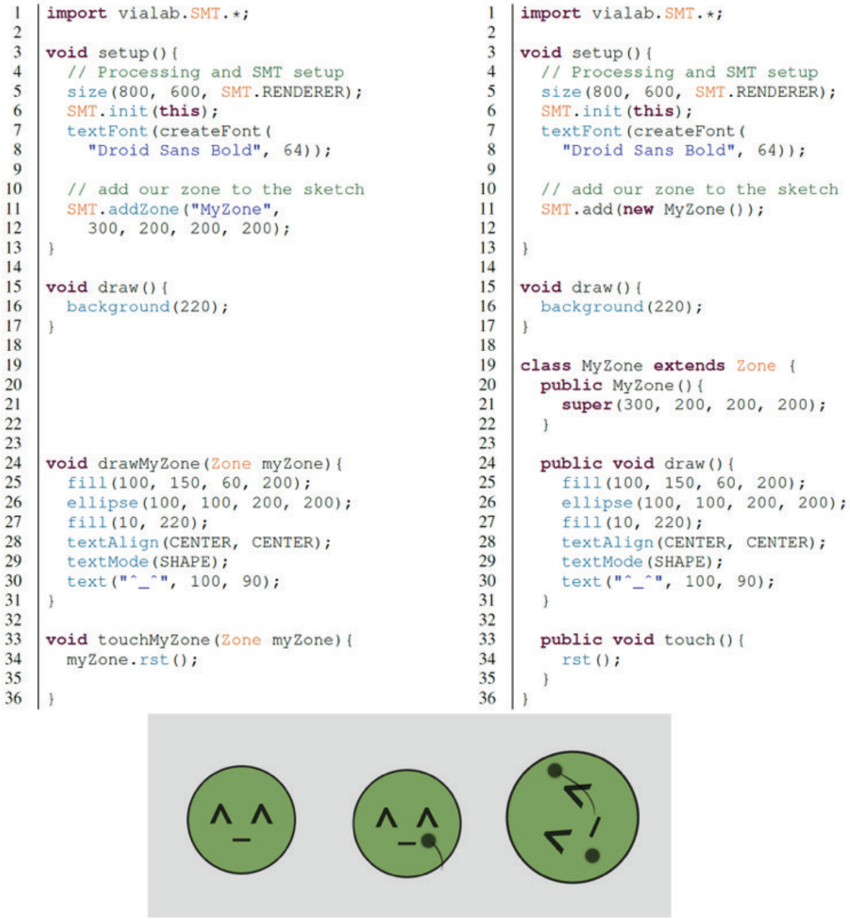
In this section we demonstrate through examples how SMT supports both novice and expert coding styles in a manner which is harmonious with the norms in the Processing programming language. Many of these examples are also available in the tutorials section of the SMT website.

Supporting Different Programming Styles

We support two main styles of development, novice and expert. Statements in each style can be interleaved in the same application, giving maximal flexibility to developers. The novice style is a hybrid of procedural and object-oriented programming (OOP), minimizing use of OOP concepts such as event processing, constructors, and object inheritance. The expert style is standard OOP. In addition, developers may use the Processing IDE (best suited for novices) or whichever development environment they prefer (e.g. Eclipse, best suited for experts). For example, the standard Java statement SMT.add(new Zone("MyZone", 100, 200, 50, 60)); can be rewritten as SMT. addZone("MyZone", 100, 200, 50, 60); in the novice style. Note that due to the constraint that all Processing sketches must extend PApplet ("Processing Applet"), we are unable to make methods available to developers without requiring the SMT. prefix.

In the following example we demonstrate how to create a simple, highly responsive application which renders a custom "happy face" Zone that supports multi-touch rotate, translate, and scale. The code is written using the Processing hybrid procedural/object-oriented style for novices (Figure 4, left) and using traditional object-oriented style for experts (Figure 4, right).

In both examples, the sketch is initialized with the import statement from SMT, which is provided automatically by Processing when the library is included in the IDE. The setup method is common to all Processing sketches. In SMT it must include a call specifying the initial window size and selecting the SMT renderer, which inserts SMT zone management into the Processing rendering queue. SMT is then initialized. In this example, a single zone called "MyZone" is added to the sketch. In the novice style, the zone is added by naming it in the addZone call, and subsequent draw and touch methods reference the specified name using reflection. That is, the novice can create a method called drawMyZone and it will be invoked appropriately to render the zone. Some people, especially those used to Java and object-oriented programming, can find SMT's reflection-invoked methods non-intuitive. Thus, in the expert style, an inner class called MyZone is created using the add method and has its own draw and touch methods.



```
import vialab.SMT.*;

void setup(){
  // Processing and SMT setup
  size(800, 600, SMT.RENDERER);
  SMT.init(this);
  textFont(createFont(
    "Droid Sans Bold", 64));

  // add our zone to the sketch
  SMT.addZone("MyZone",
    300, 200, 200, 200);
}

void draw(){
  background(220);
}






void drawMyZone(Zone myZone){
  fill(100, 150, 60, 200);
  ellipse(100, 100, 200, 200);
  fill(10, 220);
  textAlign(CENTER, CENTER);
  textMode(SHAPE);
  text("^_^", 100, 90);
}

void touchMyZone(Zone myZone){
  myZone.rst();

}
```

```
import vialab.SMT.*;

void setup(){
  // Processing and SMT setup
  size(800, 600, SMT.RENDERER);
  SMT.init(this);
  textFont(createFont(
    "Droid Sans Bold", 64));

  // add our zone to the sketch
  SMT.add(new MyZone());

}

void draw(){
  background(220);
}

class MyZone extends Zone {
  public MyZone(){
    super(300, 200, 200, 200);
  }

  public void draw(){
    fill(100, 150, 60, 200);
    ellipse(100, 100, 200, 200);
    fill(10, 220);
    textAlign(CENTER, CENTER);
    textMode(SHAPE);
    text("^_^", 100, 90);
  }

  public void touch(){
    rst();
  }
}
```

Figure 4. Code for creating the “happy face” example, using novice (i.e., more procedural) approach on left, and expert (OOP) approach on right, and the resulting sketch (bottom).

Examples

In this section, we introduce Processing sketches built with the SMT toolkit. To support a learning-by-example style of learning, as requested by students in the first in-class deployment, the SMT library in Processing comes with more than 25 example sketches which illustrate each Zone type and method. In addition, we provide 4 sketches corresponding to online tutorials, and 12 fully realized demonstration applications, including a photo organizing application, a checkers game, a login screen, and a table hockey game. We will discuss the table hockey example below.

The table hockey demonstration application was made by an intern within their first week working with SMT. The 311-line sketch produces a simple two-player table hockey game, designed to be played on a multi-touch table display. Each of the pucks are SMT Zones. All the pucks could theoretically be handled at the same time, as long as the touch devices being used can handle that many touches. Pucks can be tossed across the game board at variable velocities. To demonstrate Zone manipulation, a 160-line custom physics engine manages collisions between pucks and with board boundaries, but this could also be accomplished with a third party physics library.

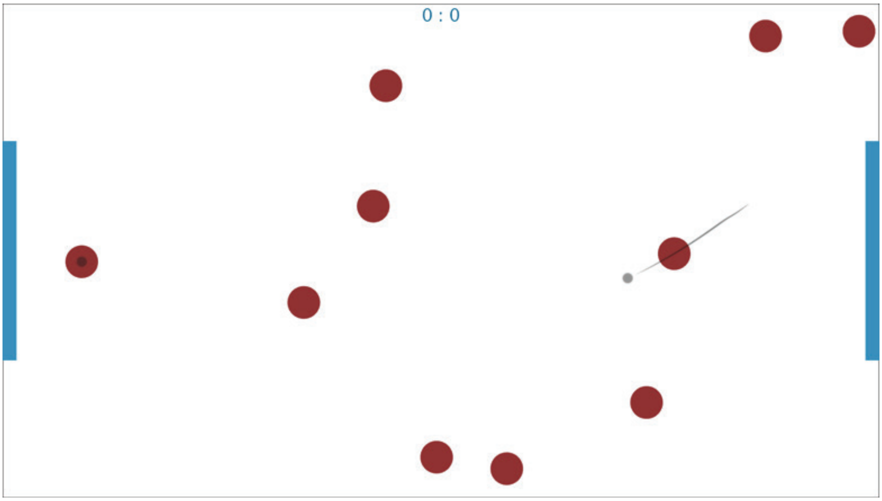


Figure 5. A table hockey game written with SMT.

Initial Evaluation

After the initial phase of development on SMT, we deployed it in two HCI classes for students to use in laboratory activities and in the development of term-long group projects. We studied the deployment of the toolkit through student feedback surveys and analysis of completed student projects. The goals of the study were to investigate whether SMT was useful for prototyping multi-touch applications, accessible to novices, and powerful for experts. In particular, we sought to understand the speed of the development cycle and whether students became comfortable with rapid prototyping (sketching) using SMT during their brief exposure to it.

Method

Participants were recruited from two HCI courses at two separate universities. At one of the universities, the HCI course was being taught to mainly management sciences students who had relatively little experience with programming ("novices"). At the other university, the students were fourth year computer science and software engineering students ("experts"). The idea behind this approach was to show separately how both novice and intermediate programmers responded to SMT.

After their first lab session using SMT, the students of these courses were asked to fill out a questionnaire on the toolkit. After their last lab session using SMT (6 weeks later), they were asked fill out the same questionnaire again. Students were invited to grant permission to use the code and images of their project for the purpose of analysis of the toolkit. The questionnaire was based on "A Cognitive Dimensions Questionnaire" (Blackwell and Green, 2007), a standardized framework for analyzing the usability of information artifacts, in particular software systems (Blackwell, 2015). All data was collected by a third party and retained until after final grades were submitted to ensure separation of the study and the course outcomes.

We received a total of 22 responses to the first round deployment of the questionnaire, but only 1 response to the second round. At one of the universities, no students completed the questionnaire. Thus, all responses we received were from the "experts" group. This made the intended comparison between the four sets of responses infeasible. Results below refer only to the first administration at one university. Four (out of 14) groups in the computer science class gave unanimous permission to evaluate their projects for the purposes of this study.

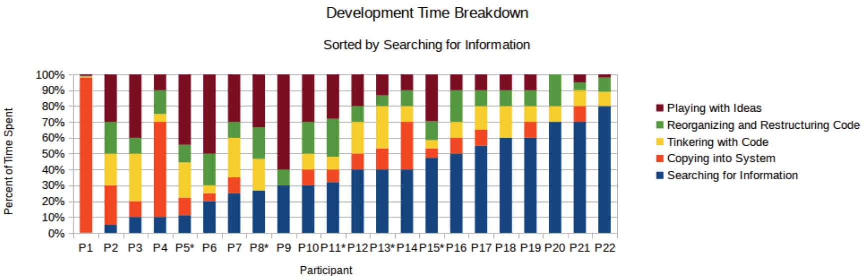


Figure 6. Breakdown of time spent, sorted by time spent searching for information.

Questionnaire Results

Below we discuss the results of the three sections of the questionnaire: time using SMT, questions about usability of the API, and suggestions for improvement.

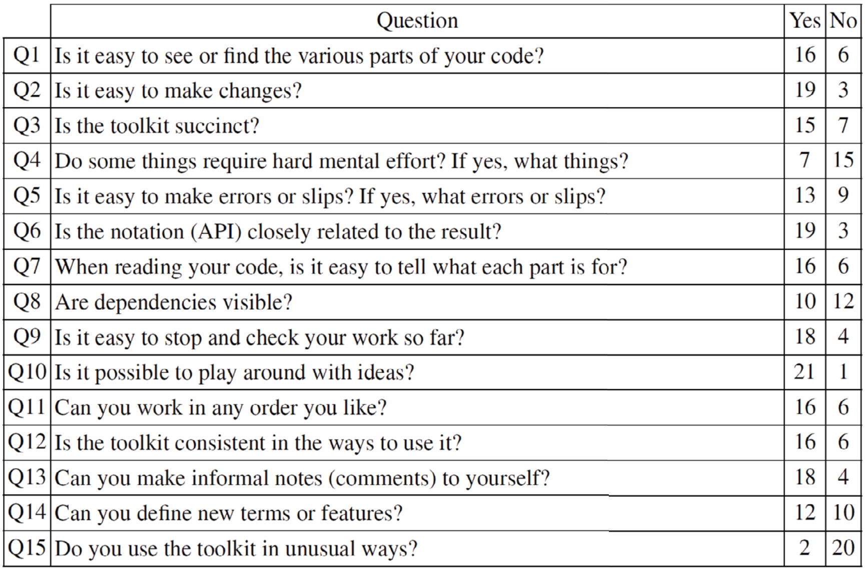
Time. Out of the 22 participants who completed the questionnaire, 19 had only spent 1-2 hours working with the toolkit. The other three participants all had spent 3-5 hours working with the toolkit.

A series of questions asked about fraction of time spent on each type of development activity that can occur while using a notation. It was intended, but not enforced, that the sum of each response would be 100%. Figure 6 shows how each participant estimated the time they spent on the various types of development activities they undertook while working with SMT. The participants are sorted by their answer to the first question. Participants whose responses did not add up to 100% have been normalized and are marked with asterisks.

The results show a marked variance in activities undertaken with the toolkit.

Given that these results come from after only a short time using SMT, it makes sense that, for many participants, searching for information and copying code examples into the system were dominant tasks. For eight participants, more than half the time was spent on the core prototyping activities of tinkering with code and playing with ideas.

Questions about API Usability. Table 1 shows the response breakdown for a series of questions related to the various features of SMT. First impressions of the students indicate that they thought SMT was easy to use (Q1, Q2, Q4), succinct (Q3), predictable and transparent (Q6, Q7, Q8, Q12), and flexible (Q9, Q10, Q11, Q13). Comments included "The concept of zones and sub zones does work well and provides an easy hierarchy to follow" (Q6) and "[It is] easy to have a short development cycle with save and run" (Q9). There is evidence that some students found it easy to make errors or slips (Q5), indicating our error checking and compile-time warnings could be improved. In particular, several participants lamented the lack of a standard debugger in Processing. Also, students indicated that they did not use the toolkit in new and different ways (Q14, Q15), which was likely due to the brevity of their experience with it.



	Question	Yes	No
Q1	Is it easy to see or find the various parts of your code?	16	6
Q2	Is it easy to make changes?	19	3
Q3	Is the toolkit succinct?	15	7
Q4	Do some things require hard mental effort? If yes, what things?	7	15
Q5	Is it easy to make errors or slips? If yes, what errors or slips?	13	9
Q6	Is the notation (API) closely related to the result?	19	3
Q7	When reading your code, is it easy to tell what each part is for?	16	6
Q8	Are dependencies visible?	10	12
Q9	Is it easy to stop and check your work so far?	18	4
Q10	Is it possible to play around with ideas?	21	1
Q11	Can you work in any order you like?	16	6
Q12	Is the toolkit consistent in the ways to use it?	16	6
Q13	Can you make informal notes (comments) to yourself?	18	4
Q14	Can you define new terms or features?	12	10
Q15	Do you use the toolkit in unusual ways?	2	20

Table 1. Responses to the questions asked in our questionnaire.

Suggestions. Twelve participants responded with specific suggestions for improvement of SMT. Seven of the responses in some way requested better documentation, often specifically requesting example-based documentation. Two of the responses recommended changing SMT to better follow object-oriented design. Two responses requested features from more a complete IDEs like Eclipse (which was related to the Processing environment and not SMT). Two responses were generally positive comments, e.g. “nice and adequately built toolkit”. One response was a specific feature request for improvements to the zone rotation process.

Student Projects. The projects the students completed as part of their course mainly involved the design of a prototype user interface. Various methods of design were taught and encouraged, including sketches and storyboards, paper prototypes, and software prototypes (created in Processing). The software prototypes used SMT to manage the touch interactions in the user interface. Prototypes developed with SMT ranged across a wide variety of topics, including mobile workout coaching for a phone-sized device and transit planning for a wall display, demonstrating the flexibility of SMT across domains and hardware.

Discussion. There were pragmatic challenges in running a classroom-based study in our own classrooms. One issue was that we were not granted approval under research ethics to incentivize our participants in any way, including through means unrelated to the course, such as monetary remuneration. In addition, we did not allocate class time for the administration of the study. Thus, requesting students to complete an optional and anonymous questionnaire on their own time with no reward contributed to our low response rate. We also hypothesize that the specific design of some of the questions, based on the Cognitive Dimensions model, may have intimidated students due to unfamiliar language referring to “notations”.

The suggestions received in the questionnaire likely reflect participants’ enrollment in a traditional computer-science program: they expected a powerful IDE and object-oriented style. To respond to these, we improved the documentation and the curriculum to explicitly help advanced students work within Eclipse in an object-oriented style if they chose to do so. We improved SMT and its documentation based on student feedback and several months of refinement with our users through the open source deployment before offering one of the courses again, with a revised study, as discussed in the next section.

Follow-up Evaluation

Based on experiences with the first use of SMT in teaching multi-touch for human-computer interaction, we made many improvements to the toolkit, including extensive documentation, online tutorials, and examples which illustrated each Zone type and method. Advanced examples and tutorials illustrated functionalities such as custom picking and viewports. In addition, we simplified our study method and conducted a second round of evaluation at one university (the second university was not offering the course).

Method

Participants were recruited from a fourth year computer science course (the same course for which results of the first study are reported). At the end of the semester, after two laboratory activities using SMT, and after using SMT to create prototypes for their term project, a questionnaire which focused on ease of learning SMT was administered. Demographic data on years of experience and self-rated programming skill was collected.

Again, students were invited to grant permission to use the code and images from their final group project, with optional acknowledgement to them, for the purpose of analysis of the toolkit. We received a total of 18 responses to the questionnaire and 7 groups provided unanimous permission to evaluate their projects for the purposes of this study.

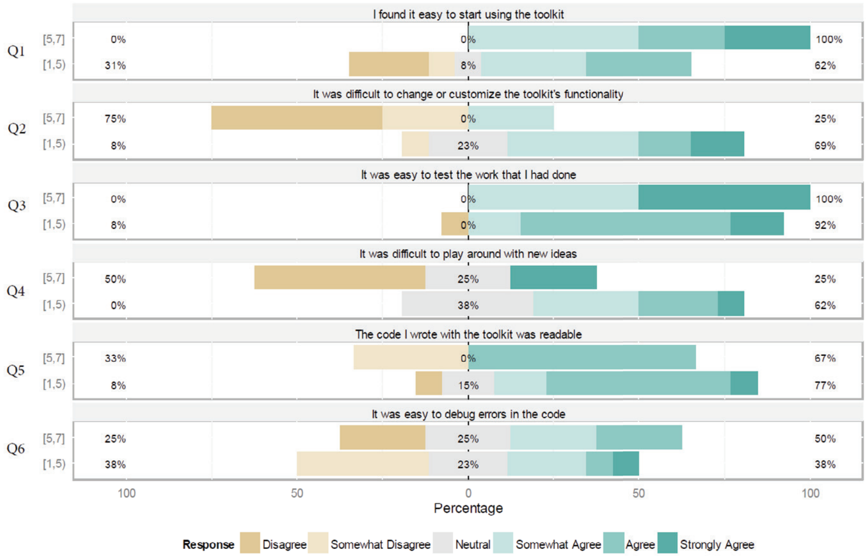


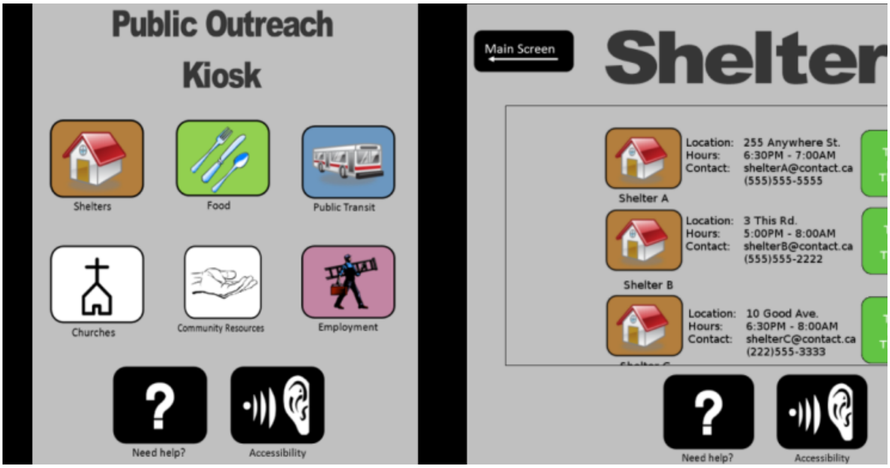
Figure 7. Ease-of-coding questionnaire results from follow-up evaluation. Participants are grouped by skill level as indicated on the left.

Results and Discussion

Students indicated an average of 19 hours experience with SMT (min: 4, max: 80). The large spread is expected as they were using SMT as part of a large group project where greater coding responsibilities may have been delegated to some students. Our questionnaire contained a series of questions investigating how long it took to learn the toolkit (from one hour to several months). All but three students indicated "agree" or "strongly agree" with feeling comfortable using the toolkit after a few hours, and all students but one were comfortable after a day. The one remaining student (skill level=4, hours of use=40) indicated "neutral" for all time periods. In the analysis which follows, we divided students into two groups: novice (self-rated 1–4, n=8) and expert (self-rated 5–7, n=9). A summary of questionnaire results by skill level is found in Figure 7.

Q1 indicated that most students of all skill levels found it easy to start using the toolkit. Q2 shows a split, with novices expressing more challenge with customizing and changing the toolkits functionality. This is not concerning as 75% of experts did not find it difficult, and this is an advanced function which normally would not be used by novices. Q3 showed that all students found it easy to test their work. Q4 again reveals a split between 25% of experts who had some difficulty playing with new ideas, and 64% of novices who had some difficulty. Both experts and novices found the code readable (Q5). The results on ease of debugging were similar between groups, with around 30% indicating some difficulty debugging. 14 students provided specific suggestions for improvement. Of these, 6 corresponded to the Processing IDE (e.g. desire for code completion). 8 comments related to feature and improvement suggestions for SMT, with 7 students suggesting further improvements to the online documentation, including coded examples for every method.

Students created a wide variety of prototype applications for multi-touch table and wall displays, including transit planning, digital board games, personal health monitoring, and a prototype public display providing access to outreach services for the homeless, pictured in Figure 8.



Location: 255 Anywhere St.
Hours: 6:30PM - 7:00AM
Contact: shelterA@contact.ca (555)555-5555

Location: 3 This Rd.
Hours: 5:00PM - 8:00AM
Contact: shelterB@contact.ca (555)555-2222

Location: 10 Good Ave.
Hours: 6:30PM - 8:00AM
Contact: shelterC@contact.ca (222)555-3333

Figure 8. Example screens from a student term project created with SMT, showing a public kiosk interface to provide information about services for the homeless.

Overall, the results of our questionnaire indicate student satisfaction with SMT across skill levels. Concerns around ease of debugging likely relate to the use of runtime warnings (e.g. if a student creates a zone called "OKButtonZone" without the "drawOKButtonZone" method, a warning is generated at run-time instead of compile time). This is due to the use of reflection and the capabilities of the Processing IDE. Students self-rating as novice did also indicate some difficulty playing with new ideas, revealing that for this group, further improvements to code simplicity and training materials are needed. We leave this for future research.

Real-World Use

SMT has been developed and actively supported for two years, during which it has been used in two human-computer interaction and interface design courses at two different universities across multiple semesters (for a total of four classes) to help students learn to develop medium fidelity prototypes of their multi-touch designs. Our toolkit has also been used for the development of research prototypes in at least six graduate student projects.

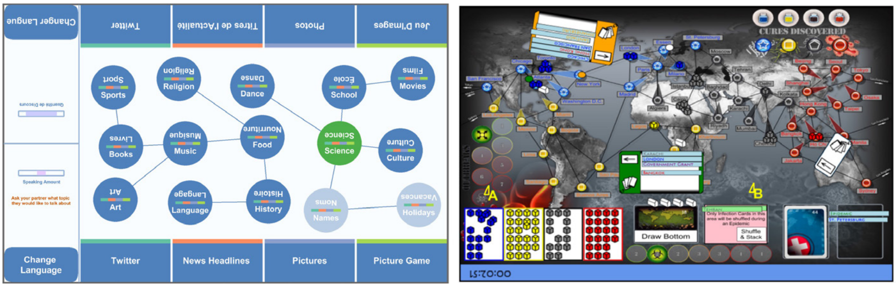


Figure 9. Graduate research projects TandemTable (left) and, Pandemic (right), created with SMT.

Feedback from the use in these real-world projects has been mostly positive, with students able to create interesting and sophisticated multi-touch designs, while not requiring significant in-class time to learn how to program. Students were instead able to focus their learning on design methods and evaluation techniques. Graduate students commented on the ease with which they could rapidly prototype, mentioning how in most cases the development took far less time than their previous experience with Application Programming Interfaces such as Windows Presentation Framework and C#.

We feature two graduate student research projects using SMT in Figure 9. The first is an assistive application for the tandem language learning method. It was developed in order to study how interactive tables can be used to augment the language learning process (Paluka and Collins, 2015). The second is a multi-touch implementation of the Pandemic board game (Chang et al., 2014). It was developed in order to study how knowledge of past game events may change people's strategies and behaviors while playing turn-based games. SMT has also been used to develop and publish a multi-touch visualization application by a research group not affiliated with the SMT authors (Dai et al., 2015). While the graphical rendering capabilities of the Processing environment were helpful to these projects, specific features of SMT were also critical to their success.

SMT is a free and open source library. Its codebase is currently hosted on GitHub at http://github.com/vialab/smt. Being hosted in a public and easily accessible venue holds many benefits to a toolkit. One of these benefits is the feedback and input from people from all over the world, whom we would never otherwise have been given the opportunity to interact with. SMT's GitHub page regularly receives 60+ unique visitors per week. Not including the authors, SMT's GitHub page has been followed by 26 people, and starred by 31. In addition to this, we have received and dealt with many bug reports and feature requests from users around the world.

Conclusion and Future Work

We have created the Simple Multi-touch Toolkit, which is a simplified software toolkit for the Processing programming language. It is designed to reduce the amount of knowledge required and the complexity involved in programming multi-touch applications. Although our toolkit simplifies multi-touch programming, seasoned developers are afforded many advanced additional features, such as access to more low-level data structures and many customization features. By combining SMT with a mouse-based multi-touch emulator, users are able to develop their applications on machines without interactive surfaces, which then run seamlessly on touch-enabled surfaces. Cross-platform development is enabled through the integration of multiple input bridges and native TUIO support. SMT has been successfully used at multiple universities for developing research prototypes as well as full-fledged applications. The toolkit has also been used in courses at these universities for teaching concepts and skills related to HCI. Our web resources include tutorials and teaching materials for using SMT in the classroom and we will continue to support its use in teaching and research environments. In the future, we plan to incorporate additional support for more complex multi-touch gestures, to add automatic layout algorithms for creating interfaces with multiple Zones, and deploy SMT for Android, which is currently in private alpha development stage.