Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TOUCHID Toolkit

Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring, Saul Greenberg

Department of Computer Science
University of Calgary, 2500 University Drive NW, Calgary, AB, T2N 1N4, Canada
[nicolai.marquardt, jlkiemer, david.ledo, sebastian.boring, saul.greenberg]@ucalgary.ca

ABSTRACT

Recent work in multi-touch tabletop interaction introduced many novel techniques that let people manipulate digital content through touch. Yet most only detect touch blobs. This ignores richer interactions that would be possible if we could identify (1) which hand, (2) which part of the hand, (3) which side of the hand, and (4) which person is actually touching the surface. *Fiduciary-tagged gloves* were previously introduced as a simple but reliable technique for providing this information. The problem is that its low-level programming model hinders the way developers could rapidly explore new kinds of user- and handpart-aware interactions. We contribute the TOUCHID toolkit to solve this problem. It allows rapid prototyping of expressive multi-touch interactions that exploit the aforementioned characteristics of touch input. TOUCHID provides an easy-to-use event-driven API. It also provides higher-level tools that facilitate development: a *glove configurator* to rapidly associate particular glove parts to handparts; and a *posture configurator* and *gesture configurator* for registering new hand postures and gestures for the toolkit to recognize. We illustrate TOUCHID's expressiveness by showing how we developed a suite of techniques (which we consider a secondary contribution) that exploits knowledge of which handpart is touching the surface.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Surfaces, tabletop, interaction, touch, postures, gestures, gloves, fiduciary tags, multi user, toolkit

INTRODUCTION

The arrival of interactive multi-touch tabletop systems heralded the development of many novel and powerful ways for people to manipulate digital content (e.g., [6,14,15,19]). Yet most technologies cannot sense what is causing the touch, i.e., they are unable to differentiate between the touches of *two different people*, or between *different hands,* or between *different parts of the hand* touching the surface. This is a lost opportunity, as this knowledge could be the basis of even more powerful interaction techniques.

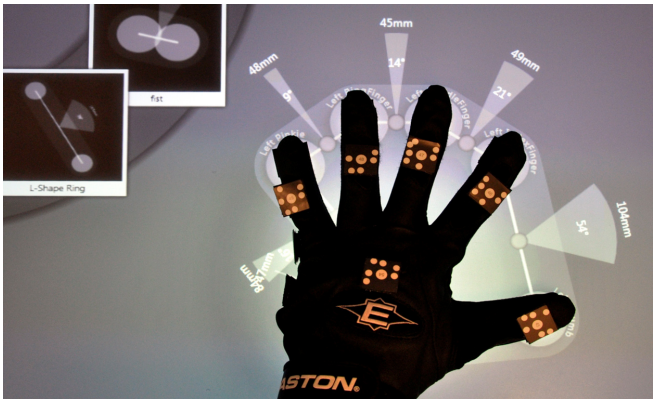

Figure 1. Using the TouchID *Posture Configurator* to train a new posture with the fiduciary-tagged glove.

Still, a few technologies do allow such differentiation, albeit in limited ways. The *DiamondTouch* surface identifies which person is touching [6,40], but little else. Muscle-sensing identifies fingers [2], and computer-vision approaches begin to infer similar information, though not robustly (e.g., [4,32]). The *Fiduciary-Tagged Glove* [24] – which we use in our own work – is perhaps the most promising. While a wearable, it offers an inexpensive, simple, yet robust tracking method for experimental development. It serves as a good stand-in until non-encumbered technologies are realistically available. As seen in Figure 1, fiduciary tags (printed labels) are glued to key handparts of the glove. When used by a person over a fiduciary tag-aware surface (e.g., the Microsoft Surface), the location and orientation of these tags can be tracked. Because the software associates tags to particular handparts, it can return precise information about one or more parts of the hand in contact with the touch surface.

The problem is that the fiduciary-tagged glove still requires low-level programming. The programmer has to track all individual tags, calculate the spatial, motion and orientation relations between tags, and infer posture and gesture information from those relationships. While certainly possible [24], this complexity limits the number of programmers willing to use it, and demands more development time to actually prototype rich interaction techniques.

Cite as:
Marquardt, N., Kiemer, J., Ledo, D., Boring, S., Greenberg, S. (2011) Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TouchID Toolkit.
Research Report 2011-1004-16, Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4, July.

Our goal is to mitigate this problem by developing the TOUCHID (Touch IDentification) toolkit that provides the programmer with knowledge of the person, hand, and handpart touching the surface. Our expectation is that by making this information easily available, the programmer can rapidly develop interaction techniques that leverage this information. Specifically, we contribute:

- The TOUCHID toolkit and API as a test bed for rapid exploration of person, hand, and handpart-aware tabletop interaction techniques.
- Easy-to-use tools for registering both hand postures (Figure 1) and gestures for recognition by the toolkit.
- An evaluation-in-practice of this toolkit, where we illustrate how we used it to rapidly develop a suite of expressive interaction techniques.
- The interaction techniques are themselves a secondary contribution, for they seed a design space of expressive multi-handpart, multi-hand and multi-user interaction techniques that could inspire future tabletop applications.

We briefly review related work in the area of touch recognition and development tools, followed by an explanation of the fiduciary-tagged gloves. We then introduce the TOUCHID toolkit: we begin with its three high level tools and then detail the toolkit's API. Subsequently, we introduce a set of novel interaction techniques that leverages the knowledge provided by TOUCHID about individual touches.

BACKGROUND AND RELATED WORK

Multi-touch surfaces have been based on a variety of technologies, e.g., [8,14,21,40], each with their own strengths and weaknesses. Touch has also been exploited in different ways, ranging from touch alone, to multi-fingers [40], to hand shape [9] and hand postures [10], and to gestures [38]. Because these and others are now familiar in the literature, we concentrate our background on two factors: previously developed techniques that identify particular users or handparts, and development tools for prototyping multi-touch applications on surfaces.

Identifying Users and Handparts Touching the Surface

Our basic assumption is that more powerful interaction techniques can be built if a tabletop system knows which person, hand, or part of the hand is touching the surface. Some earlier work explored such identification; most notably *computer vision approaches*, *specialized hardware,* and *glove-based tracking*.

In *computer vision*, some techniques recognize the position of hands and identifying fingers [23]. Another approach uses distance, orientation, and movement information of touch blobs to identify fingers and distinguish if they are from the same or different hands of a person [4]. Schmidt's HandsDown [32] system derives user information from matching features of a person's hand outline to the ones stored in a database. Later, this technique was applied to allow personalized workspaces and lenses for tabletop interaction [33]. Increasing the reliability and accuracy of the computer vision recognition remains a challenge for all of these systems.

Specialized hardware also distinguishes between handparts and users. For example, the MERL DiamondTouch surface identifies which of up to four people are touching the surface [6]. It works through capacitive coupling, where a low current running through a person's body is detected by the surface allowing the mapping of each touch contact to a particular person. DiamondTouch is notable as a development platform: its unique ability to differentiate which person was touching the surface was the basis of many user-aware tabletop explorations, including cooperative gestures [26], shared *vs.* replicated controls [27], and multi-user coordination [28]. Other hardware approaches distinguish handparts. For example, an EMG muscle sensing armband identifies a person's fingers touching a surface [2], while fingerprint recognition could provide similarly precise touch information and user identification [17].

Glove-based techniques, typically found within virtual reality research, track augmented gloves in 3D space. Methods include optical tracking, printed markers, silhouette analysis, magnetic tracking, optical fibres, and so on [35]. Returned information was exploited in a variety of ways. For example, gestures made by the tracked glove in 3D allowed a person to handle virtual objects in that augmented space [3], and to drive gestural interaction in an augmented reality immersive environment [1]. Other techniques distinguish a user's hands by using coloured glove patterns to discriminate parts of the hand as well as its posture in 3D space [36], or through fiduciary markers attached to gloves [3]. Within the area of surface interaction, Marquardt et al. [24] produced the previously mentioned fiduciary-tagged glove, where tags were tracked by a Microsoft Surface.

Most prior work suffers in some regards. Some require expensive or custom hardware, some are not particularly robust, and some provide only a subset of the desired information. We decided to use Marquardt et al.'s fiduciary-tracked glove as the TOUCHID toolkit initial sensing device. The glove is cheap and easily crafted. It returns accurate information identifying key handparts touching a surface, and for differentiating between hands and users (details are explained shortly) [24]. However, we stress that the TOUCHID toolkit could incorporate other sensing technologies as well.

Development Tools for Creating Surface Application

While the above technologies provide low level information about a touch and what caused it, the complexity of accessing and programming with that information often places them out of reach to all but a few developers. Consequently, researchers have built toolkits in order to facilitate the development of tabletop applications. The idea is that software engineers can then focus on exploring novel interactions instead of low-level programming challenges [13].

LightTracker [12] and ReacTIVision [19] simplify access to computer vision, filters, and calibration methods often

necessary when using tabletop hardware. TUIO [18], originally built for the Reactable hardware, now functions as a universal low-level protocol to transmit and process touch point information. TouchLib [7] introduced a layered architecture allowing multi-touch development independent of the actual underlying tabletop hardware used (i.e., hardware is decoupled from the developer API). Similarly, PyMT [16] allows cross platform development; but further lowers the threshold for rapid prototyping by using the dynamic language Python. Our TOUCHID toolkit extends this previous work by providing complementary information about which exact part of the hand, which hand, and which person is actually causing a particular touch event.

A few toolkits integrate the identification of the person touching the interactive surface. DiamondSpin [34], built around the DiamondTouch hardware [5,6], supports the development of orientation-aware multi-user applications. idWidgets [31] are user-aware touch widgets, while IdenTTop [30] added a framework approach to identity-enabled multi-user applications, also through a set of widgets. As we will see, our TOUCHID toolkit also deeply integrates user identification. We differ in the way that user information is combined with precise information about the actual handparts touching the surface.

THE FIDUCIARY TAGGED GLOVE

As mentioned earlier, TOUCHID is based upon fiduciary-tagged gloves [24], as these produce precise (albeit low-level) information about handparts touching an interactive surface. We chose these gloves as they differ from earlier work through its simple and inexpensive design, yet reliable identification of handparts in contact with the surface.

Figure 2. The fiduciary-tagged glove.

The current glove design works as follows. Fiduciary tags (infrared reflective markers with a unique 8-bit identification pattern) are glued to the glove on positions representing key parts of a person's hand: fingertips, knuckles, side of the hand, palm, etc. (Figure 2). Once any of the tagged handparts touch the tabletop, a Microsoft Surface [25] accurately recognizes their positions and orientations. A simple one-time calibration process associates the tags with their position of the hand (details are described below). Overall, the design of these gloves is simple, reasonably cheap, and – through the tracking of the Microsoft Surface – accurate and reliable. Even though this approach requires people to wear one or two gloves when interacting with the tabletop application, it is nevertheless a robust technique that allows the exploration of novel interaction techniques that leverage hand-part identification until more unencumbered methods are developed.

Yet the design and exploration of novel interaction techniques using the glove's information is still non-trivial. A developer has to perform a series of *low-level tasks*: tracking individual markers touching the surface, looking up user identification for each of the tags, and recognizing gestures from the tag's movements over time. Furthermore, posture recognition is tedious: the programmer has to track simultaneously recognized markers, and infer postures by comparing their relative distance and orientation. Collectively, these present a threshold preventing rapid and iterative exploration of novel interactions [29].

THE TOUCHID TOOLKIT

We developed the TOUCHID toolkit to facilitate the development of multi-touch interaction techniques leveraging knowledge of user, hand, and hand-part identification. A key part of the toolkit is an easy-to-use event driven API (application programming interface) that makes the information about touches, handparts, and users easily accessible to application developers. It also includes a set of high level tools to configure gloves, and to train new hand postures and gestures. In the following, we detail all essential parts of the toolkit.

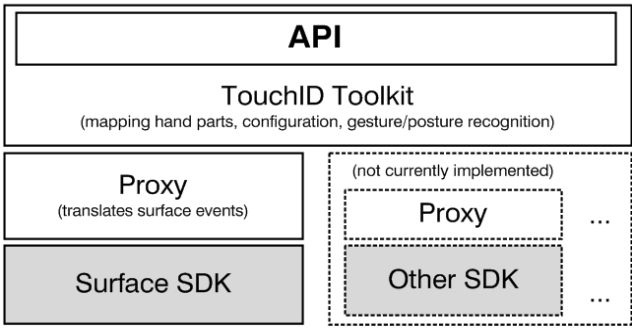

Figure 3. The TouchID toolkit layered architecture.

Architecture

TOUCHID toolkit's architecture is layered to allow for different technologies to be substituted as they become available (see Fig. 3). Its bottom layer accesses the actual input events (e.g., a tag's id, location, and orientation) from the underlying hardware. The next layer (the *proxy*) translates these events into unified events that can be used by the toolkit's API. While we currently rely on the Microsoft Surface, other hardware (e.g., another fiduciary marker tracking system such as [19], or a fingerprint recognizer) providing similar capabilities could be substituted by re-

Figure 4. The glove configurator tool (similar to [18]).

writing this layer. The next higher level is responsible for associating the events with users and handparts. This layer also includes the posture/gesture recognition engine and corresponding configuration tools. The top layer is the actual API seen by the developer. While the lower layers are normally not needed, the developer can – if desired – access the raw information held by those layers (e.g., the tag's id if applicable).

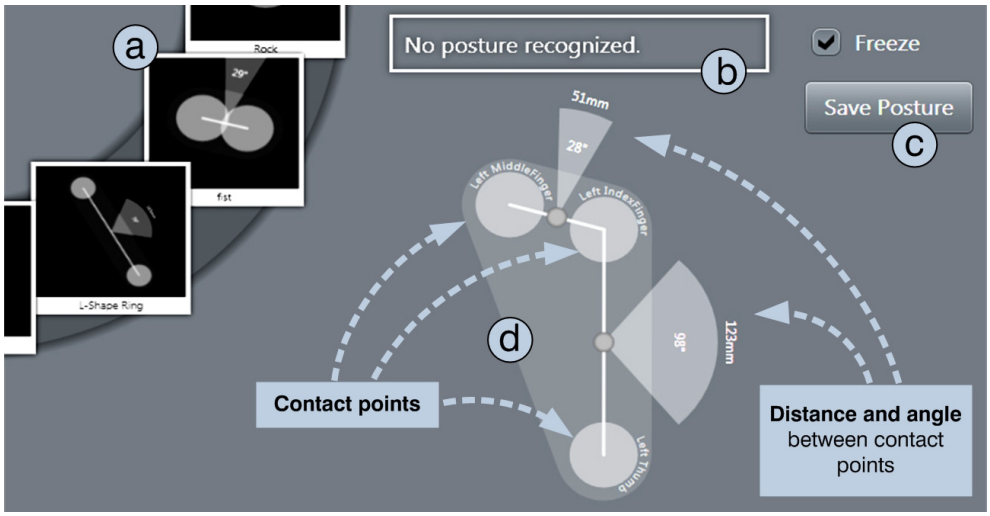

Figure 5. The posture configurator tool: (a) saved postures, (b) currently recognized posture, (c) controls, (d) contact points of handparts touching the surface, and distance/angle between.

Setup and Glove Configuration

A developer installs the TOUCHID toolkit[1] onto a standard PC attached to a Microsoft Surface. The toolkit setup installs all required tools, the developer library with the API, templates, examples, and documentation. Next, the developer starts the toolkit's glove configurator tool (Figure 4) to register each glove they built. This is done only once. This tool shows images of a hand in 3 orientations. The person simply places the three sides of the gloves on the corresponding image, and each tag gets mapped to its corresponding handpart. Each glove is assigned to a particular person. This configuration process can then be repeated for left and right hand gloves of that person, as well as for each additional person. The toolkit and gloves are now ready to use.

This configuration tool differs from previous work [24] in two ways. First, it is tightly integrated into the toolkit so that all configuration files (saved as XML files) are stored in a *central repository* and are accessible by every other TOUCHID tool and its API. Second, it is extended through a compensation mechanism in order to support differently sized gloves (and thus differing distances between the tags). The configuration tool measures the distances between key pairs (e.g., finger tips to palm) and saves a compensation factor associated to these pairs. This compensation is done implicitly by the toolkit, and does not require any additional intervention by the developer.

Posture Training and Recognition

Hand postures – such as a flat hand placed on the surface or an L-shaped posture formed with thumb and index finger – can be an expressive and powerful form of input for tabletop applications. Here, we introduce TOUCHID's Posture Configuration tool and underlying algorithm that lets a developer train the system to seamlessly recognize new postures. A later section will describe the toolkit's API.

[1] Available as open-source download at http://www.[anonymous].org. While the current version is restricted to a Microsoft surface, it can be modified to work with any surface that recognizes fiduciary markers.

Template-based Matching Algorithm

Our posture recognition works through a template-based matching algorithm [22]. For each posture, the toolkit saves one or more templates describing the configuration of information listed below.

- **Number and identification** of contact points (i.e., which handparts are included in the posture, such as "right index finger", "left knuckle of thumb", etc.)
- **The geometry** between all identified contact points captured as **distance** (in mm) and **angle** (in degrees) measures. Because exact geometry matches are unlikely, it also includes a **tolerance threshold** that indicates how much a geometry can deviate from the template and still be recognized as a given posture. A posture template can also be set to ignore the angle and/or distance.

During application runtime, the template-matching algorithm compares the currently recognized input to these saved templates. If the recognized input matches any of the saved templates within the configured tolerance range (e.g., such as 10%), the toolkit notifies the application about the recognized posture. While other posture recognition algorithms could be substituted, e.g., training of neural networks or using hidden markov models [22], our simple template matching algorithm is, in practice, reliable and powerful enough for our purposes of recognizing and differentiating between a variety of postures.

Posture Configuration Tool

The posture configuration tool, illustrated in Figure 5, allows a developer to (optionally) train TOUCHID to recognize new hand postures. In Figure 5, the upper left corner (a) displays thumbnail images of previously trained postures. The upper center (b) shows any currently recognized posture, which is useful as continuous feedback about any recognized postures (i.e., knowing if a new posture conflicts with any previously trained posture). The right side of the screen (c) shows controls to 'freeze' the current recog-

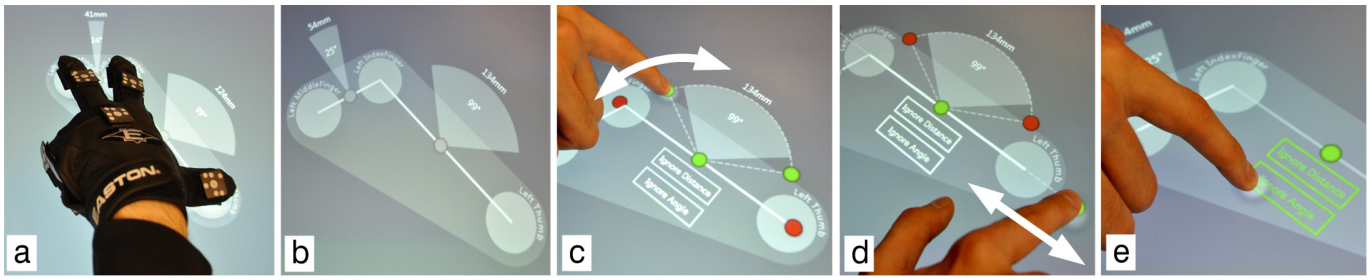

Figure 6. Using the posture configurator: (a) placing hand posture on the surface, (b) handparts get visualized with angles and distance between contact points, (c) person changes tolerance threshold for angle and (d) distance, or (e) sets to ignore these values.

nized contact points on the screen even when the hand is lifted off the surface, where that captured posture can then be saved.

To begin training a new posture, the developer simply puts her gloved hand onto the tool's center area (Figure 5d) and performs the desired posture. For example, for the posture shown in Figure 6a, the person placed three fingers on the surface: the thumb, index, and middle finger. The posture is visualized underneath (Fig. 6b) as the recognized touch points and the angle and distance between them. For example, the labels of Figure 6b states that the angle between thumb and index finger is 99 degrees, and that they are 134mm apart. Figure 6c/d illustrates how tolerance to these measures can be added by touching the handles and modifying the tolerance range for the given distance and angle values. The larger the tolerance value, the more relaxed the recognizer is when matching input to this template. If desired, the developer can ignore distance and/or angle measures for certain pairs of contact points by touching the 'Ignore Distance' or 'Ignore Angle' buttons below the visualization (Fig 6e). This can be useful to identify chording; for example, if both distance and angles are ignored, the mere placement of particular handparts on the surface (e.g., thumb and forefinger *vs.* thumb and pinkie) will trigger the posture event. Configured postures are saved as an XML file in the posture repository accessible by other parts of the toolkit. The posture training can then be repeated for any additional postures.

Besides single-hand postures, the toolkit also supports registration of bi-manual hand postures. Figure 7 illustrates this: when two hands touch the surface, the posture configurator tool now not only visualizes the contact points of both hands, but also the distance and angle *between* the two hands (and the two posture's centers respectively).

Gesture Configuration Tool

A developer can also configure gestures performed with any handpart or posture. TOUCHID recognizes gestures as performed movements, e.g., a 'circle', 'triangle' or 'C' movement made by (say) a pinched thumb and forefinger. It captures gestures by demonstration. For example, to train a circle gesture, the developer simply places the handpart or hand posture on the tabletop and performs the circle gesture movement. Internally, our toolkit uses a modified version of Wobbrock's gesture recognizer [39] to create one or multiple gesture templates. The toolkit later compares new input to the saved set of trained gestures. While our recognizer is simple, it differs from most other systems as gestures can be associated to a particular handpart or posture. For example, a flicking gesture performed with (say) a forefinger is considered different from a flicking gesture performed with the palm (e.g., the first is used to throw an object; the second is used to erase something). Alternately, a gesture can be saved as a universal template so that it is recognized regardless of the handpart performing it.

The API – Rapid and Expressive Programming

The programming API gives developers easy access to the information about people's touch interaction with the surface. Through a subscription-based event driven architecture familiar to most software engineers, developers can receive notifications about any handpart touching the surface, the person that touch belongs to, and any hand postures or performed gestures.

Walkthrough example

We illustrate the API with a deliberately simple walkthrough example that includes almost everything required to develop a first tabletop application that differentiates between handparts, postures, and people. To begin prototyping an application, the developer opens up the Visual Studio IDE, and selects the TOUCHID C# development template. The template is the starting point for new projects, containing the basic code required for all TOUCHID applications. This includes the base class (`TouchIDWindow`), a statement to initialize gloves (loading all the glove configuration XML files), and a statement to initialize a new posture recognizer and loading the posture configurations from the standard repository.

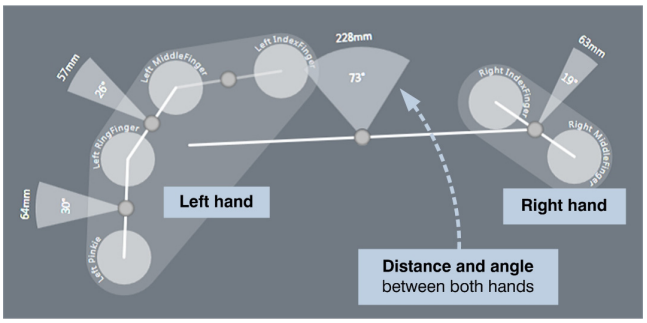

Figure 7. Bi-manual posture configuration (four fingers down with the left hand, and two fingers with the right hand).

```
public partial class Application:TouchIDWindow {
 public Application() {
  this.LoadGloves();
  this.PostureRecognizer = new PostureRecognizer();
  this.PostureRecognizer.LoadPostures(PostureRepository);
}
```

The developer adds two event callbacks to receive events when handparts (`HandpartDown`) and recognized postures (`PostureDown`) touch down onto the surface.

```
this.HandpartDown +=
    new EventHandler<TouchEventArgs>(HandpartDown);
this.PostureRecognizer.PostureDown +=
    new EventHandler<PostureEventArgs>(PostureDown);
```

While not illustrated, the toolkit also includes equivalent '`Changed`' and '`Up`' events triggered when a person moves the handpart or posture over the surface, and when the person lifts the hand up off the surface respectively.

Next, the developer adds the corresponding callback methods to the application that are called when the `HandpartDown` or `PostureDown` events occur.

```
void HandpartDown(object sender, TouchEventArgs e) {
  String   e.User.Name;            // e.g. "John"
  HandSide e.Hand.Side;            // e.g. HandSide.Left
  String   e.Handpart.Identifier;  // e.g. "Thumb"
  Point2D  e.Handpart.Position;    // e.g. x:20, y:53
  // do something with this information
}
void PostureDown(object sender, PostureEventArgs e) {
  String   e.User;                 // e.g. "Chris"
  String   e.Posture.Identifier;   // e.g. "StraightHand"
  Point2d  e.Posture.Position;     // e.g. x:102, y:79
  // do something with this information
}
```

The above callbacks illustrate use of the event property '`e`' that contains detailed information of the recognized handpart or posture. Depending on the callback, this includes:

- **Name of the user** performing the action **`[e.User.Name]`**
- **Which hand**: left or right **`[e.Hand.Side]`**
- **Part of the hand** **`[e.Handpart.Identifier]`** and its **position** **`[e.Handpart.Position]`**
- **Name of recognized posture** **`[e.Posture.Identifier]`** and its **center position** **`[e.Posture.Position]`**

Gesture recognition events are handled similarly to posture recognition. The developer initializes a gesture recognition object, loads the gesture configuration files from either the default repository or a custom folder, and then subscribes and adds matching event handlers to particular `GestureRecognized` events. The callback method gives the developer precise information about the gesture that triggered the event including the identifier of the recognized gesture, the user and handpart(s) performing the gesture.

```
this.GestureRecognizer = new GestureRecognizer();
this.GestureRecognizer.LoadGestures(GestureRepository);
this.GestureRecognized +=
    new EventHandler<GestureEventArgs>(GestureRecogn);

void GestureRecogn(object sender, GestureEventArgs e) {
  String  e.Gesture.Identifier;    // e.g. "circle"
  List    e.Gesture.Handparts;     // e.g. [IndexFinger]
  // do something with this information
}
```

The event handlers introduced so far subscribe to *all* handpart/posture/gesture events. They are global handlers as they receive all events no matter which person is performing them. In some cases, the developer may want to restrict callbacks to a particular user, or particular hand, or particular handpart. The code fragments below illustrate by example how this is done.

```
// Initialize user: name is a parameter to constructor.
// That name is associated with a glove in the glove
// configurator tool.
User john = new User("John") ;

// Subscribe to any of John's handpart down events
john.HandpartDown +=
      new EventHandler<TouchEventArgs>(JohnHandpartDown);

// Subscribe to any of John's left hand
// postures appearing on the surface
Hand johnslefthand = john.LeftHand;
johnslefthand.PostureDown +=
    new EventHandler<PostureEventArgs>
    (JohnLeftHandPostureDown);

// Subscribe to activities specific to John's thumb
Handpart thumb = john.HandParts.Thumb;
thumb.GestureRecognized += ...
thumb.HandpartDown += ...
thumb.HandpartUp += ...
```

EXPLORING INTERACTION TECHNIQUES

The last section contributed our API and its 3 configuration tools (glove, posture and gesture). This section continues by contributing (1) a diverse set of novel interaction techniques, and by doing so (2) an evaluation by practice of the API's expressiveness. While we don't include code, it should be fairly self-evident how these techniques could be implemented with TOUCHID. We describe several interaction techniques that are hand-part and posture aware. If applicable, we explain how we extended these techniques to allow for multi-user interaction. While we make no claims that the presented techniques represent the 'best' way to perform a particular action, we do believe our techniques serve as both an exploration of *what is possible* as well as a validation of the toolkit's expressiveness.

Identifying Individual Parts of the Hand

We now introduce several techniques that leverage this knowledge of the handpart associated with the owner (i.e., the user), location and orientation of a touch event.

Tool fingers. A typical GUI allows only one 'tool' or 'mode' to be activated at a time. The user selects the tool (e.g. via a tool palette), which assigns it to the mouse pointer. Our idea of 'tool fingers' changes this: for each user, their individual handpart can be its own tool, each with its own function. For example, consider the generic cut/paste operation: with tool fingers, touching an object with (say) the pinkie 'cuts' it, while the middle finger pastes it. We can also constrain visual transform actions: touching an object with the index finger allows moving, but the ring finger only allows rotating. As the toolkit further knows which glove (and thus which user) is touching the surface, we can manage left *vs.* right-handed people. The only information needed is whether a person is right or left hand-

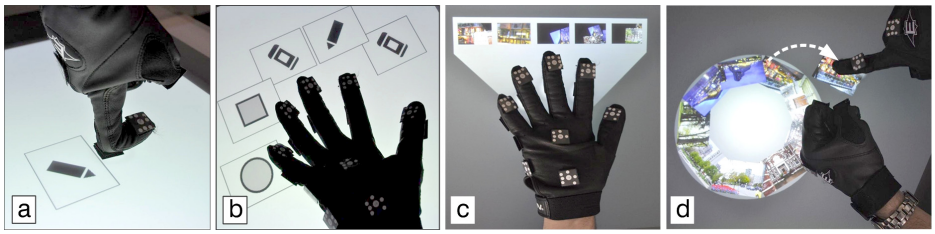

Figure 8. Preview functions: (a) knuckle shows tool assigned to this finger, (b) back of hand shows all assigned tools or (c) clipboard, (d) dragging items from a personal menu.

ed. With this, the application can assign functions to the dominant/non-dominant hand in a way that best matches that person's handedness.

Assigning tools. Tool fingers are flexible and user-aware: a person can associate a tool to a handpart simply by touching an icon in a tool palette with a particular handpart. Each individual user can choose their own tools (as we differentiate between users and their hands respectively), and also assign different tools to left and right hand. When a tool is assigned to a certain handpart, we also track which user owns that glove. Thus (say) the index finger of two persons can be easily distinguished. This gives a person the possibility to configure actions and tools according to their preferences, without interfering with other people using the surface. As discussed later in this section, this technique can be combined with a preview function of the tools assigned to handparts.

Preview and context menu. In a standard GUI, a context menu reveals the capabilities of an object. With tool fingers, we can use a different part of the finger to reveal a finger's function. For example, consider the case where each finger-tip is assigned a tool. When that finger's knuckle is placed on an empty part of the surface, a preview of the finger's tool appears (Fig. 8a). Alternately, a tool palette could be displayed around the finger's knuckle, and a new tool chosen by crossing over it with the knuckle.

Finger clipboard. Similar to the above, a different part of the finger can assign content to that finger. Consider a multi-finger clipboard, where a user can temporarily store and retrieve an object on a finger. Placing a finger's knuckle atop a graphical object will assign that object to its fingertip (either copied or cut). Touching the surface with that finger now pastes that object onto the surface at that location. When used as a context tool (i.e., the knuckle is placed on an empty location), the clipboard contents associated with that fingertip is shown.

Chorded Handparts

Besides individual handparts, TOUCHID can identify chords of two or more recognized handparts. This offers further powers, as different handpart combinations (of the same user) can be recognized as unique commands.

Combining tool effects. When tool fingers are assigned with mixable functions, their chording effect can be quite intuitive. Consider a finger-painting application, where a person assigned different colors to their fingers. Color mixing happens when different fingers are placed within an object. For example, if a person places the blue and yellow fingers inside a rectangle, it is filled with solid green. Interesting effects can be done by lifting up or placing down other fingers with associated colors, where a person can quickly alternate different color combinations without the need to remix the color in traditional color choosers.

Chorded modifiers. In the first section, we described how a knuckle can reveal aspects of its associated fingertip (i.e., its content or tool). An alternate approach is to use a chord combination to modify the behaviour of the tool finger. For example, the thumb (or the hand's wrist) can activate the preview function of the finger(s) on the surface.

Single-Hand Postures

Instead of chorded handparts, multiple handparts touching the surface can have certain meanings based on their relationship to each other. Even the same handparts may represent different commands based on their distance and angle (e.g., a fist, versus the side of the hand). We created several techniques that make use of static and dynamic postures using the posture configuration tool of TOUCHID.

Tool Postures. As with our previous examples, each posture can invoke a tool or function. For example, we can use the 'back of the hand' or 'back of fingers' posture as a context tool revealing all tools and/or clipboard contents assigned to all fingers. Figure 8b shows the tools assigned to a person's hand, and Figure 8c displays thumbnail images of clipboard data associated to the fingers. As another example, a fist posture raises a user's personal files that can be brought into the application. The interaction with these files can be restricted to its owner. For example, only the owner can make it public by dragging it from the personal menu to an empty area on the surface (Fig. 8d). Likewise, users other than the original owner may then not delete or modify someone else's data from the public area on the surface.

Dynamic postures: Grab'n'Drop. Postures can be dynamic, where changes in the posture can invoke actions. Our first example is *grab 'n' drop*, where users 'grabs' digital content with their hand, and place the content back onto the

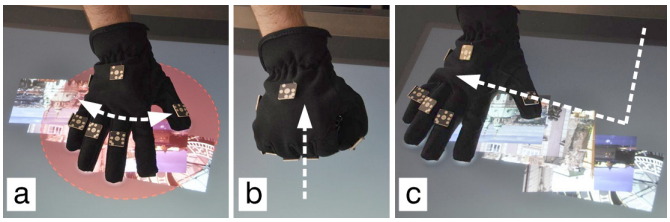

Figure 9. Grab'n'Drop: (a) spreading between fingers of flat hand defines selection area, (b) closing fingers and lifting hand up 'grabs' data, and (c) placing flat hand back down on surface and moving layouts files along the movement path of the person's hand.

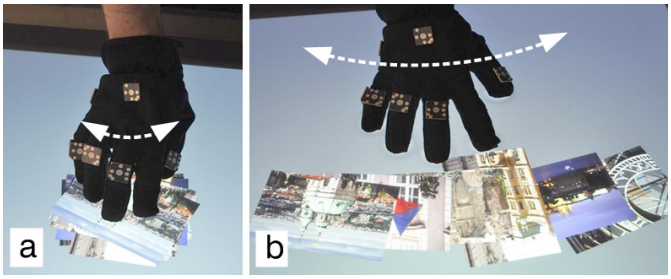

Figure 10. Interacting with piles: (a) fingers close together form pile and (b) spreading fingers reveals content.

surface at another location. The posture we designed requires the fingertips of all five fingers to be present on the surface. Spreading the fingers of the flat hand changes the selection area (Fig. 9a). Moving the fingers closer to the palm is then similar to 'grabbing' objects on a surface (i.e., making a fist, Fig. 9b). Once users grabbed objects, they are associated with their hand until they drop them. We designed two ways of dropping objects: first, as inverse operation, users put their five fingers down and move them further apart. Second, they can use their flat hand to draw a path along which the objects are aligned (Fig. 9c).

Dynamic postures: Interacting with piles. Our second example of a dynamic posture illustrates manipulations of piles of digital objects. We use the flat hand (i.e., five fingertips plus palm and wrist) and calculate the spread of fingers (i.e., the average distance of all fingers). When a user places the hand with spread fingers and then reduces this spread, items within a given radius around the hand are 'contracted' ultimately forming a pile (Fig. 10a). Likewise, the inverse operation (i.e., increasing the hand's spread) can be performed on an already existing pile to see its items (Fig. 10b). Both operations rely on the fingers' spread: the larger the spread, the larger the radius of the operation's influence.

Two-handed Interaction

The previous examples made use of handparts from only one hand. However, TOUCHID also recognizes both handparts and postures coming from two different hands and thus enables easy exploration of two-handed interactions, i.e., actions detected by two different gloves worn by the same user. Distinguishing users has high importance in such interactions to avoid accidental interference. Through our toolkit we are able to determine whether the two gloves belong to the same user. If this is not the case, multiple users may perform different actions. In the following we describe several techniques that use such interactions.

Precise Manipulations. The purpose of this technique is to allow precise object manipulations, e.g., scaling along the x-axis only, or rotating an object. Similar to *Rock and Rails* [37], we used the non-dominant hand to define the operation (i.e., scale, rotate) and the dominant one to perform it. However, we use the palm of the non-dominant hand to define the object the user wants to manipulate, while the finger of the dominant hand both defines and executes the operation. For example, dragging the index finger rotates the object, the middle finger scales along the x-axis (Fig. 11a), and the ring finger scales along the y-axis. Thus, the system is always aware of the user's intent without the need of separating operations. Naturally, chording multiple fingers combines actions. For example, using both middle and ring finger scales the object along both axes.

Level of manipulation. Applications may require users to manipulate single objects or all at once (i.e., manipulating all objects at the same time). Two-handed interaction can distinguish between these two cases, e.g., (1) using one or multiple fingers for object manipulations, and (2) using both hands (i.e., each of them with all five fingers down) to manipulate the canvas and its contained objects.

Object alignment. We also used two-handed interaction to align content by modifying *Grids & Guides*, a technique that allows for both linear and radial alignment [11]. The original method requires an intermediate step, namely defining grids and guides. Our technique allows both linear and circular alignment of objects using both hands. By using the sides of both hands, objects are aligned between them in a linear fashion (Fig. 11b). Changing the hands' distance increases/decreases the objects' spacing. Using the side of one hand while the other forms a fist results in circular placement around the fist's center (Fig. 11c). Here, the distance between both postures defines the circle's radius. In addition, both techniques rotate items accordingly.

Source and destination. Dragging objects can have two different meanings: move versus copy the object to a defined location. We designed a technique that allows both operations: fingertip equals *move*, and knuckle equals *copy*. The operation can affect either a single item (using the index finger) or a pile of objects (using the middle finger instead). For example, if users want to copy all items from a certain location, they place down the knuckle of the middle finger (Fig 11d). For all operations, the destination is given through the index finger of the second hand. This technique further allows rapidly reorganizing objects on the surface by repeatedly tapping at destinations.

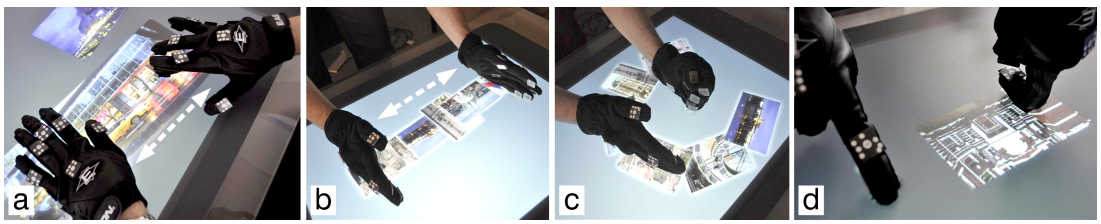

Figure 11. Two-handed interaction: (a) precise manipulations, (b) linear alignment, (c) circular alignment, and (d) shortcuts for copying objects by placing knuckle down on the object and index finger of second hand at destination.

Select by frame. A common operation in traditional desktops is the rubber band selection. Such interactions, however,

normally requires that the user starts the operation on an empty part of the workspace, which may be cumbersome or even impossible if there are many objects present. We overcome this by a two-handed selection technique. Forming an L-shape with both hands (Fig. 12) allows for precise location (i.e., intersection of index finger and thumb) and orientation of a rectangular frame. The index finger of the second hand additionally defines width and height of the selection (Fig. 12). We then extended this technique to give different meanings to such selection frames by using a combination of the thumb and different fingers for the L-shape: *selection* (index finger), *copy* (middle finger), *cut* (ring finger), and *paste* (pinkie). Additionally, we decided to use different fingers for defining the frame's size. While all fingers have the same effect, they act as a multi-finger clipboard as described before (i.e., what has been copied by the pinkie can only be pasted by the pinkie). Alternately, such frames could invoke different lenses and filters [20] to view the information contained between the hands.

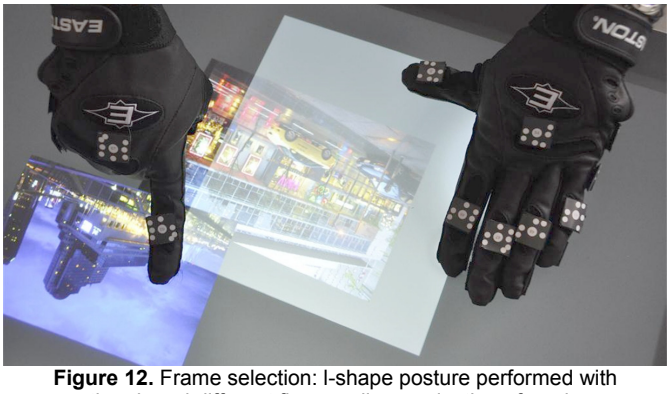

Figure 12. Frame selection: l-shape posture performed with thumb and different fingers allows selection of mode (e.g., select, copy, paste, cut); finger of second hand define second corner of selection frame.

Person-Aware Interaction Techniques

Because the *DiamondTouch* [6] could distinguish between people, the literature is replete with examples of how this information can be leveraged. TOUCHID provides similar information, and thus all techniques proposed in earlier work could be done with it as well. However, TOUCHID goes beyond that: as we revealed in our previous examples, user identification can be combined with knowledge of the particular user's handpart and hand, something that cannot be done easily with, e.g., the *DiamondTouch*. With TOUCHID, programmers can furthermore easily develop applications that make use of rules and roles (e.g., in games or educational applications) [6,28], cooperative gestures (e.g., collaborative voting through the same postures of each user) [26], or personalized widgets (e.g., users 'call' a customized widget through a personalized posture) [31]. Additionally, actions can be reverted on a per-user level, as the application is aware of which user is doing what [40].

DISCUSSION

The interaction techniques described above serve as an evaluation-in-practice of our toolkit and its expressive powers. The exposed methods and events in the API along with the three configuration tools – while simple and easy-to-use – provide all the required information in enough level of detail for designing all of the techniques above. While we don't describe how these were coded, a reasonable programmer using our toolkit should be able to replicate and extend any of these techniques without too much difficulty. Indeed, the techniques above are just an initial exploration. As we were developing these systems, we saw many other variations that could be easily created. Overall, the above examples emphasize the potential of how handpart aware techniques – as enabled by our toolkit - can lead to more expressive tabletop interactions.

Of course, some of the techniques could be (or have been) implemented without the gloves or toolkit. Yet in many cases it would require complex programming (e.g., computer vision and machine learning algorithms) to detect certain postures. In some other cases (such as robust identification of fingertips) it would not be possible at all. Overall, the toolkit allowed *rapid exploration* of handpart aware techniques, in order to find adequate and expressive forms of tabletop interaction.

Caveat. While we frame the particular interaction techniques as a secondary contribution in their own right, we do not argue that the techniques we presented are necessarily the best mapping of handparts to a particular action. In many cases there is more than one possible solution for assigning handparts, postures, or gestures to a particular action. Future qualitative and quantitative studies will help in answering the question of how far we can or should go with these techniques. Such questions are: How many functions assigned to handparts are too much? What are the personal preferences of users? What kind of single- or multi-handed postures are easy or difficult to perform?

What we do claim strongly is that the TouchID toolkit can help us explore this design space. This is why it is our primary contribution. Rapidly prototyping handpart-aware applications will allows us to compare and evaluate the benefits, performance, and problems of particular techniques.

CONCLUSION

TouchID is a downloadable toolkit that (currently) works atop a Microsoft Surface, where it provides the programmer with what handpart, what hand, and what user is touching the surface, as well as what posture and what gesture is being enacted. Its API is simple yet powerful. We illustrated its expressiveness by several novel tabletop interaction techniques that exploit this extra information: individual functions for each handpart, pairing handparts, and using single- or multi-handed postures and gestures, and distinguishing between multiple users.

Overall, we believe that distinguishing the handparts that are causing the touches on an interactive surface can lead to novel and expressive tabletop interaction techniques. We offer TouchID – currently based on the very affordable but reliable fiduciary glove – as a way for the community to

work in this exciting area. Instead of struggling with low-level implementation details such as computer vision and machine learning algorithms, we (and others) can quickly explore a large set of alternative techniques – many of which can be seen as pointers to possible future explorations.

ACKNOWLEDGMENTS

This research is partially funded by the iCORE/NSERC/ SMART Chair in Interactive Technologies, Alberta Innovates Technology Futures, NSERC, and SMART Technologies Inc.

REFERENCES

1. Benko, H., Ishak, E.W., and Feiner, S. Cross-dimensional gestural interaction techniques for hybrid immersive environments. *Proc. of VR '05,* IEEE (2005), 209-216.
2. Benko, H., Saponas, T.S., Morris, D., and Tan, D. Enhancing input on and above the interactive surface with muscle sensing. *Proc. of ITS '05,* ACM (2009), 93-100.
3. Buchmann, V., Violich, S., Billinghurst, M., and Cockburn, A. FingARtips: gesture based direct manipulation in Augmented Reality. *Proc. of GRAPHITE '04*, ACM (2004).
4. Dang, C.T., Straub, M., and André, E. Hand distinction for multi-touch tabletop interaction. *Proc. of ITS '09*, ACM (2009), 101-108.
5. Diaz-Marino, R.A., Tse, E., and Greenberg, S. Programming for Multiple Touches and Multiple Users: A Toolkit for the DiamondTouch Hardware. *Companion Proc. of UIST'03*, ACM (2003).
6. Dietz, P. and Leigh, D. DiamondTouch: a multi-user touch technology. *Proc. of UIST '01*, ACM (2001), 219-226.
7. Echtler, F. and Klinker, G. A multitouch software architecture. *Proc. of NordiCHI '08*, ACM (2008), 463–466.
8. Echtler, F., Huber, M., and Klinker, G. Shadow tracking on multi-touch tables. *Proc. of AVI '08*, ACM (2008), 388-391.
9. Epps, J., Lichman, S., and Wu, M. A study of hand shape use in tabletop gesture interaction. *CHI '06 extended abstracts*, ACM (2006), 748-753.
10. Freeman, D., Benko, H., Morris, M.R., and Wigdor, D. ShadowGuides: visualizations for in-situ learning of multi-touch and whole-hand gestures. *Proc. of ITS '09*, ACM (2009).
11. Frisch, M., Kleinau, S., Langner, R., and Dachselt, R. Grids & guides: multi-touch layout and alignment tools. *Proc. of CHI '11*, ACM (2011), 1615–1618.
12. Gokcezade, A., Leitner, J., and Haller, M. LightTracker: An Open-Source Multitouch Toolkit. *Comput. Entertain. 8*, 3 (2010), 19:1–19:16.
13. Greenberg, S. Toolkits and interface creativity. *Journal of Multimedia Tools and Applications (JMTA) 32*, 2, Springer (2007), 139-159.
14. Han, J.Y. Low-cost multi-touch sensing through frustrated total internal reflection. *Proc. of UIST '05*, ACM (2005).
15. Hancock, M., Carpendale, S., and Cockburn, A. Shallow-depth 3d interaction: design and evaluation of one-, two- and three-touch techniques. *Proc. of CHI '07*, ACM (2007).
16. Hansen, T.E., Hourcade, J.P., Virbel, M., Patali, S., and Serra, T. PyMT: a post-WIMP multi-touch user interface toolkit. *Proc. of ITS '09*, ACM (2009), 17–24.
17. Holz, C. and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. *Proc. of CHI '10*, ACM (2010), 581-590.
18. Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. TUIO: A Protocol for Table-Top Tangible User Interfaces. *Proc. GW '05*, (2005).
19. Kaltenbrunner, M. and Bencina, R. reacTIVision: a computer-vision framework for table-based tangible interaction. *Proc. of TEI '07*, ACM (2007), 69–74.
20. Käser, D.P., Agrawala, M., and Pauly, M. FingerGlass: efficient multiscale interaction on multitouch screens. *Proc. of CHI '11*, ACM (2011), 1601–1610.
21. Krueger, M.W., Gionfriddo, T., and Hinrichsen, K. VIDEOPLACE - An Artificial Reality. *SIGCHI Bull. 16*, 4 (1985), 35-40.
22. LaViola, J.J. *A Survey of Hand Posture and Gesture Recognition Techniques and Technology*. Tech. Report CS-99-11, Department of Computer Science, Brown University, 1999.
23. Letessier, J. and Bérard, F. Visual tracking of bare fingers for interactive surfaces. *Proc. of UIST '04*, ACM (2004), 119-122.
24. Marquardt, N., Kiemer, J., and Greenberg, S. What Caused That Touch? Expressive Interaction with a Surface through Fiduciary-Tagged Gloves. *Proc. of ITS '10*, ACM (2010).
25. Microsoft Inc. Tagged Objects. MSDN Library, http://msdn.microsoft.com/en-us/library/ee804823 (v=Surface.10).aspx, Retrieved June 17, 2010.
26. Morris, M.R., Huang, A., Paepcke, A., and Winograd, T. Cooperative gestures: multi-user gestural interactions for co-located groupware. *Proc. of CHI '06*, ACM (2006).
27. Morris, M.R., Paepcke, A., Winograd, T., and Stamberger, J. TeamTag: exploring centralized versus replicated controls for co-located tabletop groupware. *Proc. of CHI '06*, ACM (2006), 1273-1282.
28. Morris, M.R., Ryall, K., Shen, C., Forlines, C., and Vernier, F. Beyond "social protocols": multi-user coordination policies for co-located groupware. *Proc. of CSCW '04*, ACM (2004).
29. Olsen, J. Evaluating user interface systems research. *Proc. of UIST '07*, ACM (2007), 251–258.
30. Partridge, G.A. and Irani, P.P. IdenTTop: a flexible platform for exploring identity-enabled surfaces. *CHI '09 Extended Abstracts,* ACM (2009), 4411–4416.
31. Ryall, K., Esenther, A., Forlines, C., et al. Identity-Differentiating Widgets for Multiuser Interactive Surfaces. *IEEE Comput. Graph. Appl. 26*, 5 (2006), 56-64.
32. Schmidt, D., Chong, M.K., and Gellersen, H. HandsDown: hand-contour-based user identification for interactive surfaces. *Proc. of NordiCHI '10*, ACM (2010), 432–441.
33. Schmidt, D., Chong, M.K., and Gellersen, H. IdLenses: dynamic personal areas on shared surfaces. *Proc. of ITS '10*, ACM (2010), 131–134.
34. Shen, C., Vernier, F.D., Forlines, C., and Ringel, M. DiamondSpin: an extensible toolkit for around-the-table interaction. *Proc. of CHI '04*, ACM (2004), 167–174.
35. Sturman, D.J. and Zeltzer, D. A Survey of Glove-based Input. *IEEE Comput. Graph. Appl. 14*, 1 (1994), 30-39.
36. Wang, R.Y. and Popović, J. Real-time hand-tracking with a color glove. *Proc. of SIGGRAPH '09*, ACM (2009), 1-8.
37. Wigdor, D., Benko, H., Pella, J., Lombardo, J., and Williams, S. Rock & rails: extending multi-touch interactions with shape gestures to enable precise spatial manipulations. *Proc. of CHI '11* ACM (2011), 1581–1590.
38. Wobbrock, J.O., Morris, M.R., and Wilson, A.D. User-defined gestures for surface computing. *Proc. of CHI '09*, ACM (2009).
39. Wobbrock, J.O., Wilson, A.D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *Proc. of UIST '07*, ACM (2007), 159-168.
40. Wu, M. and Balakrishnan, R. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. *Proc. of UIST '03*, ACM (2003), 193-202.