Hands-On-Math Final Report – September 2010

Andries van Dam, Robert Zeleznik, Andrew Bragdon, Donnie Kendall, Ferdi Adeputra
Brown University

Summary report, September:

This report summarizes the work conducted under the 1 year Surface initiative project. Our goal was to explore the value proposition of adapting pen-centric mathematical sketching techniques to a hybrid pen and multi-touch environment based on a Microsoft Surface. Our results were positively received during evaluation as participants commented that our Hands-On Math prototype not only made interacting with math fun and efficient, but also provided value for note-taking tasks in general. This work has also been accepted as a full paper at UIST 2010.

Project Background:

Having developed pen-based interfaces for over a decade including pen-centric systems for mathematics, we believed that the pen was overloaded for many tasks which could naturally be off-loaded onto touch interfaces. Our aim with this project was to explore three areas, including: 1) technical solutions for sensing pen input on a Surface; 2) the design of a hardware abstraction layer API to unify pen and multi-touch input across multiple platforms; and 3) multi-touch techniques for exposing additional CAS (Computer Algebra System) functionality that complement our existing pen-centric math sketching UI. Our expectation was that such a hybrid system would provide a non-trivial example of the potential synergistic benefits of pen and multi-touch computing.

Work product: Pen Sensing on Surface + API

We evaluated several technologies for recognizing pen input on a Surface, including: a light pen, 2 ultrasonic pens, and an Anoto pen. Of these solutions, the light pen was the easiest to use because it required no calibration or additional sensors, although its performance would be insufficient for a real production system.

Anoto: Collaborating with Dr. Michael Haller of the Upper Austria University of Applied Sciences, Media Technology and Design/Digital Media, we determined that although the Anoto pen can provide the most accurate input at the highest sampling rate, it was not feasible for use on a Surface because the IR light that is emitted by the surface interferes with the IR sensing of the Anoto pen. Using the Anoto pen with our software on a front projected wall or surface produced excellent results.

Ultrasonic: We used both an eBeam ultrasonic pen and an IOGear pen (Figure 1). The eBeam pen is designed to generate native mouse input across large surfaces while the IOGear pen produces mouse input at a much higher sampling rate but is designed for 8.5"x11" surfaces and thus provides severely degraded input beyond this footprint. Both of these pens were able to provide mouse input that was robustly disambiguated from touch input, but both suffered from calibration issues where samples could be off by as much as an inch. In addition, these techniques support only a single pen and are subject to line-of-sight issues with receivers that must be mounted on the Surface.

Figure 1. eBeam and IOGear pens

Lightpens: We used both a Microsoft supplied IR light pen prototype and a commercial IR light pen (1). Both of these pens were comparable with the commercial pen being physically more robust, but the Microsoft pen being better tuned for the Surface with a felt tip and better balance. In both cases, light pen input was largely separable from touch input although false positive and negative recognitions occurred around 1% of the time. In addition, the sampling rate of these devices was rather low and limited by the camera speed to about 30fps maximum. When users were allowed to write mathematical expressions on the surface with either light pen, we noticed that the size of their writing was much larger than with pen and paper and that they held the light pen at

Figure 2. Commercial light pen

awkward angles to ensure that the tip-switch would activate. These problems were acceptable under demo conditions, but would have been highly undesirable in a product.

API support for pens, multi-touch, and desktop mouse/stylus events

To leverage our previous work in pen-based mathematics, we chose to build our prototype system using the starPad SDK. The starPad SDK is a toolkit developed at Brown under Microsoft's sponsorship which aims to simplify the development of gesture-based applications, particularly those which require recognition of handwritten mathematics. In addition, we chose to use the Microsoft Surface SDK because of its rich support for multi-touch interactions.

However, the Surface SDK introduces a new event type, ContactEvents, but does not generate either Mouse or Stylus events. Thus, any interaction libraries that respond to the Mouse or Stylus, such as those included in the starPad SDK, are not directly usable through the Surface SDK. To remedy this in a way that would facilitate component reuse across desktop, tablet and the Surface, we extended an input abstraction framework, provided by Michael Haller, to map Contact, Mouse and Stylus events to our a generic class of Point events. In this way, a widget can be written once to accept Point events and then it will work in a mouse, pen or touch environment without modification. At a higher level, we also extended the starPad SDK to provide interaction templates for touch-activated pen gestures and for a range of multi-touch gestures.

All of the code written for the Hands-On Math system is also part of the starPad SDK. A SurfaceLib within the starPad SDK contains classes corresponding to each of the interaction techniques used in the system, such as a PalmPrint, PieMenu, and several TAP gestures.

Work Product 2 - rich multi-touch and pen math application

The major focus of our research was to create a novel driving application a la Fred Brooks, Hands-On Math, that would showcase the Surface's unique qualities for improving mathematical sketching. Our approach to this system involved not just targeting interactions that specifically operated on mathematical expressions, but rather considered an end-to-end study process. The notion was to support workflows that involved reading, creating, and annotating documents within the unified space of a single pannable virtual desktop. To support the broad functionality required, we developed several novel input techniques to evaluate their individual potential and their value as part of a complete system. We hypothesize that the impact of such a system will be that students of all ages will not only work more efficiently, but will also gain deeper insights since the machinations of a CAS system are exposed as tangible, hopefully intuitive, direct manipulations.

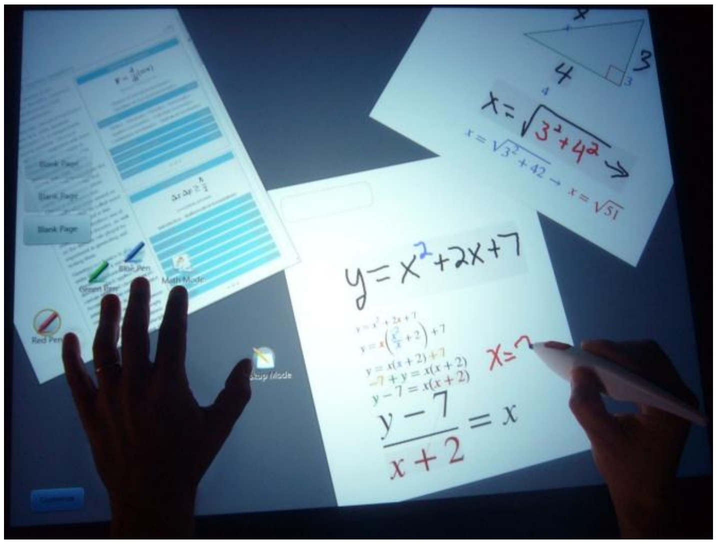

Figure 1. Document and note pages showing recognized math and embedded computation.

Thus, the specific contributions that we made include:

- The design of a virtual paper/CAS hybrid prototype that attempts to expose CAS tools fluidly as bimanual pen and touch operations (Figure 2).
- The design of several synergistic pen and touch techniques that collectively comprise a functionally rich page-based system capable of supporting domain users performing representative tasks (**Error! Reference source not found.**1).
- A qualitative pilot evaluation and usability discussion of the prototype system as a whole, and the novel bimanual pen and touch techniques developed for it

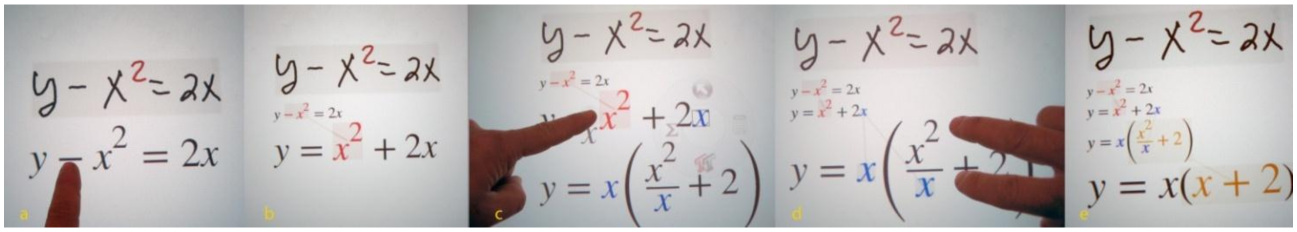

Figure 2. Math transformation. a) Dragging $-x^2$ across the equality. b) Result with terms highlighted. c) Dragging the blue x (from 2x) across other right-hand side terms factors it and interactively updates the result below. d) Squeezing x^2/x simplifies it. e) The final expression.

PAGES AND WORKSPACE

There are two dominant traditional environments for problem solving – paper and pencil and whiteboards – each with its own benefits. Whiteboards are convenient because often entire workflows can be captured without having to perform any spatial management; however, they quickly fill up and cannot be used like a notebook for longer term workflows. Paper on the other hand is non-interactive and can provide no active assistance. Thus, we designed a hybrid tabletop solution that blends whiteboard-like interactions with resizable pages that live on a large virtual desktop. Users can grow existing pages to accommodate complex problems or pan the desktop to make room for new work.

Beyond basic multi-touch interactions for dragging and rotating pages with one or more fingers, we designed three page management interactions, including a virtual desktop with panning bar, bezel gestures to create and delete pages, and a page "folding" gesture to make room for more work.

Page Management

To facilitate an exploratory mindset when problem solving, we wanted to minimize the cost of creating a new virtual page, so we associate it with a bezel gesture (Figure 3). Our bezel gestures are parameterized by the number of fingers that cross the bezel and which bezel is crossed. Swiping two fingers through the left or right bezel, as if reaching beyond the desktop for a new sheet of paper, creates a page. Swiping through with one finger pans the desktop. The physical distinctiveness of one versus two fingers along with bezel size enables eyes-free interaction

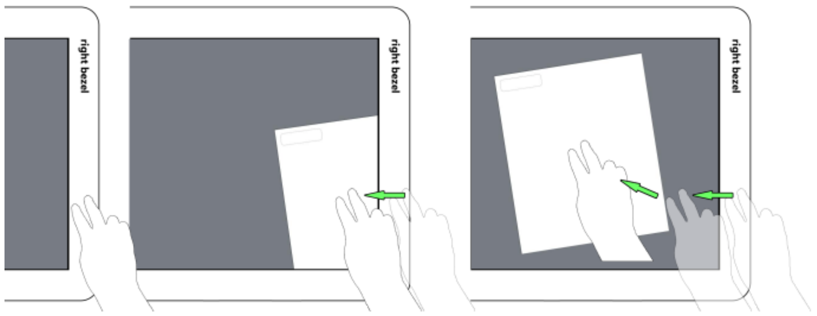

Figure 3. Swiping over bezel with 2 fingers creates a page.

Deleting a page requires a more deliberate gesture to avoid accidental triggering and to encourage pages to be "left around" on the virtual desktop as a paper trail of problem solving activity. After dragging a page into the bezel region, a trash can icon appears near where the finger contact left the screen; by continuing the drag back across the trashcan (the small grey icon in the middle figure), the page is deleted (See Figure 4).

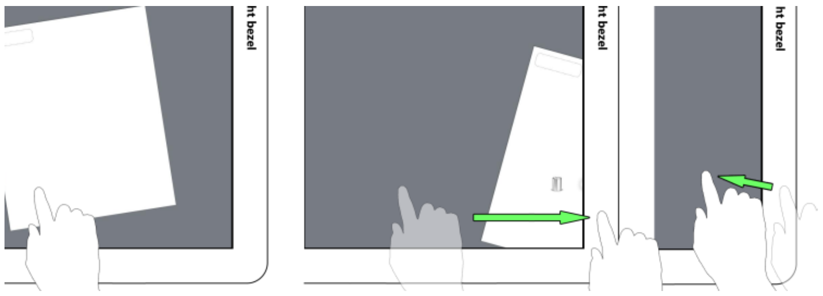

Figure 4. Dragging over bezel displays trashcan widget.

Since pages can be manipulated with one or two fingers, they are subject to accidental manipulation, particularly when writing if the palm of the writing hand may be mistaken as a finger contact. The palm rejection technique of ignoring large contacts partially addresses this problem, however, users may also explicitly hold a page steady with two or more fingers from their non-dominant hand while writing – similar to how people control physical pages.

Panning Bar

To free users from having to worry about what to get rid of in order to make room for new material, we adapted the notion of a continuous virtual desktop that was previously found useful for desktop code development. Unlike fixed-size desktops, where the clutter of overlapping pages becomes an interaction burden, continuous virtual desktops allow users to create more space on demand while preserving a spatial record of their previous work.

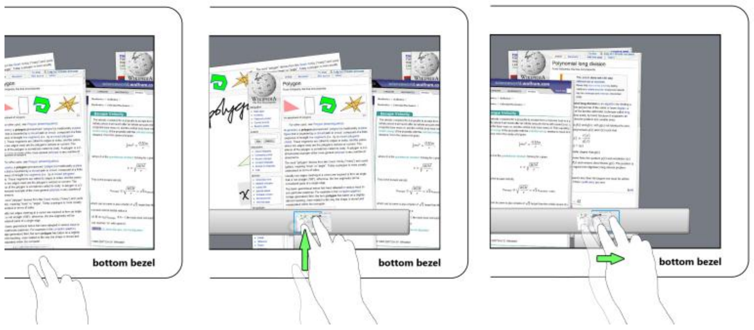

Figure 5. Swiping up through bezel with two fingers shows panning bar; continued dragging pans the desktop.

Although swiping through the bezel with one finger supports fine-grained panning of the virtual desktop, we wanted a complementary technique to facilitate larger scale navigation. After recognizing an upward two-finger swipe through the bottom bezel, we display an interactive panning bar (Figure). The panning bar displays a live, miniaturized panorama of the user's workspace that provides a visual overview of the user's work history. The currently visible desktop is always centered under the user's finger when the panning bar appears; as the user drags horizontally along the panning bar, the desktop scrolls correspondingly to that location. While one hand controls scrolling, the other hand can grab pages to relocate them within the virtual workspace. Invoking the panning bar requires a two-finger swipe to reduce the likelihood of unintentional triggering which can occur, for example, when leaning over the surface as accidental contacts are made with an arm, shirt, etc.

Folding

As a complement to the management of different pages, we support spatial management within a page by making page folding available with multi-touch gestures. Thus, for example, a user may choose to fold away part of a complex derivation in order to simultaneously view the problem statement and their current step in the derivation without having to scroll between the two. By pinching in the margin of a page, users interactively simulate a 3D fold in which the page buckles up between their fingers (Figure 6), before collapsing into a suggestive “crease” indicated with a soft shadow. Tapping the shadow line unfolds the page.

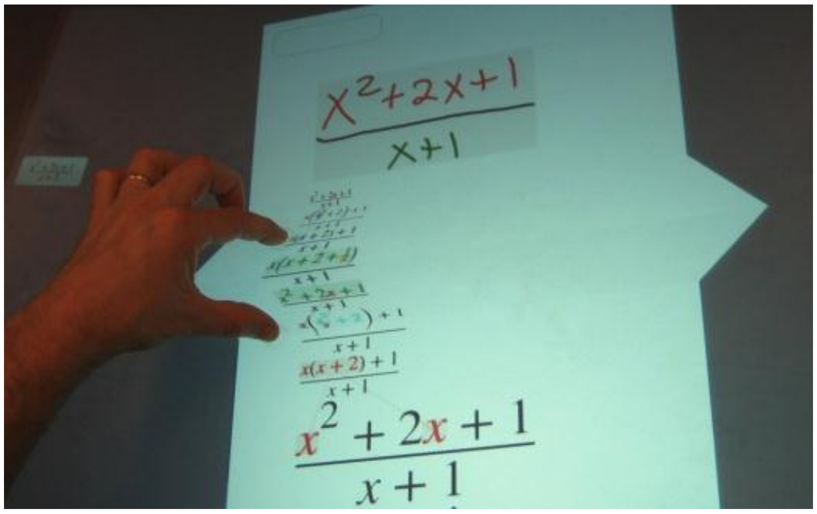

Figure 6. Folding page contents to make space on the page

Since this technique is intended to support complex work, it does not actually shorten the page the way a real page fold would, but instead causes the page contents to slide up and create open space at the bottom of the page. This operation is akin to code elision techniques in source code editors except it does not operate on syntactic structures.

GESTURES

Inherent ambiguities exist when trying to decide whether input is intended to be ink, a gesture or a direct manipulation. To support fluid command invocation and mode switching, we developed three complementary gesture techniques which have minimal overlap with other activities and thus potentially can be robustly recognized.

Under-the-Rock menus

Under-the-rock menus are a general purpose mechanism for associating contextual actions with display elements. In essence, they are context menus that are “hidden under the rock”, only to appear when objects (*i.e.*, rocks) are moved. For example, dragging a term in a mathematical expression might default to a factoring operation; the under the rock menu for that term, however, would allow the interaction to be changed to reordering, term splitting, or something else.

An under-the-rock radial menu grows, after an initial lag, out from an object’s initial location as the object is dragged away (Figure 7). Growing a semi-opaque menu from the start of a drag provides unobtrusive disclosure which is critical for not interfering with a default dragging behavior. The menu is only activated and made fully opaque after a second contact is made over its center; menu items can then be selected by sliding the second contact over them. An active menu can be deactivated by sliding the second contact back to the menu center and releasing. Since the menu appears predictably, centered on the drag’s starting point, trained users can anticipate this and co-articulate menu selection with dragging before the menu has become fully opaque as with marking menus. Depending on manual dexterity, this interaction can be performed with one hand.

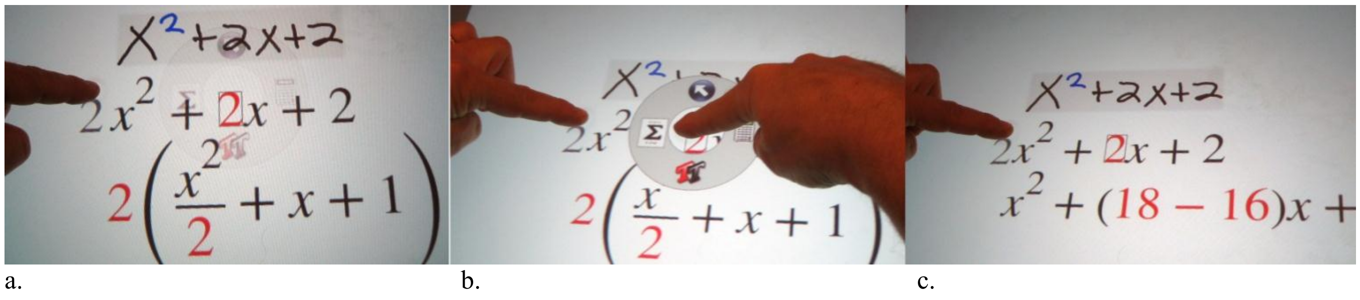

Figure 7. Under-the-rock menus. a) The 2 in the 2x term is dragged as a semi-transparent ghost until it is in front of the leading x^2 – this factors it from the expression. As the user drags, a semi-opaque radial menu grows to allow the user to change the semantics of the interaction. b) Touching the menu center makes it opaque and activates it, and dragging over the ∑ icon converts the default factoring interaction into a ‘split into sum’ interaction. c) Now, as the user drags, the number ‘2’ changes and a negative term is added to ensure that the sum of the two terms is unchanged.

Touch-Activated Pen Gestures

Pen gestures are often touted as being well suited for invoking spatially parameterized commands. However, pen gestures are by definition ambiguous with regular inking activities unless distinguished through special hardware buttons (e.g., stylus-mounted, or external buttons). In our previous work, a set of pen-only gestures (for scribbling out, selecting, graphing and performing undo and redo) was designed and evaluated to be compatible with writing mathematical expressions. Still, in a general inking context, loosely defined pen-only commands, such as lasso selection and scribble deletion, are likely to conflict with regular inking and thus require additional input for disambiguation. Instead, hybrid pen and touch gestures can be readily disambiguated from isolated ink and touch activities and enable fluid direct manipulation transitions.

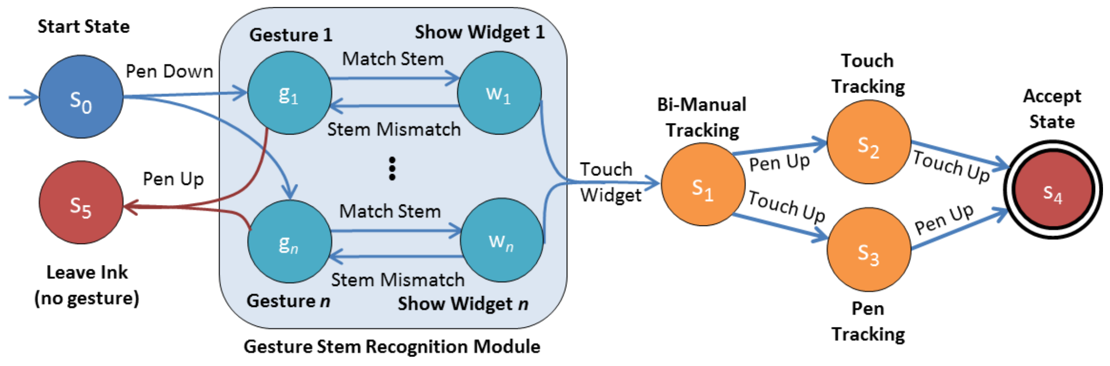

Figure 8. TAP gesture FSA. When the pen goes down, a set of gesture recognizers continuously monitor the pen input to see if it matches their gesture stem. When one or more stems are recognized, corresponding feedforward widgets are displayed. When a feedforward widget is touched, it triggers recognition of that gesture to the exclusion of all others. The recognized gesture can then provide a bi-manual direct manipulation state, followed by either a touch or pen-only manipulation state before the gesture is completed.

Our solution, touch-activated pen (TAP) gestures, recognizes pen gesture stems on-the-fly as they are drawn (Figure 8). A gesture stem is a non-trivial subset of a complete pen gesture. Recognizing a gesture stem performs no command but rather introduces a feedforward widget(s) that can perform a command if triggered by concurrent touch input. If the pen stroke ends without touch input or if the stem no longer matches, the widget disappears. Triggering a TAP gesture with touch input allows fluid transitions to bi-manual, pen-only, or touch-only interactions. False recognition of TAP gestures is unlikely if widgets are positioned away from the hand holding the pen. In addition to improved recognition accuracy, TAP gestures are more scalable than pen-only gestures since a similar or even a single gesture stem can trigger multiple non-overlapping feedforward widgets. To avoid the “noise” of unintended feedfoward during regular inking, all widgets are displayed at reduced opacity until activated by touch input.

Example 1: Making 2D selections

Although users can lasso ink by drawing an enclosed loop, there are times when this is inconvenient or inappropriate such as when selecting large regions of ink or when making a rectangular image clipping. Thus, we support an additional gestural selection technique: as a user draws a crop mark, a semi-transparent dashed rectangle appears in registration with the crop mark (Figure 9). The user can ignore this rectangle and keep drawing, or by dragging a finger across it switch from inking to rubber-banding a rectangular selection marquee. Since the pen and finger-touches control opposite corners of the marquee, its position and size can be adjusted simultaneously. For larger selections, this TAP gesture also provides the benefit of requiring less movement than a corresponding lasso would.

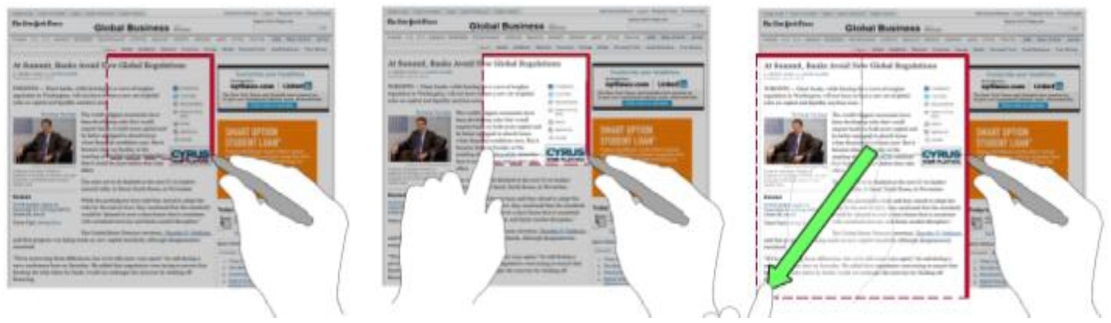

Figure 9. Drawing a crop mark (red polyline from left to right and then down) displays a feedforward selection/clipping marquee widget. Dragging the marquee while drawing switches to bi-manual selection.

Example 2: Inserting space

When problem solving, users often are unable to plan ahead spatially for their future notations and find themselves in need of inserting space between existing notations (and occasionally wanting to remove space). In these cases, they need to specify where and how the space should be created (*e.g*., vertically or horizontally, across the whole page or locally). We provide a TAP gesture for space insertion (Figure 10): as the user draws a straight line in any orientation, space insertion widgets appear on either side of the line starting point for selecting the page contents on either side of the line respectively. If the user ends their ink stroke, the widget disappears. However, if the user touches either widget, the drawn ink becomes a “push-bar”. Moving both ends of the push-bar is like manipulating the top (or bottom) corners of a picture frame where the page contents below (or above) the push-bar is the picture. If either contact is released, then the push-bar is constrained to move its contents along one axis.

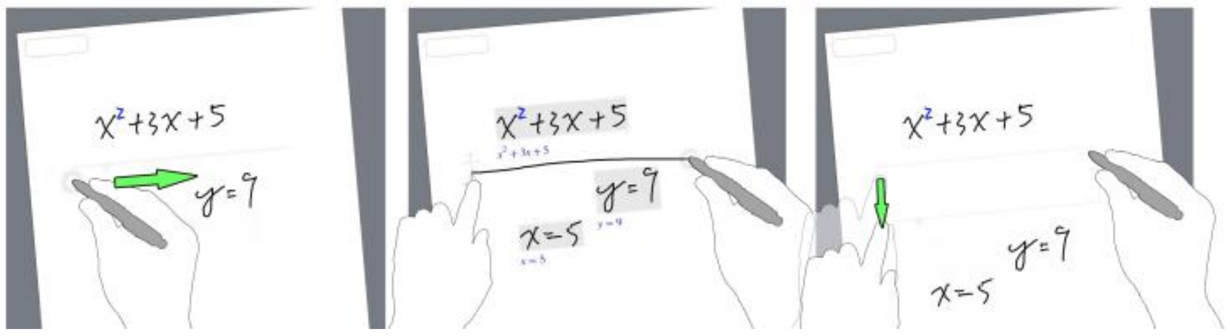

Figure 10. Drawing a line displays an insert space widget. Dragging the widget inserts or removes white space.

Example 3: Clipboard pasting

We also provide a TAP gesture for pasting clipboard contents. As a ‘p’ is drawn, we display a paste icon. If the user taps this icon, the ‘p’ disappears and the clipboard contents are pasted. If instead the paste icon is dragged, then the clipboard contents are interactively adjusted to fit within the boundary defined by the pen and touch contacts.

PalmPrints

Although gestures are efficient for executing many types of commands, they require significantly more effort than simply pressing a button. In situations where efficiency is important, such as when switching between different pen colors when drawing a diagram or alternating between math notations and drawing elements, the overhead of performing a path-based gesture (or locating and clicking a toolbar item) can be burdensome. In such cases, having the desired functionality on a button beneath one's fingertips on the non-dominant hand is both efficient and robust.

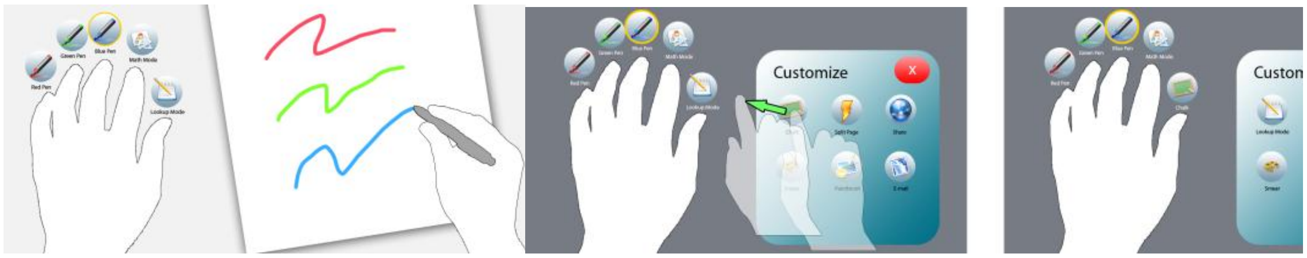

Figure 11. Placing palm and fingers on surface activates PalmPrint menu. Lifting and tapping a finger changes the pen mode. Chording is also possible. A Customize palette allows drag and remapping of functions

We thus developed PalmPrints which are similar to finger-tapping techniques but which instead activate implicitly when an open hand is placed on the surface (Figure 11) and deactivate when it is lifted to do something else. While the users palm rests on the surface, up-down fingertip transition are recognized to invoke a command associated with that finger. Identifying each fingertip is done by sorting the initial five fingertip contacts according to their radial angle relative to the larger palm contact(s). When a fingertip contact is lost, we trivially know which finger was lifted. When all lost contacts are regained (as new contacts), a chord is triggered that executes the commands associated with each finger that had been lifted.

By associating functionality with each fingertip, users can execute commands without looking; however, we display an icon above each fingertip for disclosure (Figure 11). Dragging and dropping functions from a customization palette onto an icon reassigns that fingertip's command.

Alternatively, the user can use their primary hand to "lock" their PalmPrint on the display. If the user then lifts their hand after having locked the PalmPrint, the PalmPrint will transform itself into a five-item toolbar. This toolbar can be dragged with either hand similar to a Toolglass. The toolbar can be dismissed with a tap, or it can be restored to a PalmPrint if the user simply places their hand back down on the surface in the registration pose.

FingerPose

Since touch-and-drag is preferable to the more heavyweight bi-manual PalmPrint for common actions such as dragging a window or panning its contents, we created FingerPose which selects one of two input modes based on vertical finger posture.

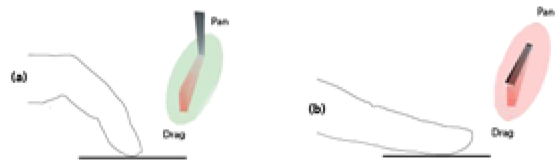

Figure 12. Finger posture selects a drag mode. A pause-activated widget reveals functionality and thresholds.

FingerPose estimates the posture of a finger based on *initial* contact geometry to select between two manipulation functions (Figure 12). By using one-finger FingerPose for window dragging, and content scrolling, and two-finger pinching for zooming, each function is isolated and inadvertent manipulations are unlikely. The fingertip posture is recognized as a contact with a physical area below a calibrated threshold, otherwise a finger pad contact is recognized. Since contact area is dynamic, particularly just after the finger touches the surface, we do not distinguish between poses until after the contact has moved by 25 pixels. In addition, if the contact has not moved after 100ms, we display a dynamic 3D representation of the recognized “posture” and its associated function. Finger posture is ignored after dragging begins.

Math

Using existing SDKs, we support math-specific interactions for writing mathematical expressions, computing values with extended mathematical notations, and creating graphs with gestures. In addition, we extended this work with a suite of multi-touch interactions for transforming mathematical expressions both to compute solutions and to gain insight on the problem domain and the process of solving problems – concerns that apply equally to students and experienced mathematicians. From a mathematics perspective, our goal was to extend the functionality of previous pen-based math systems to support algebraic manipulations that are fundamental to mathematical reasoning but which are generally hidden by CAS systems. Although pen-based manipulation techniques are possible, we felt that multi-touch interactions were more suitable for the manipulation nature of mathematical transformations. Unlike pen interactions, which inherently conflict with writing mathematics and require a compound selection/manipulation interaction, multi-touch gestures have the promise of integrating selection with manipulation in a single, memorable and efficient physical action. For example, to join two additive terms, users can just pinch them together; a hypothetical pen-based counterpart would need to select terms and indicate the join operation (and not conflict with the entry of new math notations) before initiating direct manipulation feedback. Nonetheless, the set of possible mathematical operations is large, requiring subtlety to increase the expressiveness of multi-touch manipulation.

Fingertip Area Selections

Tapping an expression activates it for manipulation, and increases its font size to facilitate syntax-aware, contact area-based selection; touching a symbol selects it subject to convenience shortcuts based on how the fingertip contact regions intersect mathematical symbols (Figure 13).

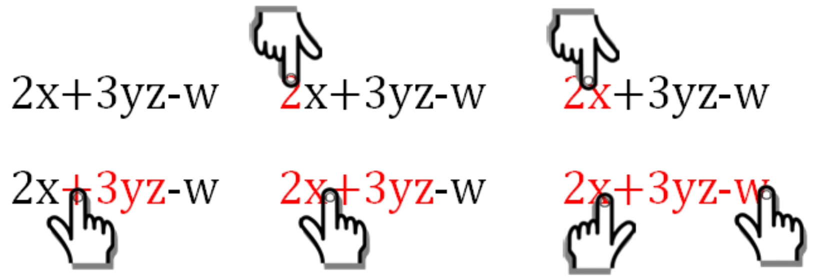

Figure 13. Fingertip area selection: 1) one symbol; 2) two symbols by touching both; 3) term by touching operator; 4) neighboring terms by touching part of one and operator; 5) a long span.

Touch Manipulations

The interface for performing mathematical transformations consists of sequences of pinch, drag, and stretch direct manipulations of terms within an expression (Figure 1), as summarized in Table 1. The associated algebraic transformations are naturally parameterized by where the terms are dragged. Thus, to preserve context, we do not modify the touched expression directly. Instead, "ghost" copies of just the touched symbols follow the user's finger contacts while the resulting transformed expression interactively updates below the original. Since there are more math transformations than unique input affordances, mode switching is required. To reduce dependency on an explicit mode switching UI, we select different default manipulations based on what was touched. For example, dragging an operator defaults to reordering the term on its right, whereas, dragging a variable(s) defaults to successively factoring it out of the expressions it passes over. During all manipulations, an under-the-rock menu grows out from the original location of the dragged symbol – when needed, this menu can override the default manipulation. The limited expressivity of touch input makes it easy to start "playing with math" but also demands the adoption of interaction strategies. For example, to simplify x+y+x,one x must be moved next to the other before a pinch can join them.

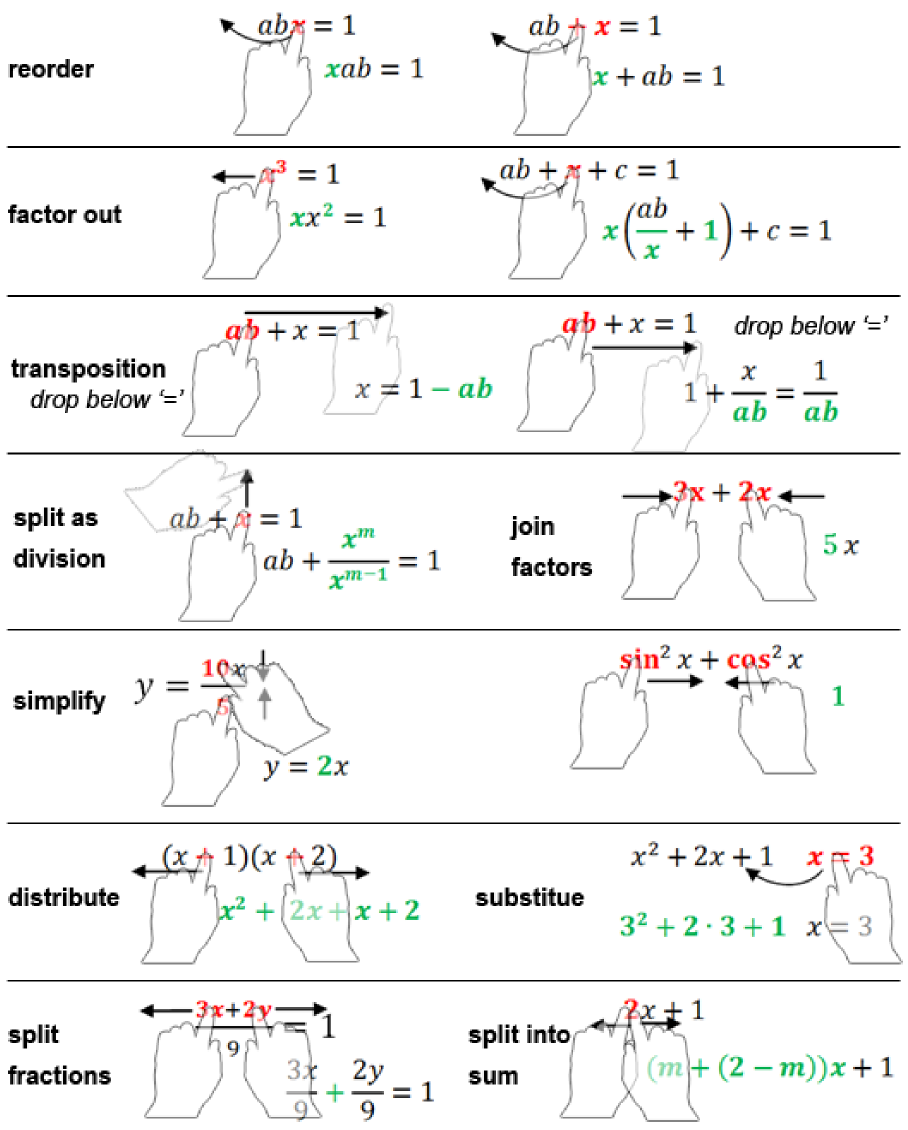

Table 1. Multi-touch algebraic identity transformation UI. Manipulated terms are highlighted in red. Resulting expressions are shown below with modified terms in green.

Although users may enjoy learning the math transformation UI by playfully exploring "what happens if" scenarios, a technique such as GestureBar could provide additional, more efficient, disclosure of functionality and strategies.

Visual Feedback

Since the structure of an expression may change significantly as the result of a single transformation, we did not feel that it was possible to show the results of a transformation in-place with the original expression. In addition, since a transformation may produce a result that is structurally quite different, we felt that it was important to leave a visual trail that would explicitly show what happened in a transformation step. Thus, as users manipulate a term, the resulting transformation is displayed directly below. We use an arrow-like visualization in combination with colorization and shading cues to depict how terms have changed from one transformation step to the next (Figure 1).

Pilot Evaluation

To gain insight into the utility of Hands-On Math for domain users and to gain feedback regarding its techniques and features, we recruited 9 students from the undergraduate population of Brown University. All students made some use of mathematics in their coursework. In addition to letting participants play with the system, each was asked to perform a set of exploratory tasks:

- Create and manipulate pages
- Perform a "back of the envelope" calculation
- Solve a more complex math expression via a multi-step derivation
- Graph an equation and manipulate the result
- Use the PalmPrint to change colors while drawing a diagram
- Make a web clipping
- Manipulate the contents of a page with TAP gestures and page folding.

Discussion and future directions

Our central hypothesis, that people would learn and work more efficiently if CAS functionality were available in a paper-like environment, was supported by our pilot evaluations. Users were enthusiastic about the potential of the system to let them concentrate on mathematics problem solving by removing the burdensome and error-prone steps of transcribing equations or missing intermediate steps. They also were in unanimous agreement, unlike paper and pencil, they felt they would be less likely to make "stupid mistakes" and would be better able to take chances while exploring more complex problems. The ability of our system to support free-form note taking, symbolic and numerical computation, graphing, and function transformation all "without" a UI led participants to conclude that the system has "great potential" not just in their math-oriented classes, but also as a study platform in general. The most significant perceived obstacle to adoption was the bulky, non-portable form factor of our Microsoft Surface hardware and the low quality of light pen input, as compared to physical pens or even Tablet PC ink. It was also clear that extending the set of possible mathematical operations should be a high priority. We also feel important recognition techniques still need to be addressed, for example, to interpret math written on angled baselines, to automatically distinguish mathematics notations from diagrams and free-form inking, and to recognize, anchor and track annotations of typeset terms and symbols.

***Page metaphor*.** We also found that the choice of using manipulable pages as a primary UI element, as opposed to a whiteboard or book metaphor, appeared to provide a viable alternative to explicit grouping as an organizing principle. Users seemed to have a strong, *a priori* sense of how to organize information with pages. They expected math written on one page to be in the same computational scope, and distinct from math written on other pages. They had strong feelings about wanting to grow pages to add related information and to use a new page to enter logically different information. Being able to fold pages to make more space seemed natural and “cool” to most participants; however, many felt that a pen-based technique was needed to precisely define the pinch boundaries while also admitting they might not need folding functionality very often. Alternatively, users were captivated by the panning bar, identifying it as a convenient tableau for collecting informal collections of pages and addressing the desire to spread a working set of pages out beyond the limited dimensions of the display surface. Pages also provided a natural work unit in which users could explore a problem, then discard the page if they were off-track, or push it aside to use a new page when handling an interruption. We expect that pushing the page metaphor more, for example, to flip, curl, staple, hyperlink, or embed pages may reap benefits.

Sandwich Problem. With regard to specific UI choices, we were somewhat surprised to find that users were not inclined to be receptive to bi-manual interaction. We summarize their reticence as the *sandwich problem* in which participants felt that it was unnatural to require bi-manual interaction since their other hand might be doing something else, like holding a sandwich. We interpret this to mean that users are not only concerned with *actually* using their other hand to do something else but they also were concerned that they *might* want to do something with their other hand besides improve a manipulation they could do almost as well with one hand. In essence, if the effort expended on bi-manual interaction appears to greatly exceed any performance benefit gained, then uni-manual interaction may be preferred. It is possible bimanual gestures take “getting used to” and so it may be appropriate to always have unimanual alternatives to ease the learning curve and to address the sandwich problem.

Recognition. Counterbalancing the sandwich principle somewhat, we observed that the most likely gestures to be misrecognized were those that required either multi-touch or pen input, but not both. Hybrid TAP gestures were not accidentally triggered during the evaluation. However, largely due to the poor quality of the pen used, there were occasions when a TAP gesture stem was not recognized, causing ink to be left on the display. Thus we expect that it may be important to increase recognition latitude of the pen part of a TAP gesture, perhaps in response to sensing a hand hovering over the display, and/or to develop efficient recovery techniques when the pen stem is not recognized.

Physical skill. We also found it notable that different users employed different, often inefficient, physical strategies when performing gestures. When shown a more efficient technique, they were almost instantly able to improve their performance, in many cases having an “Aha” moment. For example, switching between finger tip and finger pad touching requires only the bending of the second joint of the index finger; however many users adopted awkward poses such as fully extending the index finger and rotating their arm to be perpendicular to the surface. Similarly, when switching from writing ink with a stylus to dragging terms with their finger, several users tried to find a place to put the stylus on the table instead of tucking it up in their palm. Several users noted that they would like to use under the rock menus with one hand, but did not figure out on their own that this could often be accomplished more easily with the index and forefinger instead of the thumb and index finger. Thus, we expect that pen and multi-touch techniques may require more sophisticated and in-depth disclosure mechanisms than pen only gestures, for instance.

Disclosure and entrenchment. Even though multi-touch input is relatively new, it seemed clear that some techniques have already become entrenched and others were harder to discover and master. For example, several users had trouble considering finger posture as an option for panning a graph because they felt that two-finger dragging was the *de facto* scrolling standard based on their experiences with MacBooks. They also considered their experiences with iPhones and MacBooks where all touches are equal. Thus extending disclosure techniques like GestureBar for surface interaction is worthy area for future research.

Conclusion

We presented a prototype system, Hands-OnMath, which reduces the barriers to accessing computational assistance during math problem solving by unifying CAS functionality with a virtual paper UI. This system contributes novel bi-manual and gestural techniques for managing and writing on virtual note pages in addition to direct manipulation techniques for algebraically transforming mathematical expressions. Pilot studies indicate that, after refinement, a mature version of Hands-On Math would be a desirable tool for scientific and academic note-taking and ideation.