Interaction and Modeling Techniques for Desktop Two-Handed Input

Ken Hinckley, Mary Czerwinski, and Mike Sinclair
Microsoft Research, One Microsoft Way, Redmond, WA 98052
{kenh, marycz, sinclair}@microsoft.com; Tel: +1-425-703-9065

ABSTRACT
We describe input devices and two-handed interaction techniques to support map navigation tasks. We discuss several design variations and user testing of two-handed navigation techniques, including puck and stylus input on a Wacom tablet, as well as a novel design incorporating a touchpad (for the nonpreferred hand) and a mouse (for the preferred hand). To support the latter technique, we introduce a new input device, the TouchMouse, which is a standard mouse augmented with a pair of one-bit touch sensors, one for the palm and one for the index finger. Finally, we propose several enhancements to Buxton's three-state model of graphical input and extend this model to encompass two-handed input transactions as well.

Keywords
Two-handed input, three-state model, input devices, tablets, touchpads, TouchMouse, map navigation

INTRODUCTION
Two-handed input is a promising technique to improve the directness and degree of manipulation afforded by desktop computers. A strong foundation of research exists [7][8][16][24], yet techniques that allow both hands to drive continuous input signals are still not in common use.

There are many reasons for this. Designers have only recently begun to develop an understanding of human bimanual behaviors [9][15][13][12]. The application of this knowledge to the design and development of interaction techniques, with transducers capable of capturing appropriate input signals, is still lacking. We also lack descriptive models to specify our interaction techniques, or to explore alternate designs. On the pragmatic side, until the recent introduction of the Universal Serial Bus standard, it has been difficult to connect multiple input devices to PC's, and furthermore users are very accustomed to the mouse and keyboard and thus transitioning the installed user base to new input techniques is a major issue.

UIST '98. San Francisco, CA

Fig. 1 Input devices for our desktop two-handed interaction techniques: a touchpad for the nonpreferred hand and the TouchMouse, a modified Microsoft IntelliMouse that can sense when the user is touching it.

The current work makes a number of contributions. We describe implementations of bimanual input techniques for a map manipulation task using two different sets of input devices: (1) a 12x12" Wacom tablet with a puck and pressure-sensitive stylus and (2) a novel input design incorporating the TouchMouse and a touchpad (*Fig. 1*). We provide informal usability observations and suggest strengths and weaknesses for each approach.

With the goal of more precisely modeling our devices and techniques, we suggest enhancements to Buxton's three-state model [4], a descriptive model for pointing devices, and we extend this model to two-handed transactions using a Petri net representation. We present these extended models in the context of the current work because our input devices and design problems helped to suggest the new models, while at the same time the new models helped us to better understand and describe our designs.

PREVIOUS WORK
A number of systems demonstrate two-handed input techniques. Buxton and Myers' two-handed input study [7] shows that using a pair of touch-sensitive strips for jumping and scrolling with the nonpreferred hand can result in improved performance. The T3 interface paradigm [16] explores the use of tablets, two hands, and transparency for high-end artwork applications. The ToolGlass metaphor [2] uses a mouse and a trackball, driven by the nonpreferred hand, to position semi-transparent tool sheets. Zeleznik et al. [24] use both hands on a Wacom tablet to perform camera manipulations. Previous examples of two-handed map manipulation include the metaDesk [23] and the HoloWall [20]. Other examples that leverage physical objects for two-handed interaction include Bricks [8] and real-world interface props [11].

Balakrishnan and Patel describe the PadMouse, which is a touchpad integrated with a mouse for nonpreferred hand input [1]. Balakrishnan and Patel use the touchpad for command selection, whereas we use our touchpad for spatial positioning tasks. The PadMouse also has some similarity to our TouchMouse since it can sense when the user is touching the pad. We present models that distinguish these devices later in this paper (*Fig. 8, Fig. 9*).

Electric field sensing devices [26][21] can detect the capacitance of the user's hand or body to allow deviceless position or orientation sensing in multiple dimensions. Our TouchMouse also senses capacitance, but we use this signal in a contact sensing role, and we describe a number of novel applications for such contact-based signals. These two different modes of sensing capacitance could potentially be combined in future input devices. Finally, Harrison et al. [10] use pressure sensors to detect contact with handheld displays to (for example) automatically detect the user's handedness.

INPUT DEVICES AND INTERACTION TECHNIQUES

The first implementation of our two-handed map manipulation techniques utilizes a single 12x12" Wacom ArtZ II tablet (*Fig. 2, left*) with a puck for the nonpreferred hand and a pressure-sensitive stylus for the preferred hand. Our design goal is to support annotation, panning, and zooming of maps from a user interface that is free of heavyweight mode switches (e.g., mode switches that require the user to click on an icon or perform a menu selection to change the behavior of the pointing devices). The additional degrees-of-freedom afforded by a two-handed interface can potentially simplify the syntax of required input actions and thus result in a simpler user interface [3].

In essence, the two-handed interface allows users to think in terms of "navigating the map," rather than strictly in terms of the atomic actions of panning or zooming. Using Buxton's cognitive chunking approach [3], a hierarchy of subtasks for map navigation is illustrated in Fig. 2 (*right*). Note that even performing just the *Zoom* subtask with a mouse is problematic because the mouse senses two continuous degrees of freedom, while continuous zooming requires three degrees of freedom (the scaling factor and an (x, y) origin for the scaling transformation).

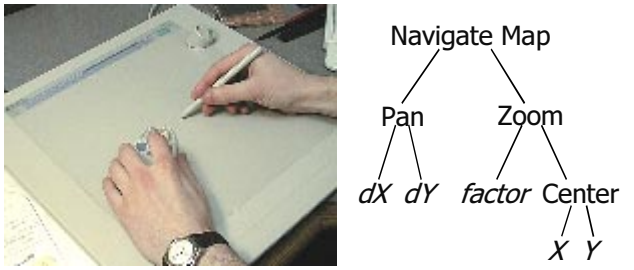

Fig. 2 *Left*: 12x12" Wacom tablet with puck and stylus. *Right*: Chunking of subtasks for the map navigation task.

The puck and stylus interface provides this functionality as follows: clicking and dragging with the puck alone pans the map. Pressing down with the stylus alone annotates the map by leaving behind an ink trail. Clicking the puck while pressing down with the stylus allows two-handed dragging.

The two-handed dragging follows a "stretch and squeeze" metaphor: pulling one's hands apart stretches the map (to get more detail) while pushing one's hands together squeezes the map (to compress the map, getting a bird's eye view). While stretching and squeezing, users can also translate both hands in the same direction, and thus achieve compound panning and zooming actions[1]. This navigation metaphor is similar to a technique described by Kurtenbach et al. [16], except that rotation is not allowed.

For our user testing, we first had 6 test users[2] practice using each device individually on the tablet. We told users about the "stretch and squeeze" metaphor and had them try out several map manipulations. Users then performed tasks (such as "Pan and zoom the map so that Los Angeles and Salt Lake City are both visible") on their own.

Some users were initially uncomfortable using the puck in the nonpreferred hand. Several users had trouble discovering that the stylus sensed pressure, or that they needed to click a button on the puck for it to pan the map. However, after the initial practice phase, all users were able to complete the structured tasks without further intervention from the usability tester. When first trying out the two-handed dragging actions, most users performed pure zooming actions (stretching or squeezing), but after just a few trials, users gained sufficient skill to perform compound panning and zooming actions.

Test users also tried using a mouse with Microsoft TripPlanner98, a commercially available mapping program. Several users commented that the availability of standard mouse functions (e.g. right-clicking) made it easier to discover how to use it. However, even given the incomplete nature of our tablet-based prototype, reactions from users were generally quite positive when compared to the TripPlanner98 application and the mouse. One user commented that it "seemed pretty natural to me," while another felt that the tablet was "the most efficient way to do it… it's just a matter of learning to use the left hand first." Another user remarked that using the mouse to look at maps requires "lots of steps" whereas using both hands "is more continuous… one movement is one thing."

The TouchMouse

We were encouraged by user responses to our tablet-based techniques, but we knew that in the long run a design

[1] The technique samples the position of the puck (call this P_0) and the stylus (S_0) when the user first grabs the map. It computes the midpoint of the cursors, $M_0 = (P_0 + S_0) / 2$. It repeatedly takes that initial point M_0 and for the device positions P_1 and S_1 (with midpoint M_1) at each subsequent frame it computes a transformation matrix X consisting of a translation offset and a uniform scale (with the scaling relative to an origin at M_0), such that X satisfies: $M_0 X = M_1$.

[2] Test users were recruited through the Microsoft usability labs and had varying computer experience. All users had mouse experience.

including a mouse would probably be more practical for common desktop applications. As we began to explore this design space, we found it was useful to have a mouse that could sense when it was being touched, so we begin with a description of this device, which we call the TouchMouse.

The TouchMouse (*Fig. 1, right*) is a modified Microsoft IntelliMouse that integrates the mouse with the ability to sense when the user's hand is touching it. Our prototype design can support up to 4 touch sensors; the "touch sensors" are made with a conductive paint that is applied to the mouse shell. The conductive paint is connected to the TouchMouse internal circuitry, which senses the parasitic capacitance of the user's hand when it contacts the touch sensor(s)– no mechanical actuation of a switch is necessary. The capacitance of the user's hand induces a slight time delay in the circuit. When this time delay passes a critical threshold, a "Touch" or "Release" event is generated. To provide a strong coupling with the passive tactile feedback that the user feels when he or she touches the device, the capacitance sensors generate Touch/Release events only and exactly when the user's finger or palm actually makes (or breaks) contact with the surface. Our current prototype sends the touch data to the host PC's parallel port.

The TouchMouse described here provides two independent bits of touch data, one for the palm area (which actually extends along the side of the mouse to sense thumb contact) and one on the left mouse button which senses the index finger. The palm sensor is useful for implicit actions that can take advantage of knowing when the user grabs or releases the mouse. For example, when the user first touches the mouse, the computer reveals the mouse cursor and performs a quick animation (0.3 seconds long) of a 200-pixel radius circle collapsing on the mouse cursor position. This helps direct the user's attention to the focus of interaction. When the user lets go of the mouse, the computer hides the mouse cursor by fading it out over 2 seconds. The palm sensor may also have applications for user modeling [14]; for example, one can now differentiate a user dwelling over an icon with the mouse, versus a user that has let go of the mouse and just happened to leave the cursor over an icon.

The finger sensor is intended for more explicit user actions, since it is easy to lift one's finger from the button as an intentional action. For example, in our two-handed TouchMouse + touchpad interaction technique, we use the finger sensor (in combination with the touchpad's ability to sense contact) as a cue that the user is beginning an interaction involving both input devices. We have also experimented with touching the button as a "touch-to-talk" mechanism for voice recognition applications that gives the user fine-grain control over whether or not the computer should listen for voice commands. Although we have not yet user-tested this feature, we feel it is more effective than common alternatives such as using a push-to-talk button on the keyboard or using voice commands to activate and deactivate the recognizer.

Users were able to quickly learn and use the touch-sensing features of the TouchMouse and their reactions to the device were generally positive, although discovering the sensors may be an issue [10]. In order to get an unbiased first impression, we did not tell users about the touch sensors during initial use; half of 6 test users discovered both touch sensors without any prompting. Two users would grab the mouse very quickly and thus never notice that it sensed when they touched it. Another user was in the habit of *always* holding his index finger off of the button when not actually clicking or dragging, and thus he did not initially notice the finger sensor.

Two-handed TouchMouse + Touchpad Map Navigation

Here, we propose a two-handed technique that uses the TouchMouse and a Synaptics touchpad to support the "stretch and squeeze" metaphor. We decided to explore designs incorporating a mouse plus a touchpad for nonpreferred-hand use because we felt that touchpads have a number of desirable properties for this application, such as low cost, a small, fixed device footprint, rapid device acquisition speed, and the ability to sense when being touched. A critical distinction between the tablet implementation of these techniques versus a mouse and touchpad implementation is that a mouse and touchpad cannot sense the same state information as the puck and stylus devices. Buxton's three-state model [4] illustrates this distinction by defining three states as shown in Table 1:

State	Description
0	*Out Of Range:* the device is not in its physical tracking range.
1	*Tracking:* moving the device causes the tracking symbol to move.
2	*Dragging:* allows one to move objects in the interface.

Table 1 The states defined by Buxton's 3-state model [4].

The puck and stylus on the Wacom tablet can each sense all three of these states. For example, Fig. 3 shows the state diagram for the puck:

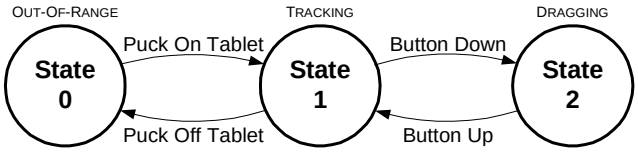

Fig. 3 State transitions for the puck on the Wacom tablet.

By comparison, the touchpad senses only states **0** and **1**, while a standard mouse senses only states **1** and **2** [4][6], as shown in Fig. 4:

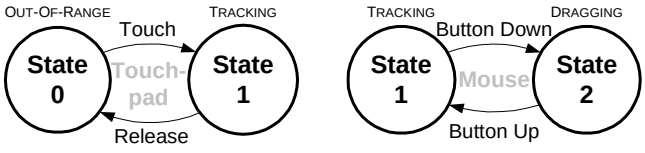

Fig. 4 State models for a touchpad (left) and a standard mouse (right).

Fig. 5 Stretching the map with the TouchMouse and touchpad. The user first positions the cursors around an area of interest (left). Clicking and dragging the mouse and touchpad stretches the map; a dragging rectangle helps to show the area of the map that has been "grabbed" (middle). When the user releases the mouse button, the map updates, filling in additional detail (right). The new map fades in (as a 1 second animation) to reduce any visual discontinuities.

In principle, a dragging state (state **2**) can be introduced for the touchpad, as is commonly done on laptops with a touchpad. Typically the user holds down a button while touching the pad, or performs a "tap-and-drag" gesture to drag objects. In practice, however, such mechanisms can be tedious or can interfere with cursor motion [17]. Thus, we considered these mechanisms unacceptable for our nonpreferred-hand usage of the touchpad.

The touchpad's lack of a dragging state can be overcome by conceiving of the TouchMouse and touchpad as a "unified" two-handed device that supports an out-of-range state (when you are not touching the devices), a cursor tracking state, and dragging states (both one-handed and two-handed). The devices can be used individually or in combination: using the touchpad alone pans the map, while clicking and dragging the TouchMouse alone annotates the map by leaving behind an ink trail. When used in combination, touching the index finger to the mouse allows cursor tracking with *both* devices, while clicking the mouse allows dragging ("stretching and squeezing the map") with *both* devices. Thus the nonpreferred hand provides only positional and contact information; we never require the nonpreferred hand to click button(s) or control pressure on the touchpad. Fig. 5 shows the visual feedback provided by our prototype mapping application.

Although we feel that the TouchMouse + touchpad design results in effective two-handed input, we should note that this feels different than the tablet-based technique. Based on our usability observations, two-handed actions with the TouchMouse + touchpad are not as natural as they are on the tablet. We see two main limitations of the TouchMouse + touchpad input technique. The first is that the TouchMouse and touchpad operate in a separate coordinate space for each hand (unlike the tablet, where both devices operate in a common absolute coordinate system). Thus, the TouchMouse + touchpad design cannot benefit from the user's natural ability to know where one hand is relative to the other [12]. A second difficulty is that the control / display ratios of the two devices may not be equivalent. This means that equal physical movements of the hands might result in unequal cursor movements on the screen. In practice, we find that this is only a significant problem if mouse motion is set to one of its fastest settings.

The split control space is both a limitation of the technique and an advantage. On the tablet, the puck and stylus can bump into one another, making it impossible to place the cursors right next to one another (unless an offset is introduced). With the TouchMouse and touchpad, this cannot happen. But this in itself is also a problem because the map transformation is much more sensitive to small movements whenever the cursors are close to one another.

We refer to this as the "cross-over problem" because if the user is trying to zoom out by squeezing the map together, it is easy to overshoot, have the cursors cross over one another, and suddenly find that the map is zooming back in (because the distance between the cursors is now increasing, and thus stretching the map). This is very disorienting. To address this issue, we implemented software constraints that limit where the cursors can go during two-handed dragging. It prevents the cursors from crossing over by setting up an imaginary wall between the touchpad cursor and the mouse cursor. It also defines a small "dead zone," currently set to a radius of 30 pixels; when the cursors are within this radius of one another, the map is not scaled regardless of the cursor separation. We find that this technique effectively eliminates the problem.

We also provide interactive audio feedback for panning and zooming of the map. Audio feedback for panning takes the direction and speed of panning and maps these to volume, pitch, and timbre to produce a realistic "paper sliding on a desk" sound. Stretching the map (zooming) creates a sound like a guitar string being stretched. The volume also becomes slightly louder as the size of the region being manipulated increases. The sounds are generated by sequencing parametric sound events using the MIDI synthesizer on a Creative Labs AWE64 Gold card.

During user tests[3], after a brief tutorial and some practice to get a feel for the devices, users were able to make effective use of both hands and complete all of the structured tasks on their own. User reactions were more varied than with the tablet version of the interface. One user commented that "I think it's great. It just takes some time to learn it." while another felt strongly that "It didn't work that well for me. I'd prefer to just use one hand." Another user commented that "to get the two hands to operate together is a new feeling—it only took a minute, though."

Alternative Mouse + Touchpad Techniques

An alternative design uses the touchpad and a standard mouse (without touch sensing). Without a touch sensor, it is impossible to know if the user is using the touchpad alone, or if the user is trying to use both the mouse and the touchpad together. Thus in this design touching the pad alone only moves the cursor for the nonpreferred hand. Clicking and dragging the mouse alone performs the inking behavior. Clicking and dragging the mouse while touching the pad initiates two-handed dragging.

Panning the map can be achieved by clicking the mouse without moving it, and then moving the nonpreferred hand on the touchpad. The rationale for this design choice is that it is *not* natural to perform two-handed actions where the nonpreferred hand moves relative to the preferred hand [9]; thus such a movement with the devices should be interpreted as a one-handed behavior. It is also possible to pan by resorting to clicking a button on the touchpad, but this feels awkward with the nonpreferred hand[4].

Another variation uses the touchpad only for panning, while the wheel on the IntelliMouse is used for zooming (centered on the mouse cursor). This technique does not support compound panning and zooming tasks very well; in practice, panning the map with the touchpad or zooming the map with the wheel must be serially interleaved. However, it does not depend upon coordinating the action of the two devices and thus is not affected by the split control spaces or potentially unequal control / display ratios of the devices.

EXTENDING THE THREE-STATE MODEL

As we implemented our interaction techniques and input devices and contemplated design alternatives, we started to feel that Buxton's 3-state model, as it currently exists, was not quite appropriate for some of the design problems we faced. For example, we felt that the TouchMouse afforded some very different interactions than a puck on the Wacom tablet, but the 3-state model treats these as nearly identical three-state devices (both devices sense out-of-range, tracking, and dragging states).

In light of this experience, our goal here is to extend Buxton's 3-state model to a wider range of design problems, input devices, and interaction techniques. We offer two straightforward extensions to the 3-state model. First, we show how annotating states of the model with continuous properties (such as position, rotation, or pressure) sensed while in that state is a useful tool for design and can better describe the idiosyncrasies of various devices. Second, we draw a distinction between out-of-range events based on touch, versus those based on proximity of an input device to a sensor.

Annotating States with Continuous Properties

We propose that annotating each state with the continuous device properties that are sensed while in that state is useful as a reminder of exactly what actions are possible while in a given state, and can potentially suggest new devices. This notation can also capture many properties of multi-channel input devices (such as the IntelliMouse or PadMouse [1]).

This has some similarities to the design space proposed by Mackinlay, Card, and Robertson [19], which describes devices with multiple continuous and discrete sensors. Although the design space shows connections among the properties that a device senses, it does not describe the state transition behavior of devices. This is an important issue because the continuous properties of a device often vary depending on its current state. Our contribution is to add a notation for continuous properties (*Table 2*) to the 3-state model, which does capture state transition behavior.

Notation	Description of property sensed
x	absolute position sensing
dx	relative position sensing (motion sensing)
R	absolute angular (e.g. calibrated dial)
dR	relative angular (e.g. Intellimouse wheel)
F, dF	Force or change in force
T, dT	Torque or change in torque
nil	State cannot sense any continuous signal.

Table 2 Notation for continuous properties. This is essentially the same shorthand notation used by Mackinlay et al. [19]. We add ***nil*** to indicate states that cannot sense any continuous property.

This notation addresses one limitation of the original 3-state model that Buxton pointed out, namely, representing transducers capable of sensing pressure. For example, Fig. 6 describes the Wacom pressure-sensitive stylus.

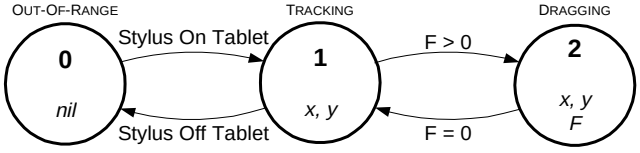

Fig. 6 Annotated states for a pressure-sensing stylus, with stylus pressure (F) used to control a continuous property (e.g. line thickness) in an "inking" mode.

State models for the IntelliMouse (*Fig. 7*) and the PadMouse [1] further illustrate the utility of this approach. For both of these devices, the continuous properties that can be effectively sensed vary depending on the current state of the device. We defer our model of the PadMouse (*Fig. 9*)

[3] Another 6 users (who had not tried the tablet, but were from the same pool of users) participated in this user test.

[4] A pressure-sensing touchpad might be useful in resolving this problem (where light pressure moves the cursor, heavier pressure pans the map). The Synaptics touchpad senses contact area [17].

until after the following section because the PadMouse also incorporates touch-sensing capabilities.

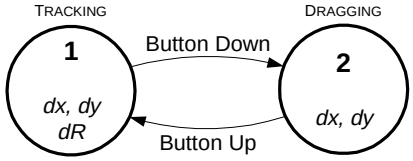

Fig. 7 The IntelliMouse. We omit *dR* from state 2 because it is difficult to turn the wheel while dragging the mouse. This precludes some useful transactions, such as scrolling a document while dragging a figure into place.

Distinguishing Touch vs. Device Proximity

Buxton's 3-state model does not distinguish between out-of-range states based on touch versus those based on device proximity. However, when considering how we should model the TouchMouse, the touchpad, and the Wacom tablet, we found it useful to differentiate the two. We should note that Buxton's input device taxonomy [5] *does* draw a similar distinction between devices that work by touch versus devices that require a mechanical intermediary. Our contribution is to apply this distinction to the 3-state model and to show that devices that have some discrete properties of each classification are possible. In short, we propose that Buxton's out-of-range state 0 is actually the union of two potentially different states:

- State **T0**, an out-of-range state triggered by **T**ouch (hand or body contact with a device); and
- State **P0**, an out-of-range state triggered by device **P**roximity to a sensor. (We do not consider the hand itself to be a device; only a physical intermediary can play this role for state **P0**.)

We claim this distinction is useful for several reasons. First, we found it useful in the design of our two-handed techniques. For example, it shows why it would be awkward to implement a similar technique with the TouchMouse and a puck (instead of a touchpad): the puck can generate an out-of-range event only when it is lifted and that would be much more awkward than lifting one's finger from the pad. The technique would only work well if the puck could also sense when it was being touched.

Second, both states can be used profitably in the design of interaction techniques, but are natural to use for different things. We have already seen application of state **T0** in the palm sensor of the TouchMouse to provide enhanced cursor functionality. State **P0** on the Wacom puck can be used to implement a relative mode where the user can make "skating" gestures (much like mouse users will do) by lifting the puck to allow relative motion on the tablet.

Third, this suggests new devices that sense both states **T0** and **P0**. We are not aware of any such device, either commercially or in the research literature. To show that there could be such devices, we propose three examples:

(1) The "TouchPuck," a puck for the Wacom tablet that senses when the user is holding it;

(2) A TouchMouse that can also sense when it is lifted off of the desk (state **P0**).

(3) The "TouchBricks," a modified version of Fitzmaurice's Bricks where each Brick would sense when the user is holding it.

At this point, we can introduce a descriptive model for the TouchMouse (*Fig. 8*). For the interaction techniques that we have developed, we like to think of the TouchMouse as a four-state device that supports state **T0** but not state **P0**, with the fourth state being a "touch-dragging" state that is triggered by touching the index finger to the mouse button. We call this state **T2** (short for **T**ouch-based state **2**). Note that in comparison to the mouse of Fig. 7, state **T2** is an intermediary state between mouse tracking (state **1**) and dragging by clicking the button (state **2**). State **T2** provides an example of how distinguishing touch vs. mechanical actuation can be useful to differentiate dragging states.

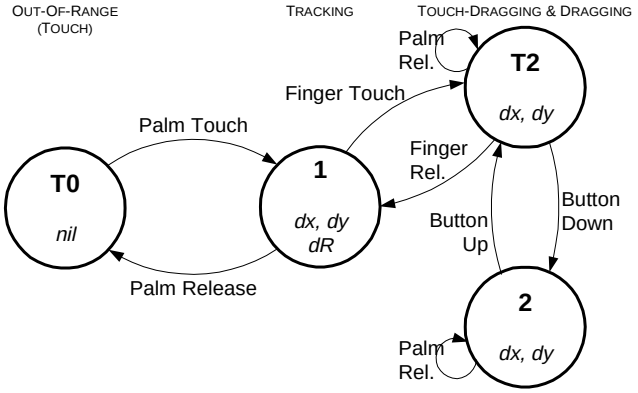

Fig. 8 The TouchMouse. State **T2** is the "touch-dragging" state supported by this device.

The PadMouse [1] offers a second example of state **T2** (*Fig. 9*). This is also our first example of a device state that can sense multiple pairs of coordinates; we indicate that the touchpad coordinates *x, y* are not necessarily in the same coordinate system as the mouse coordinates *dx, dy* by distinguishing them with a prime symbol: *x', y'*. For an input device such as the Wacom tablet, which can sense multiple devices (puck and stylus) registered to the *same* coordinate system, the prime symbol would be omitted.

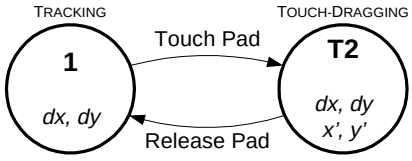

Fig. 9 The PadMouse [1] also supports state **T2**, since the touchpad can sense finger contact.

DESCRIBING TWO-HANDED TRANSACTIONS

The state models that we have considered so far handle individual devices well, but they are less effective when one considers two-handed interaction techniques. For example, the puck and stylus on the Wacom tablet each support all three states of Buxton's model. To avoid ambiguity, we name the states for the stylus $\mathbf{P0}_p$, $\mathbf{1}_p$, and $\mathbf{2}_p$ (**p** subscript for preferred-hand usage), and the states for the

puck **P0**$_n$, **1**$_n$, and **2**$_n$ (**n** for nonpreferred-hand). The cross product yields a state model with a total of 9 compound states, shown in Fig. 10:

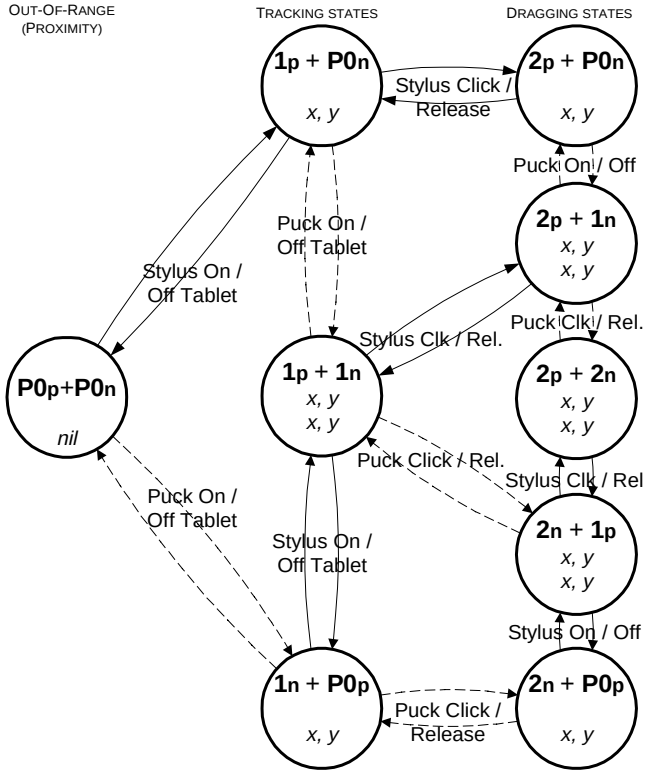

Fig. 10 Full state model for Wacom puck and stylus. The dashed arcs show events for the puck, solid arcs events for the stylus. It is hard to see how this model corresponds to the individual 3-state models for the puck and stylus.

Although this straightforward state model provides a technically correct model of the puck and stylus, we find it unwieldy because:

- It is hard to see how the states correspond to the original 3-state models for each individual device.
- It does not express the inherent parallelism of two hands with two devices.
- "Extra" states that may not be of interest for the interaction technique must be included. For example, in Fig. 10 states **1**$_p$ **+ P0**$_n$, **1**$_p$ **+ 1**$_n$, and **1**$_n$ **+ P0**$_p$ are all just variations of states that track one or both cursors.

Petri net model

Here, we propose a new model which is essentially a special case of a Petri net model (see Tanenbaum for a quick review [22]). The Petri net model is appropriate for thinking about two-handed interaction design problems in terms of device states and input events, and is intended for "whiteboard design" sessions where one is hypothesizing about what transitions between states would be most effective. Unlike the state model shown in Fig. 10, the Petri net model preserves much of the flavor of the 3-state model while also explicitly representing the inherent parallelism of two-handed input. Note that we use the Petri net model to describe composite two-handed *interaction techniques*, rather than the specific capabilities of the individual *devices*; in practice, we often use our Petri net model side by side with the 3-state diagrams for the individual devices.

The Petri net model starts with a set of desired states that a two-handed interactive technique is to support. These states always contain exactly two tokens, a black token (representing the preferred-hand device) and a gray token (representing the nonpreferred-hand device). Two rules govern movement of the tokens:

(1) The black token can only visit states with a solid border; the gray token can only visit states with a dashed border. That is, the borders indicate which token(s) are allowed to visit a particular state.

(2) A token may traverse any arc, either dashed or solid, as long as the arc leads to a state the token is allowed to visit (as in rule (1) above). The coloring (solid or dashed) of the arcs indicates which input device generates the signal corresponding to that arc.

Let's start with a very simple example for puck and stylus devices that do not interact at all (*Fig. 11*):

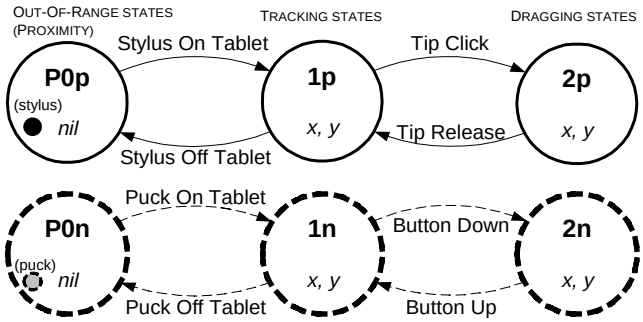

Fig. 11 Simple model for puck and stylus on the tablet. The tokens start in states **P0**$_p$ and **P0**$_n$, indicating that neither device is in proximity. We label the tokens here for clarity.

Fig. 11 shows how the states in our Petri net model correspond directly to the 3-state models for the individual devices. It also shows how the Petri net represents the parallel nature of the multiple devices by using a separate token to represent the current state for each device. The tokens move through states in response to device events. For example, placing the stylus on the tablet moves the black token to state **1**$_p$, as shown below (*Fig. 12*):

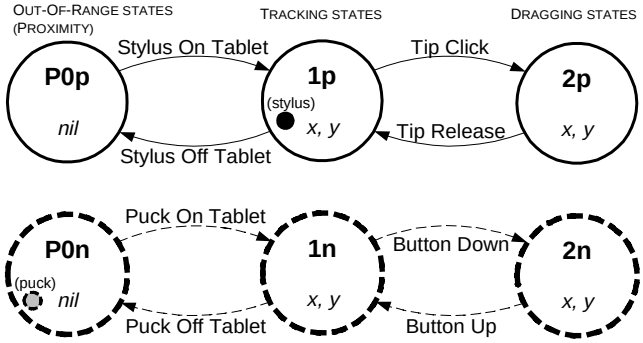

Fig. 12 Movement of the black token in response to placing the stylus on tablet.

Now let's refine the example of Fig. 11 to incorporate two-handed dragging, where the puck and stylus *do* interact. We introduce one additional state to the model, state $\mathbf{2}_{np}$, which is a dragging state using both the nonpreferred-hand device (puck) and the preferred-hand device (stylus). State $\mathbf{2}_{np}$ is the only state in the diagram that *both* the black and the gray token can enter; thus state $\mathbf{2}_{np}$ is colored with both a solid and a dashed border. We also add several arcs from state $\mathbf{2}_{np}$ to indicate how the interaction technique maps device events to allow the user to start and stop two-handed dragging (*Fig. 13*):

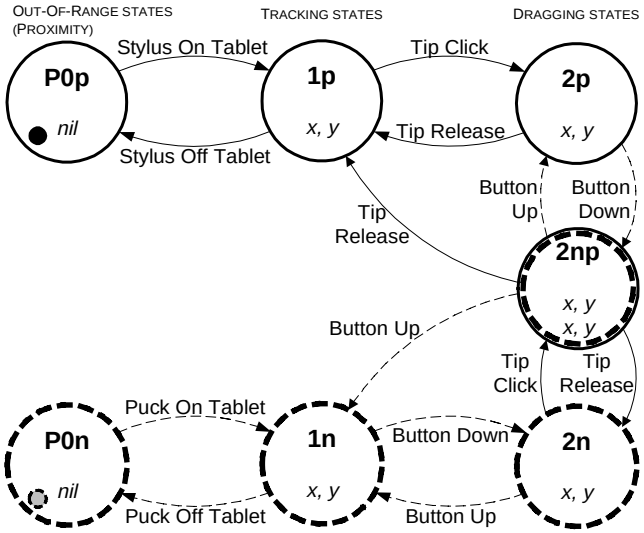

Fig. 13 Complete Petri net model for puck and stylus showing an additional state ($\mathbf{2}_{np}$) that models the interaction of the devices for two-handed dragging.

To see how this works, let's assume the user is currently dragging with both puck and stylus (black and gray tokens) in state $\mathbf{2}_{np}$; thus the tip of the stylus is depressed (clicked) and the puck button is held down. Relaxing pressure on the stylus causes the tip to release, and thus the *black* token moves from $\mathbf{2}_{np}$ to $\mathbf{1}_p$ (following the **Tip Release** arc). The gray token cannot follow this arc because it is not allowed to move to a solid-colored state. Relaxing pressure on the stylus also causes the *gray* token to move from $\mathbf{2}_{np}$ to $\mathbf{2}_n$ (following the shorter **Tip Release** arc). The black token cannot follow this arc because it leads to a dashed state, and only the gray token is allowed to enter the dashed states. This example shows how a single input event can potentially cause both tokens to move.

So, after relaxing pressure on the stylus, the black token is in state $\mathbf{1}_p$ and the gray token is in state $\mathbf{2}_n$; this corresponds to dragging with the puck only. From this point, if we once again click with the stylus, the black token moves from $\mathbf{1}_p$ to $\mathbf{2}_p$, and then directly to $\mathbf{2}_{np}$ (because the puck button is still down, and thus the token can move across the Button Down arc as well). This example shows how a token might pass through an intermediary state on the way to its final destination state. In parallel, the gray token moves from $\mathbf{2}_n$ to $\mathbf{2}_{np}$, and both tokens are once again in state $\mathbf{2}_{np}$ indicating a two-handed dragging operation.

Modeling the TouchMouse + Touchpad

We now model the interactions between the TouchMouse and touchpad for our two-handed interaction technique. The touchpad supports 2 states ($\mathbf{T0}_n$ and $\mathbf{2}_n$) and the TouchMouse supports 4 states ($\mathbf{T0}_p$, $\mathbf{1}_p$, $\mathbf{T2}_p$, and $\mathbf{2}_p$), as shown in Fig. 14. We use dragging ($\mathbf{2}_n$), rather than cursor tracking ($\mathbf{1}_p$), as the default state for touchpad contact since our technique uses touchpad contact to pan (drag) the map:

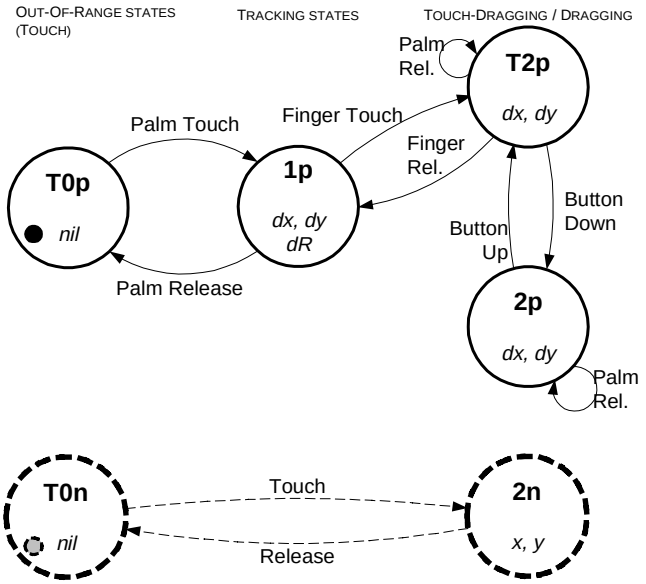

Fig. 14 Simple Petri net model for device states of the TouchMouse (top 4 states) and touchpad (bottom 2 states).

Our touchpad + TouchMouse interaction technique augments these basic device states with two additional states: $\mathbf{1}_n$ for cursor tracking with the touchpad, and $\mathbf{2}_{np}$ for two-handed dragging with both touchpad and TouchMouse. First, let's add state $\mathbf{1}_n$ to these basic device states (*Fig. 15*):

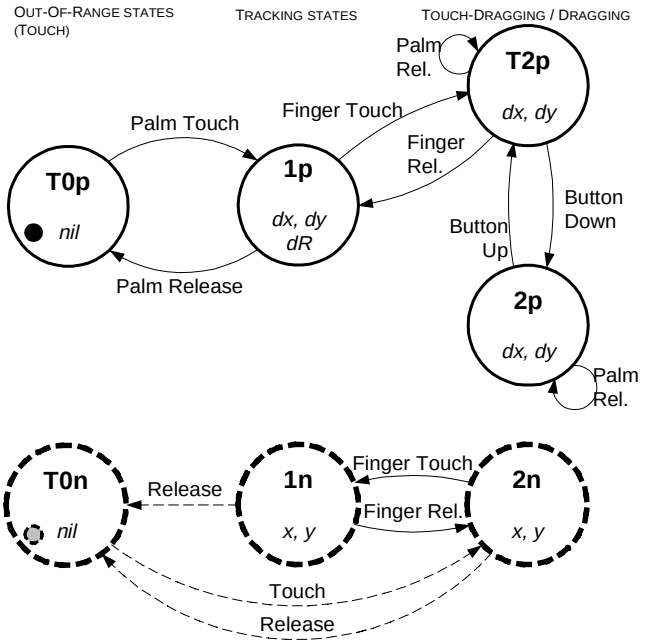

Fig. 15 Adding state 1_n to the device states to support both cursor tracking and dragging with the touchpad.

To see how the tokens move in this example, suppose the user grabs the mouse (moving the black token from $\mathbf{T0}_p$ to $\mathbf{1}_p$) and then touches the finger sensor (moving the black token from $\mathbf{1}_p$ to $\mathbf{T2}_p$). If the user then touches the touchpad, the gray token moves from $\mathbf{T0}_n$ to $\mathbf{2}_n$, and since the preferred hand is still touching the finger sensor, the gray token immediately continues along the **Finger Touch** arc to state $\mathbf{1}_n$., initiating the cursor tracking state with the touchpad.

Finally, let's incorporate two-handed dragging (state $\mathbf{2}_{np}$) into the Petri net model for the TouchMouse and touchpad (*Fig. 16*). This model provides a full specification of the device states and events used in our two-handed TouchMouse + touchpad map navigation technique:

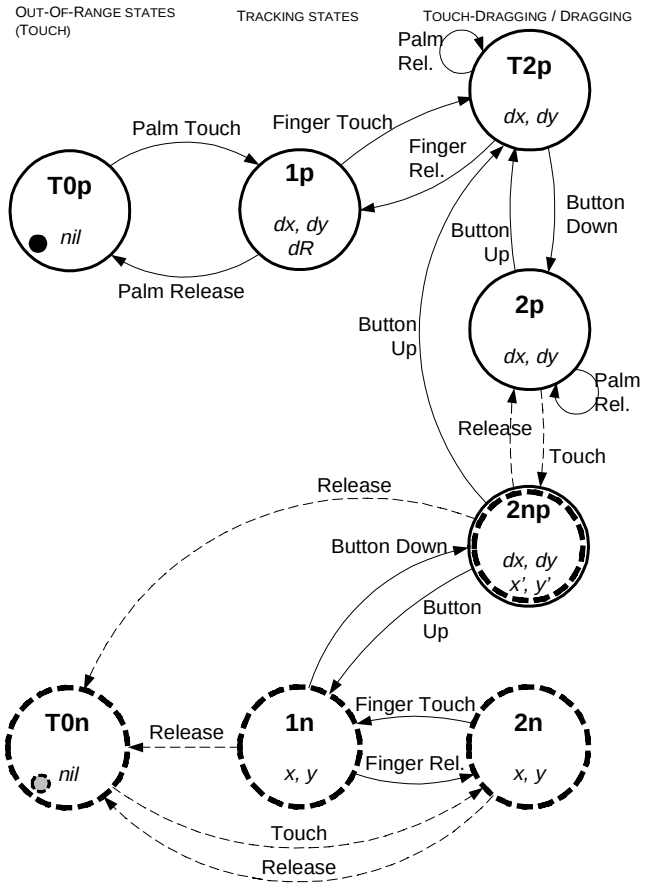

Fig. 16 Complete Petri net model for all states and events in our TouchMouse + touchpad interaction technique.

CONCLUSIONS AND FUTURE WORK

The various examples above provide specific instances of how our Petri net model can be applied, but in future work we still need to flesh out this model and characterize the design options for two-handed interaction techniques *in general*. Also, our Petri net model doesn't help the designer to see what pairs of states are possible (or not possible). For example, in Fig. 16, tokens cannot be in states $\mathbf{2}_n$ and $\mathbf{2}_p$ at the same time. As another example a token can only reach state $\mathbf{2}_{np}$ when it is accompanied by the second token. For some tasks, such analyses might be important.

We have described several new interaction techniques and input devices to support user interfaces that use both hands. The current demonstration uses map navigation as an example application for these techniques, but we would like to explore how these ideas generalize to other applications.

We need to perform further user testing with our designs to get a better sense of their strengths and limitations. We would also like to explore alternative input devices to support two-handed input, such as a trackball instead of a touchpad, as well as additional configurations of touch sensors and touch-sensing input devices. Experimental studies are needed to analyze the factors influencing two-handed input and touch-sensing devices in general.

It may be possible to further extend the interaction models presented here with quantitative data. For example, one could add "weighting factors" to state transitions to quantify the cognitive and motor costs of an interaction technique. One could also quantify the extent to which maintaining a particular state interferes with device motion (such as holding down a button while moving the mouse); an approach such as that described by MacKenzie may be appropriate here [17][18]. A comprehensive model might also incorporate the extent to which multiple continuous degrees of freedom in a single state can interfere with one another, perhaps using a coordination metric such as that introduced by Zhai [25]. Such a model might unify these various performance metrics, if possible, with a state transition model to help characterize an input technique's overall performance.

ACKNOWLEDGEMENTS

We would like to thank the Brown University Computer Graphics Group for letting us use their Wacom tablet driver; Synaptics for touchpad samples; the Geography products unit for their help with interfacing to Microsoft's commercial mapping engine; the Hardware products group for ideas and discussions; Dave Thiel for the audio feedback; Dan Robbins for photographs; Maarten van Dantzich and Jeff Pierce for reviews of this paper; and George Robertson for managerial support and design discussions.

REFERENCES

1. Balakrishnan, R., Patel, P., "The PadMouse: Facilitating Selection and Spatial Positioning for the Non-Dominant Hand," CHI'98, 1998, 9-16.

2. Bier, E., Stone, M., Pier, K., Buxton, W., DeRose, T., "Toolglass and Magic Lenses: The See-Through Interface," Proceedings of SIGGRAPH 93, 1993, 73-80.

3. Buxton, W., "Chunking and Phrasing and the Design of Human-Computer Dialogues," Information Processing '86, Proc. of the IFIP 10th World Computer Congress, 1986, 475-480.

4. Buxton, W., "A three-state model of graphical input," Proc. INTERACT'90, 1990, 449-456.

5. Buxton, W., "Touch, Gesture, and Marking," in *Readings in Human-Computer Interaction: Toward the Year 2000*, R. Baecker, *et al.*, Editors. 1995, Morgan Kaufmann Publishers. p. 469-482.

6. Buxton, W., Hill, R., Rowley, P., "Issues and Techniques in Touch-Sensitive Tablet Input," Computer Graphics, 19 (3): p. 215-224, 1985.

7. Buxton, W., Myers, B., "A Study in Two-Handed Input," Proceedings of CHI'86: ACM Conference on Human Factors in Computing Systems, 1986, 321-326.

8. Fitzmaurice, G., Ishii, H., Buxton, W., "Bricks: Laying the Foundations for Graspable User Interfaces," Proceedings of CHI'95: ACM Conference on Human Factors in Computing Systems, 1995, 442-449.

9. Guiard, Y., "Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model," The Journal of Motor Behavior, 19 (4): p. 486-517, 1987.

10. Harrison, B., Fishkin, K., Gujar, A., Mochon, C., Want, R., "Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces," CHI'98, 1998, 17-24.

11. Hinckley, K., Pausch, R., Goble, J., Kassell, N., "Passive real-world interface props for neurosurgical visualization," Proceedings of CHI'94: ACM Conference on Human Factors in Computing Systems, 1994, 452-458.

12. Hinckley, K., Pausch, R., Proffitt, D., Kassell, N., "Attention and Visual Feedback: The Bimanual Frame-of-Reference," ACM/SIGGRAPH Symposium on Interactive 3D Graphics, 1997, 121-126.

13. Hinckley, K., Pausch, R., Proffitt, D., Patten, J., Kassell, N., "Cooperative Bimanual Action," Proceedings of CHI'97: ACM Conference on Human Factors in Computing Systems, 1997, 27-34.

14. Horvitz, E., Breese, J., Heckerman, D., Hovel, D., Rommelse, K., "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, July 1998, 256-265. Morgan Kaufmann: San Francisco.

15. Kabbash, P., Buxton, W., Sellen, A., "Two-handed input in a compound task," Proceedings of CHI'94: ACM Conference on Human Factors in Computing Systems, 1994, 417-423.

16. Kurtenbach, G., Fitzmaurice, G., Baudel, T., Buxton, B., "The Design of a GUI Paradigm based on Tablets, Two-hands, and Transparency," Proceedings of CHI'97: ACM Conference on Human Factors in Computing Systems, 1997, 35-42.

17. MacKenzie, I.S., Oniszczak, A., "A Comparison of Three Selection Techniques for Touchpads," CHI'98, 1998, 336-343.

18. MacKenzie, I.S., Sellen, A., Buxton, W., "A comparison of input devices in elemental pointing and dragging tasks," CHI '91, 1991, 161-166.

19. Mackinlay, J., Card, S., Robertson, G., "A Semantic Analysis of the Design Space of Input Devices," Human-Computer Interaction, 5: p. 145-190, 1991.

20. Matsushita, N., Rekimoto, J., "Holo Wall: Designing a Finger, Hand, Body, and Object Sensitive Wall," Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology, 1997, 209-210.

21. Smith, J. R., White, T., Dodge, C., Allport, D., Paradiso, J., Gershenfeld, N., "Electric Field Sensing for Graphical Interfaces," IEEE Computer Graphics and Applications, May, 1998.

22. Tanenbaum, A., "Computer Networks". 1989, Englewood Cliffs, NJ: Prentice Hall.

23. Ullmer, B., Ishii, H., "The metaDESK: Models and Prototypes for Tangible User Interfaces," Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology, 1997, 223-232.

24. Zeleznik, R., Forsberg, A., Strauss, P., "Two pointer input for 3D interaction.," ACM/SIGGRAPH Symposium on Interactive 3D Graphics, 1997, 115-120.

25. Zhai, S., Milgram, P., "Quantifying Coordination in Multiple DOF Movement and Its Application to Evaluating 6 DOF Input Devices," CHI'98, 1998, 320-327.

26. Zimmerman, T., Smith, J. R., Paradiso, J., Allport, D., Gershenfeld, N., "Applying Electric Field Sensing to Human-Computer Interfaces," Proceedings of CHI'95: ACM Conference on Human Factors in Computing Systems, 1995, 280-287.