Touch-Display Keyboards: Transforming Keyboards into Interactive Surfaces

Florian Block[1], **Hans Gellersen**[1] **and Nicolas Villar**[2]
[1]Lancaster University, [2]Microsoft Research Cambridge
[1]{block, hwg}@comp.lancs.ac.uk, [2]nvillar@microsoft.com

ABSTRACT
In spite of many advances in GUI workstations, the keyboard has remained limited to text entry and basic command invocation. In this work, we introduce the *Touch-Display Keyboard* (TDK), a novel keyboard that combines the physical-ergonomic qualities of the conventional keyboard with dynamic display and touch-sensing embedded in each key. The TDK effectively transforms the keyboard into an interactive surface that is seamlessly integrated with the interaction space of GUIs, extending graphical output, mouse interaction and three-state input to the keyboard. This gives rise to an entirely new design space of interaction across keyboard, mouse and screen, for which we provide a first systematic analysis in this paper. We illustrate the emerging design opportunities with a host of novel interaction concepts and techniques, and show how these contribute to expressiveness of GUIs, exploration and learning of keyboard interfaces, and interface customization across graphics display and physical keyboard.

Author Keywords: Touch-Display Keyboard, I/O Device, Interactive Surface, Interface Customization, GUI.

ACM Classification Keywords: Input devices and strategies; Graphical user interfaces.

General Terms: Design, Human Factors.

INTRODUCTION
Despite improvements in displays and input devices, GUIs still suffer from a chronic lack of power and expressiveness: simple tasks are often laborious to carry out with a mouse, there is never enough display space available to make commands easy to reach, and customization and tailoring are difficult to do. To address these issues we propose to rethink the role of the keyboard in GUI workstations — arguing that it is currently underutilized and limited to text entry and basic command invocation. We introduce the *Touch-Display Keyboard* (TDK) that seamlessly extends the GUI, provides novel and rich interactions, and facilitates a new quality of interface customization across keyboard and display.

The keyboard is a highly evolved input device. Its physical properties (embossed keys, force feedback, spring threshold) support tactile acquisition and operation of keys, while users can rest their fingers on the keys without triggering, and maintain a focus on the screen [9]. Grounded in these ergonomic qualities, users can touch-type at high speeds for text and data entry and access commands efficiently via hotkeys. The principal idea of the TDK is to combine these qualities with the flexibility of the graphical user interface.

In terms of device hardware, a TDK is a keyboard that retains the physical and mechanical properties of the state of the art keyboard, and that has two extensions: a dynamic display embedded in each key, and a touch-sensor in each key. Related keyboard hardware has been developed in recent products, albeit separately for display [3] and touch [1]. These developments demonstrate the technical and economic feasibility of the proposed device. With this hardware as platform, we conceptualize TDKs to leverage touch and display in four fundamental ways:

- The matrix of key-displays is conceived as a coherent display surface that can extend the primary display in a user interface.
- Graphical elements can be distributed between and moved across keyboard and primary display.
- Mouse interaction is extended across the keyboard display.
- Touch-sensing adds an additional layer and state of input on the keyboard.

A TDK can be viewed as a keyboard that has additional capabilities to improve and expand its core utility as text and command input device. This is the viewpoint from which prior work has investigated keyboard extensions: embedded displays to make key labels dynamic [3]; touch-sensing to add a state for previewing of key actions [17]; and multi-touch gestures to expand the input space of keyboards [1].

However, a TDK can also be viewed as an interactive touch-display. From this perspective, a TDK has very distinctive properties as a result of retaining the physical form of a keyboard. Some of these may appear more as a constraint than as a design opportunity – the uneven surface; the fixed key layout; and the coarse-grained resolution of touch sensing. Other features though expand beyond what an interactive surface commonly supports – tactile feedback; three-state touch input; and the complementary input spaces of mouse and touch (discrete versus continuous). The result is an intriguing design space, for which we provide a first investigation in this paper.

The contributions of this paper are thus threefold. First, we introduce the concept of the TDK together with a proof-of-concept implementation. Secondly, we provide a systematic exploration of the interaction design space of dynamic keyboards. This covers keyboards extended with displays, mouse interaction, touch sensing, and combinations of these, and is of relevance not only for understanding the potential of the TDK, but also for understanding what 'just display' or 'just touch' can add to a keyboard. Thirdly, we demonstrate a range of novel interaction techniques in this design space. These leverage the TDK for innovative command input and control, for interaction with dynamic content on keyboards, and for customization of interfaces across keyboard and primary display.

RELATED WORK

Studies have consistently shown that keyboards with mechanical keys outperform alternative input devices for text entry, including soft-keyboards on touch-displays [13, 19, 20]. In GUIs, they improve command input performance with reduction of required physical movement and with bimanual interaction [7, 14, 15, 16]. The tactile nature of the keys allows users to locate keys by touch, which is known to improve keying performance [9] and enables tactile input modes, such as touch-typing. A significant ergonomic quality and advantage of the physical keyboard is that users can rest their fingers on the keyboard surface without triggering input [22]. These findings are important to our work, as they highlight the value of maintaining the physical characteristics of the conventional keyboard in the design of novel keyboard concepts.

The extension of keyboards with additional input/output capabilities has been demonstrated in a number of recent developments. Art Lebedev have produced a keyboard that has a 48x48 pixel OLED display in each of the keys, to support different input languages and application command sets [3]. United Keys provide a keyboard with a smaller set of display-enabled function keys [2]. Fingerworks produced specialist keyboards with embedded touch-sensing, to support multi-touch input gestures on the keyboard surface [1]. Capacitative touch-sensing in keyboards and keypads has also been demonstrated for finger-sensing and preview of key input [17], for key-touch as additional input mode [11], and for use of the keyboard surface for touchpad-like pointing [4]. Beyond these works, we demonstrate integration of *both* display and touch into mechanical keyboards.

Conversely to adding touch and display to a keyboard, a variety of works have investigated how physical input and tactile feedback can be added to touch-display devices. In the context of mobile interactions, researchers have shown that tactile feedback improves performance with soft-keyboards, while performance with physical keyboards is still significantly better [10]. A different approach, possible on vertical display surfaces, is to use transparent devices for tactile manipulation of underlying dynamic content. Data Tiles demonstrated this with acrylic tiles that afforded certain direct manipulations [18]. SLAP provides transparent widgets made of silicone, including a keyboard to add physicality to interaction with soft-keyboards [23]. Hartmann *et al.* added conventional keyboards and mice on an interactive table, and explored techniques that leverage location and orientation sensing of these devices for co-location of soft inputs and displayed information [8]. A TDK can also be used on interactive tables, but is more generally conceived for use in any setting in which the conventional keyboard is found now.

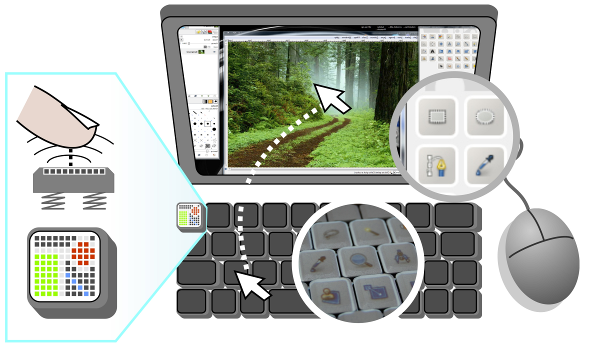


Figure 1. The Touch-Display Keyboard: each key is augmented with a touch-sensor and a graphics display. This enables interaction with graphical elements across keyboard and primary display device.

A TDK is, in effect, an array of physical keys that can be dynamically adapted. Greenberg and Boyle introduced mechanisms for dynamic mapping of graphical controls to *phidgets*, and demonstrated a variety of benefits of end-user customization of interfaces with physical controls [5]. A TDK is more rigid in physical control layout than an interface constructed with a physical UI toolkit, but the keys on a TDK are adaptable in both their function *and* appearance. Moreover, the keys are seamlessly integrated in a mouse and display space that extends over the primary screen and keyboard surface. This enables their customization using powerful direct manipulation techniques that have previously been confined to adaptation of screen-based interface elements [21].

TOUCH-DISPLAY KEYBOARD PRINCIPLES

The TDK is a new type of device for human-computer interaction that is defined by three characteristics:

- A TDK retains the physical form, tactile quality and input functionality of the conventional computer keyboard.
- Each key on a TDK is augmented with a graphics display on its surface.
- Each key is also augmented with a touch sensor that is capable of detecting whether or not a finger touches its surface.

The keyboard extensions of the TDK are illustrated in Figure 1. As indicated, the keyboard is transformed into an interactive surface on which graphical elements can be placed and manipulated. In a conventional interface configuration, a keyboard can only provide input to a graphics display. A TDK, in contrast, enables new forms of interaction and customization between the keyboard and a primary display, as graphical user interfaces can now be mapped across the two devices. In order to facilitate such novel interactions, we conceive the TDK with four properties that extend over the 'raw' addition of display and touch to each key:

- The matrix of individual key-displays is supported as a coherent keyboard display. The keyboard display can be manipulated on the level of key regions (predefined by the keyboard layout) as well as pixel-level. The keyboard display seamlessly extends a primary graphics display, in the same manner as secondary screens do in contemporary interfaces.
- Graphical elements can be distributed between keyboard and primary display. Any graphical element displayed on a screen can also be displayed on the keyboard display. On the keyboard, graphical elements become associated with the key regions they occupy.
- Mouse interaction is seamlessly extended from the primary display across the keyboard. The mouse can be used to select and manipulate graphical elements, key regions and individual pixels on the keyboard, and to drag&drop objects across the display space, including across the boundary of keyboard and primary display areas.
- Touch is supported on the keyboard as a third state for manual key input, to the effect that three-state input is supported consistently across both finger and mouse modalities, and across both keyboard and the primary display.

With these properties, a keyboard is turned into an interactive surface that users can in principle interact with in the same ways as with a GUI on a conventional display device. However, the keyboard surface naturally differs significantly from a conventional GUI surface, as it remains optimized for efficient text and command input. These differences are important for consideration of the novel interactions that a TDK enables in a keyboard-mouse-display configuration:

- Discontinuous surface: The keyboard surface is uneven and has gaps between the keys. This compromises the quality of display across the surface, with discontinuities between the individual keys.
- Fixed layout: in order to make graphical elements accessible via key input, their arrangement must match the physical layout and sizes of the keys.
- Relative spatial arrangement: a TDK is a vertical interactive display whereas a primary display is commonly horizontal. The relative spatial arrangement can impact on ease of interaction across the two surfaces (e.g., switching of visual focus).
- Different input / output resolutions: the resolution of graphical output and mouse input is consistent across TDK and primary display, but touch and key input are of lower granularity (dictated by size and arrangement of the keys). Consequently, graphical elements can only be accessed by touch if they occupy at least one key region fully. Smaller elements can only be accessed using the mouse.
- Discrete versus continuous input: keyboard input is discrete and limited to three-state interaction with graphical elements on the TDK (touch, press, release). Graphical elements and operations with more complex input requirements can only be manipulated with the mouse.

PROOF OF CONCEPT IMPLEMENTATION

For proof-of-concept we have built a fully functional prototype of a TDK (Fig. 2). The prototype is based on an off-the-shelf keyboard (ADVENT KBW100). A dynamic display was added by means of overhead projection, for which the surface of the keys was spray painted in matte white. An XVGA projector (1024 x 768) was mounted top down on a cross bar about 1 meter above the keyboard. In order to maintain a consistent projection, the keyboard was fixed to the table at a suitable position. A mask was used to limit the projection to the area covered by keys. The effective display area per key was approximately 48x48 pixels.

We integrated 10 Quantum E1103 touch-sensors (with 10 channels each) on custom-built boards into the base of the keyboard. From there we routed copper wires through the keyboard to each key, connecting to capacitative tape under each key cap. The tape was carefully sized and placed to only generate sufficient capacitance when the finger touches the center of the key (to avoid unwanted trigger of adjacent keys). For aggregation of the touch-sensor signals we used a small external board connected to the computer via USB.

On top of the hardware layer, we developed a TDK framework on .NET and WPF, as software platform for exploration of the interaction design space of dynamic keyboards. Because we used overhead projection as secondary display connected to the computer, graphical output and mouse input were implicitly extended to the keyboard surface. The key regions in the display were managed with a mapping table associating a key code with a rectangle in the display coordinate space. On the input side, we detached the keyboard signals from the operating system via a global input hook, to redirect key events via a custom input layer that integrated touch-sensing. The key and touch events (pressed, touched, released) are handled in the same way as mouse events (click, hover, release) for three-state interaction (but key and mouse input are maintained separate, i.e. key touch does not move the mouse pointer).

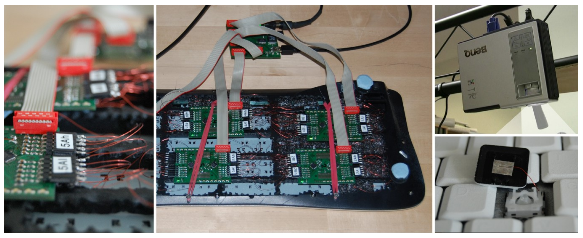


Figure 2. Proof-of-concept: prototype implementation of a TDK using capacitive touch-sensing and overhead projection.

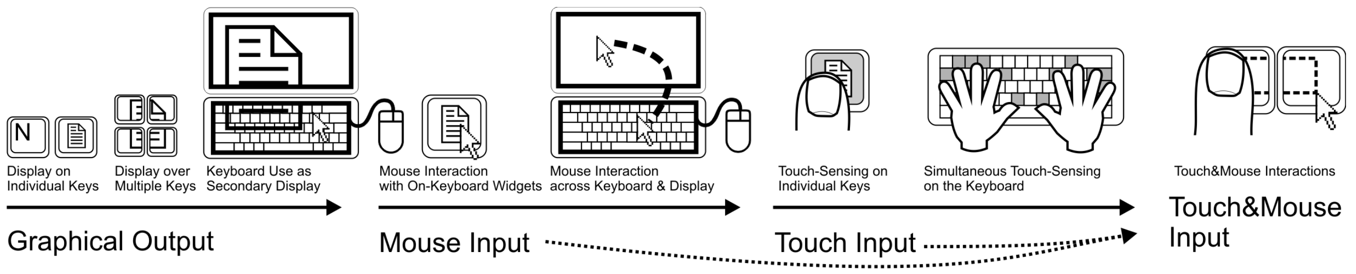

N
Display on Individual Keys
Display over Multiple Keys
Keyboard Use as Secondary Display
Mouse Interaction with On-Keyboard Widgets
Mouse Interaction across Keyboard & Display
Touch-Sensing on Individual Keys
Simultaneous Touch-Sensing on the Keyboard
Touch&Mouse Interactions
Graphical Output
Mouse Input
Touch Input
Touch&Mouse Input

Figure 3. The design space of TDKs is defined by three dimensions of graphical output, mouse input and touch input in addition to conventional key input. New design opportunities emerge along each dimension and in addition from the novel combination of mouse and touch input on the keyboard.

Handling keyboard events like mouse input has a number of advantages. First of all, we can define generic *hover* and *invoke* behaviors, which abstract from finger versus mouse input. This allows us to design graphical controls independently of whether they will be placed on a conventional display or on the keyboard, and also in abstraction from the modality the user chooses for interaction. Secondly, the keyboard surface can be designed in essentially the same way as for conventional GUIs. For instance, several TDK interfaces can stack, while only the top-most interface receives the events, just like with stacked windows. A TDK interface can thus simply be activated by bringing it to the front. In the same way, parts of a TDK interface can be dynamically overloaded, for example for context-dependent mapping and re-labeling of a group of keys. A general advantage is that existing interface programming tools, such as GUI-builders, can be readily applied to TDK interface development.

Our implementation facilitates integration of the TDK with many existing applications using Microsoft Interface Automation. This enables us to hook into existing graphical user interfaces, to query detailed information (e.g., screen bounds and description of graphical elements) and to inject events (e.g., invoking graphical controls). We can also copy the visual appearance of controls by using the Win32 API. Our implementation further supports OLE drag&drop, allowing us to associate a variety of data, such as images, files or audio clips with keys on the TDK. All examples of TDK interaction techniques that we provide in this paper are implemented using on this framework, and integrated with real applications such as Microsoft Office and Firefox.

THE DESIGN SPACE OF TOUCH-DISPLAY KEYBOARDS

The design space of TDKs is defined by the three dimensions of graphical output, mouse input, and touch input in addition to the conventional key input. As shown in Figure 3, graphical output as such adds new possibilities in terms of display on individual keys, over multiple keys, and over the entire keyboard as secondary display complementary to the primary screen. Mouse input added to graphical output facilitates new forms of interaction with widgets on the keyboard, and interaction across the keyboard and primary display device. Touch input adds a further dimension, providing an additional input state for each individual key, and enabling multi-touch input across the keyboard. New design opportunities arise also from the novel combination of mouse and touch input on the keyboard.

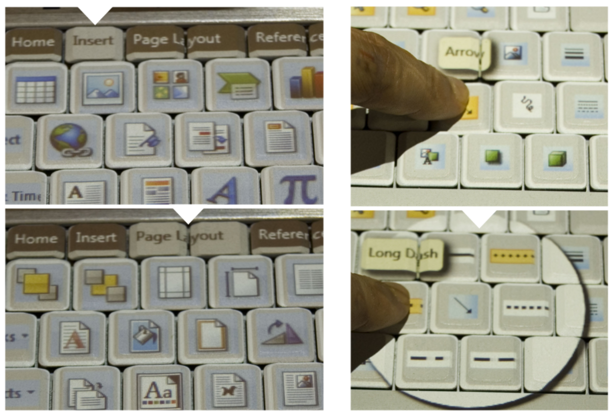
In Figure 3, the new design dimensions are shown in sequence although they do not necessarily built on each other. Mouse techniques by necessity built on graphical output on the keyboard, but touch input can extend the expressiveness of the keyboard as such, and provide new opportunities both with and without dynamic display on the keyboard.

Subsequently, we provide a systematic analysis of the design space combined with introduction of novel interaction techniques. Figure 3 provides a roadmap for our purpose, which we will traverse from the left (analysis of what display on keys facilitates) to the right (exploration of touch&mouse interaction on the keyboard).

Display on Individual Keys

Display on individual keys enables dynamic mapping and relabeling of keys. This can be utilized to improve keyboard interaction in a number of ways:

- *Visualization of hidden functionality*. Keys are routinely overloaded with hotkeys for fast access to commands. However, in contrast to GUI controls (which can be visually explored), hotkeys are hidden and difficult to acquire and memorize by the user [7, 24]. Some techniques that address this problem are already available in Art Lebedev's display keyboard. This includes display of control icons when a modifier key is pressed (e.g., displaying the 'cut' icon on the 'X' key when CTRL is held down), and dynamic adoption of hotkeys according to the active applications (e.g., different icons for Photoshop and Word) [3].
- *Spatial mapping of keys*. On conventional keyboards, mappings are commonly mnemonic (e.g., 'B' for Brush in Photoshop) to help users memorize hotkeys. However, when visualized on the keyboard, such mappings can appear random, since the association with the original key is no longer obvious. Instead, spatial mapping strategies can be implemented, for example arranging related hotkeys in rows, similarly to toolbars on the screen. Previous work has shown that such spatial mapping strategies can improve user performance [15, 16].
- *Multiple key maps*. A dynamic keyboard can support alternative input languages, as demonstrated in Art Lebedev's product (e.g., enabling users to switch latin and cyrillic character sets [3]). More generally, dynamic labels enable support of multiple key maps between which the user can switch. This can provide access to alternative symbol



(a) Each function key acts as graphical tabulators, activating a different keyboard interface (example from Microsoft Word).

(b) Pie menus allow the temporary overloading of adjacent keys, eg. to provide refinement of a command.

Figure 4. TDKs enable dynamic mapping strategies for keyboard interfaces.

sets for text input, and to alternative hotkey allocations for command input. As a novel technique to support this, we have developed *Keyboard Tabs* (see below).

- *Context-dependent mapping*. Dynamic mapping can be used to overload keys in a contextualized manner. For example, if a command is triggered that requires parameter input, then adjacent keys can be temporarily overloaded to display the available parameters (this is exemplified in *Hotkey Pie Menus*, a technique we introduce below).

Keyboard Tabs
Keyboards Tabs are designed to provide fast access to different keyboard maps. A row of keys at the top of the keyboard, in our prototype the row of function keys, is treated as tabulators. Each of the tab-keys provides access to an associated key interface. The technique is illustrated in Figure 4(a), showing its use in the context of a word processor. When a tab-key is pressed, the keys are overloaded with an associated panel of hotkeys. Pressing a tab again hides the interface and brings back the text interface. This concept is similar to graphical menus in conventional GUIs, such as tabulator panels (each tab showing a different set of controls), or 'ribbons' (switchable interface panels). In our prototype up to 12 different hotkey panels can be supported (one for each function key), facilitating fast and uniform access to over a thousand distinct commands. This has the potential to improve command input in feature-rich software, overcoming the problem that keyboard shortcuts are only provided for a subset of commands (which has been found to be problematic as it requires users to decide whether a shortcut is available [12]).

Display over Multiple Keys
A group of keys can be used as a coherent area for interaction with a graphical element. This can be used to provide multi-key input devices:

- *Multi-Key Buttons*. A graphical element can be arranged over multiple keys that together function as a larger button, such that pressing of any of the keys triggers the default action associated with the element. This is useful when the element itself cannot easily be represented on a single key (e.g., a URL), and it can also be used to make a control easier to acquire by increasing its size.

- *Keyboard Scalar Controls*. Scalar controls can be supported on the keyboard by grouping adjacent keys that represent steps on a scale. A keyboard provides only a low resolution for interaction with scalar controls. However, this type of control can be useful in cases that do not require high precision (e.g., zoom control) or where discrete steps are desirable (e.g., a control for line thickness). Alternatively, finer-grained control can be added with mouse integration on the keyboard (see Fig. 6).

- *Keyboard Menus*. A group of keys can implement a menu, with each key representing an item. As a concrete example, we introduce Hotkey Pie Menus, which also demonstrate context-dependent overloading of keys.

Hotkey Pie Menus
Pie menus are well known from tablet computing. When a pen is put down on an element, a pie menu appears around the point of contact, and provides a set of menu items radially arranged around the pen, for selection with minimized movement. Figure 4(b) shows how we transferred the concept to keyboard interfaces: when a key is pressed, a pie menu temporarily overloads the functionality of the surrounding keys. In the shown example, the user has selected a 'draw arrow' tool, and can then refine the command by selection of line style options. Like the original pie menus, Hotkey Pie Menus provide a gestural interface. The adjacent arrangement of command and options facilitates muscle memory with frequent use. As the acquisition of adjacent keys is also guided by tactile feedback, experienced users could learn to use the technique while maintaining their visual focus on the primary screen.

Keyboard use as Secondary Display
The keyboard can more generally be considered as a secondary display. The physical properties of a TDK limit its utility for general-purpose display, but it provides display space that is 'ready to hand' when the keyboard is not used for input. From this perspective, it can complement the primary display with additional, albeit lower-quality, display real estate. The following are concrete examples for complementary use as secondary display:

Keyboard context display
Keyboard and primary display can be combined as a focus&context display, to provide simultaneous access to information detail and surrounding context. For example, map data can be shown at fine-grained resolution for a small geographical region area on the primary display, and at lower resolution for a larger area on the keyboard display. The keyboard facilitates fast focus switch by key input.

Keyboard palette

Applications such as Photoshop make extensive use of tool palettes for access to editing tools, but the palettes obstruct the actual work surface. The keyboard surface can be used to offload the palettes from the primary work surface. In related work, external handhelds have been used to offload palettes [6] but that requires inter-device communication, whereas a TDK seamlessly extends the primary display space.

Keyboard clipboard

The keyboard can be used as display space for the clipboard. Clips can be represented by thumbnails, and directly accessed by key input, for pasting into applications on the primary display.

Mouse Interaction with On-Keyboard Widgets

Support of the mouse on the keyboard is an unusual concept as both are primarily associated with input. However on a display-augmented keyboard, mouse interaction can facilitate interface behaviors that are standard in GUIs, in order to adopt their usability benefits for the keyboard context.

- *Interface Exploration.* An important function of the mouse in a GUI is interface exploration. A common technique are *tooltips* that are triggered when the mouse is moved over a graphical control. They allow users to inspect what command or tool an icon represents, and are an aid for novices to discover and learn interface functionality. Also common are are context menus for inspection of interface properties (see below). Extension of mouse support to the keyboard enables use of these techniques to improve keyboard usability (e.g., exploration of available shortcuts).
- *Meta-interaction on widgets.* In a GUI, the mouse is also used for meta-interaction with interface elements. For example, 'right click' on widgets is a standard technique for opening a context menu or dialog, through which properties of the widget can be customized. Extension of this behavior to the keyboard can specifically facilitate customization of key mappings and shortcuts, and support other techniques that we have introduced (e.g., customization of the menu items in a keyboard pie menu). A concrete example for a context menu is introduced below.
- *Continuous Input on Scalar Control.* Above, we have proposed mapping of scalar controls onto arrays of keys. The mouse can complement keyboard input on such controls for more fine-grained input. A concrete implementation of such a multimodal control is introduced below.

Context menu for keyboard widgets

We have implemented context menus that can be accessed with the mouse on keyboard widgets (see Fig. 5). As shown, context menus can be employed to provide generic access to the properties of a widget. However context menus can also be provided for specific keyboard tasks. As a concrete technique, we have implemented a context menu for text entry that can be invoked on a key to access variants of the input character (e.g., versions of the character with different accent marks). Such techniques have previously been limited to soft keyboards (e.g., iPhone).

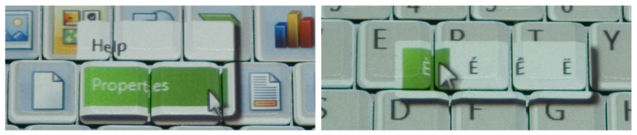


Figure 5. Context menus allow the inspection of on-keyboard widgets.

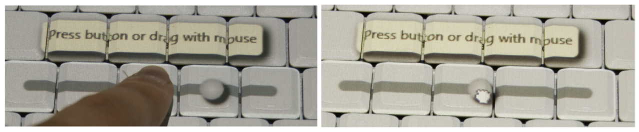

Press button or drag with mouse
Press button or drag with mouse

Figure 6. Continuous controls (such as sliders) can be operated in discrete steps using the keys, or at finer resolution using the mouse.

Multimodal slider control

Figure 6 shows a multimodal slider we have implemented as widget for use on keyboards. The slider can be controlled by key input, for adjustment in discrete steps, and by mouse input for fine-grained adjustment. Both input modes can be used interchangeably.

Mouse Interaction across Keyboard & Display

The mouse has a distinct role in the TDK concept, as it supports continuous input across the combined interaction space of keyboard and primary display. The seamlessness of mouse interaction across the two surfaces is significant in two ways:

- *Interface customization across keyboard and display.* The mouse can be used to customize the extended interface using direct manipulation techniques. Prior work has demonstrated advanced customization mechanisms for GUIs that can in principle be adopted. As a concrete technique, we have implemented drag&drop to facilitate rearrangement of TDK interfaces (see below). This enables, for example, dragging of graphical control from the primary display onto the keyboard to create a hotkey.
- *Uniform interaction with graphical elements.* The mouse supports uniform interaction with graphical elements, irrespective of whether they are located on a screen, or on

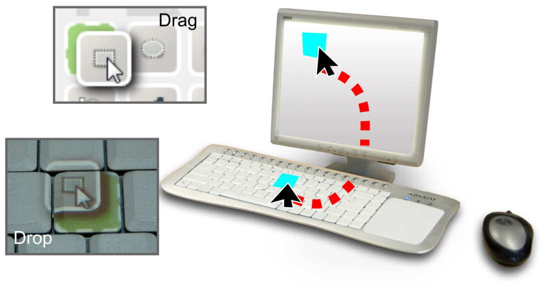


Figure 7. Drag&drop of graphical items between primary display and keyboard. This technique facilitates interface customization by drag&drop of items such as widgets, browser tabs, and image clips onto the keyboard to make them physically accessible.

the keyboard. When a user moves a graphical control from the display onto the keyboard, they can still interact with it using their accustomed mouse techniques (e.g., for exploration and editing of properties, as discussed above).

Drag&drop interface customization

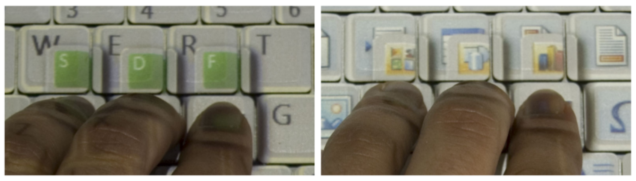
In our prototype, we have explored new ways of customizing graphical interfaces across keyboard and display. Figure 7 shows, how mouse interaction can be used to move graphical elements between display and keyboard. A graphical element can be picked up by holding down the right mouse button and moving the mouse, causing the control to be detached from its current location on the screen. The control can then be dropped onto a key, while size and appearance are adapted to the new interface slot. The technique can be applied not only to controls but to any type of graphical element. Our implementation supports the dragging of different types of control (e.g., interface containers such as tabs in Firefox), hyper-links, items from the task bar, and clips (e.g. images). This enables very flexible use of a TDK beyond text and command input, for instance for task switching, access to frequently used web sites, and pasting of clips.

Touch-Sensing on Individual Keys

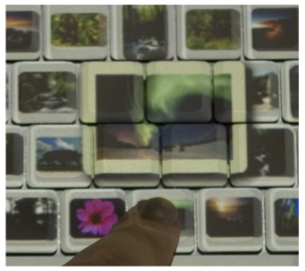
In the context of individual keys, the significance of touch-sensing is that it adds a third input state. Recent work has demonstrated the utility of three-state input on keyboards and keypads [17], however a TDK provides a different context as it also supports dynamic display on keys, affording the following interactions:

- *Proactive preview*. The idea of proactive preview is to inspect an input before it is triggered. This is useful in particular for novice users, and is routinely supported with tooltips in GUIs. Rekimoto *et al.* have demonstrated proactive preview for keyboard input, providing tooltips on a separate display [17]. A TDK, in contrast, enables tooltip display directly on the keyboard. Touch and mouse-over can be used interchangeably to invoke the preview. Instead of a tooltip, it also possible to invoke a *live preview* by simulating the effect of the input (e.g. touch of 'CTRL+B' would show selected text in bold, but if the keys are not pressed, the formatting is not actually applied). In addition to these existing forms of preview, we introduce two new techniques that are specifically motivated by display on keys: *Preview of content* and *Finger occlusion preview*.

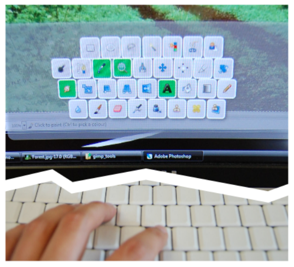
- *Touch-based actions*. Touch can also be used to directly trigger input [11]. Such a strategy conflicts conceptually with proactive preview, but there are scenarios for which touch-based input may be appropriate. In particular, for keys with fixed and familiar functionality (where preview may not be necessary) touch can be mapped to actions. An example might be the arrow keys for scrolling in text document: 'touched' could be mapped to slow scrolling, and 'pressed' to fast scrolling. However, such overloading has to be designed with care. Keys that are prone to accidental touch, as well as keys that may be touched in rest position during typing are not suitable for such a mapping.



(a) Finger-occlusion preview: graphical controls occluded by fingers are shown above the actual key.



(b) Content-preview of clips.



(c) Preview of finger positions on a virtual on-screen keyboard

Figure 8. Proactive preview techniques on the TDK.

- *Touch-based context menus*. Generally, it is desirable to facilitate manual access as alternative to mouse access for interaction with elements on the keyboard. For example, context menus that support text entry with access to special characters (as introduced above) are more efficient if they can be opened manually rather than by mouse-over. A possible way of implementing this is prolonged touch to open the menu, and layout of the menu items over adjacent keys.

Preview of content

TDKs can act as output device for presentation of content, but content elements become occluded when users interact with them via key input. Figure 8(b) shows a content preview technique we have implemented to address this problem. When the users touches a key that displays content (in this case an image clip), then the content is displayed in enlarged form above the key (or next to the key, if it is in the top row).

Finger occlusion preview

A beneficial ergonomic property of keyboards is that users can rest their fingers on the surface without triggering functionality (e.g., in a 'home position'). When a user rests their fingers in this way, the display of the underlying keys is occluded. To provide visual guidance in such a context, we have implemented finger occlusion preview, as shown in Figure 8(a). This is similar to key preview on the iPhone.

Simultaneous Touch-Sensing on the Keyboard

The keyboard as a whole supports simultaneous touch-sensing on the granularity of keys. This enables interactions that exploit sensing of finger positions and multiple touch points:

- *Finger sensing as context*. Presence/absence of fingers can be detected and used as context, for example to make

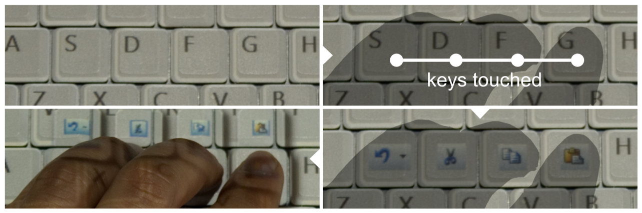


Figure 9. A group of four commands (e.g., undo, cut, copy and paste) can be assigned to the user's finger tips. Whenever the fingers touch a row of adjacent keys, the underlying keys are overloaded accordingly.

the keyboard available for output, when it is not used for input. Sensing of finger positions can also be used to infer user activity, for instance whether the user is primarily typing or formatting (which can be used to adapt the interface, e.g. context menus). Finger positions can also be displayed as context to the user. This has been demonstrated in related work by Rekimoto *et al.*, and is useful when keys and visual focus are separated [17]. We have also implemented this for the TDK, with a soft keyboard on the primary display for feedback on which keyboard elements are touched (cf. Figure 8(c)).

- *Implicit gestures.* Closely related is sensing of gestures that are implicit or natural with particular uses of the keyboard. For instance, placement of eight fingers on the home row can be used as an implicit gesture to switch the keyboard into text-entry mode.

- *Motion gestures.* The resolution of the touch input space is coarse-grained but sufficient to detect finger and hand motions across keys. This can be used to support distinct motion gestures as input on the keyboard (see below for examples we prototyped).

- *Multipoint input.* Multipoint input gestures that have been developed for multi-touch surfaces can be adopted on the keyboard, for example for two-handed manipulation of objects displayed on the primary screen. The uneven surface and segmentation of the keyboard surface can limit smooth execution of multi-touch gestures but on the other hand provides opportunities for multi-touch input with tactile feedback. An example is introduced below.

- *Finger position gestures.* The relative positions of fingers can be interpreted as input. For example, touch with of two, three or four fingers on adjacent keys can be treated as different gestures. Below we describe a technique that exploits this to provide *fingertip commands*.

Motion gestures for keyboard adaptation
A *finger flick* gesture across the keyboard has been implemented for 'flicking' through different keyboard mappings (e.g. different input languages in text mode, or different tool palettes for command input). Complementary, a *push gesture* is provided as hand motion toward the primary display, to make the keyboard interface visible within the primary visual focus (a push in the opposite direction moves the keyboard visualization off the screen).

Scaling with tactile feedback
This technique supports fine-grained scaling of objects on the screen with relative input from two fingers on a keyboard row. When the fingers are moved apart by one key, the object is scaled up by one pixel, and conversely the object is scaled down when the fingers are moved together.

Fingertip commands
This technique allows binding of commands to the four fingertips of a hand. For example, undo, cut, copy and paste might be mapped in this way. When the user touches the keyboard with the four fingers on adjacent keys, this is treated as a gesture that temporarily overlays the keys with fingertip commands (see Fig. 9). This means that the commands move with the fingers and can be triggered on any row of four keys on the keyboard, which makes them particularly easy to acquire.

Touch&Mouse Interactions
Finger and mouse input can be combined on a TDK:

- *Multimodal input.* The two modalities can complement each other for coarse-grained versus fine-grained input (as demonstrated above, with a *Multimodal Slider*). The modalities are used on the same object but separated in time.

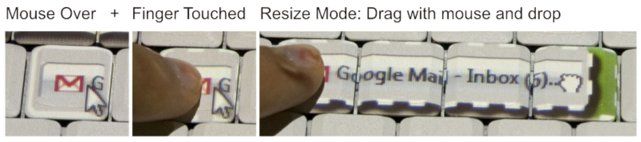
- *Hold and move input.* Touch naturally affords 'select and hold' due to segmentation of the keyboard, whereas the mouse can be used more easily for 'select and move' interactions. This can be combined for resizing and cloning of objects on the keyboard, see below.

- *Combined input.* The two modalities can be used simultaneously on the same key. This enables four additional input states (key touch versus press, combined with mouse-over versus mouse-down). Combined input is used in the two techniques below.

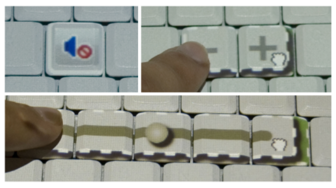
Resizing of keyboard elements
This technique complement drag&drop customization as introduced above. A 'drop' maps an element to a single key. If the element requires more space, then it needs to be resized. This can be fluently supported by combining touch and mouse, as shown in Figure 10(a): the dropped element is touched to hold it on the key, while the mouse is dragged across adjacent keys to 'stretch' the control. Elements can be resized also in two dimensions (cf. Figure 10(c)).

Adaptive resizing of scalar control
This technique supports adaptive resizing and mapping of scalar controls. When such a control is dropped onto a key, it is mapped to on and off states that can be toggled. When the control is resized across two keys, it is mapped to '+' and '–' buttons. When it is further enlarged, it becomes mapped to a scalar control (cf. Figure 10(b)).

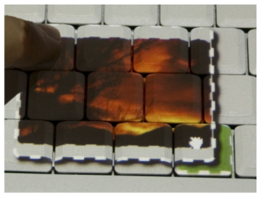
Cloning of graphical keyboard controls
It can also be desirable to replicate graphical objects on the keyboard. In a GUI this is typically supported by holding



(a) Controls enter a resizing state when both mouse and finger are over the associated key, and the mouse can be dragged for resizing.



(b) Adaptive mapping and resizing of a continuous control.



(c) Elements can be resized across keyboard rows.

Figure 10. Touch&Mouse interaction to customize keyboard widgets.

down a modifier key (e.g., CTRL) while the object is selected with the mouse. The mouse can then drag a copy of the object to a target location. On a keyboard, the same mouse operation can be used, but in conjunction with literally holding the original object in place (by pressing the corresponding key).

DISCUSSION

The analysis of the design space shows that the TDK concept gives rise to a wide spectrum of new and compelling uses for the keyboard in a GUI workstation. The contributions of the TDK include rich and expressive interaction and meta-interaction techniques, novel support for interface exploration and learning, and a new quality of interface customizability.

Expressiveness. In conventional GUIs, many tasks have to be carried out using the mouse, which is often laborious, reduces parallelism and limits the user's general expressiveness. We have shown that TDKs address this issue by providing access to a variety of tasks and allowing the user to off-load many activities to the keyboard's surface. We have demonstrated a series of techniques that go beyond traditional hotkeys, and explored ways of enabling feature-rich and expressive keyboard interfaces that combine the flexibility of graphical elements with the known benefits of physical and tactile controls.

Exploration and Learning. Conventional keyboard interfaces are chronically difficult to learn, and many users fail to adopt keyboard-based input strategies in practice. As demonstrated, TDKs not only make keyboard interfaces visual but also allow the direct application of common tools for exploring GUIs to keyboard interfaces – conceptually unifying how users can explore interfaces across keyboard and display. Additionally, we have presented techniques that can help the user make the transition from effective to more advanced techniques. For instance, finger-preview is used to assist the tactile acquisition of keys while providing visual guidance on the primary screen, close to the current task. Using these techniques, dynamic keyboards can also assist the learning of touch-typing. For novice typists, training applications could provide visual guidance directly on the keyboard (e.g., arrows pointing from current finger-position to the target key). Finger-preview can then be utilized to move the user's primary focus away from the keyboard.

Interface Customization. TDKs facilitate a new quality of interface customization across keyboard and display. The keyboard is enabled as additional space for customization, and supports customization of graphical elements with physical features. In principle, any type of graphical element can be mapped to a physical key using direct manipulation. This includes intuitive mapping of controls to hotkeys but also dynamic mapping of elements that are typically not keybound, including short-lived elements such as clips or task items. The keyboard thus becomes a reconfigurable palette on which users can arrange graphical elements for many purposes beyond the functionality of conventional keyboards, including as clipboard, taskbar, or drop area for hyper-links. Elements customized in this way become physically accessible, and the TDK provides distinctive ways of interacting with graphical elements that are not available on standard display devices, for example tactile navigation of controls.

From a practical stand-point, we have successfully integrated the design space in a single working prototype and tested it with existing applications. Some of the techniques – such as multi-touch interaction or text input – have to be enabled or disabled depending on the application requirements, since they require exclusive access to the keyboard. However, we have experienced that switching in most cases is very seamless, for instance based on the active application, the user's activity (using keyboard touch as context), or explicit keyboard switching by the user (e.g., via keyboard tabs). The majority of the techniques has proven to co-exist without conflict and to provide many synergies. For instance, we found that finger preview works well to assist the blind operation of keyboard pie menus. In fact, keyboard pie menus are easy to use in this mode, since acquiring adjacent keys based on direction is supported by the keyboard's physical features. Overall, we conclude from our practical experience that TDKs can not only support the diverse techniques demonstrated in the design space, but integrate these synergistically. Moreover, we found that our implementation of the TDK worked transparently with existing applications in a standard operating system environment.

The demonstrated TDK concepts and techniques were implemented using overhead projection for display on the keyboard. For wider deployment, the display needs to be more tightly integrated with the keyboard. The feasibility of a fully embedded implementation has already been demonstrated in an existing display keyboard product [3]. An alternative could involve pico-projectors, that might for instance be integrated into the lid of a notebook and automatically adjusted to point at the keyboard.

CONCLUSION

We have introduced the TDK as a new concept that transforms the keyboard into an interactive surface, and facilitates its tight integration with GUIs. The TDK retains the physical characteristics of a conventional computer keyboard with its proven advantages for efficient input and combines these very effectively with the flexibility of graphical direct manipulation. As shown in this paper, this gives rise to compelling new uses for the keyboard, and to innovative techniques for interaction and meta-interaction across keyboard, mouse and primary screen. As demonstrated, these techniques contribute very significantly to the expressiveness, accessibility and customizability of GUI workstations.

In this work, we have focused our analysis on how TDKs transform the conventional use of keyboards in a typical computer work environment with keyboard, mouse and screen. However, keyboards are also used as input device in many other contexts, including on mobile devices, embedded with appliances, and in conjunction with interactive tables. A challenge for future work is thus to understand how touch- and display-augmentation can impact in those contexts, for instance to provide access to larger command sets on smaller keyboards and keypads, or to support multi-user interactions via multiple keyboards on a shared interactive surface.

REFERENCES

1. Fingerworks. http://www.fingerworks.com/.
2. OLED Display Keyboard, United Keys Inc. http://www.unitedkeys.com/.
3. Optimus Maximus Keyboard, ArtLebedev Studios. http://www.artlebedev.com/everything/optimus/.
4. W. Fallot-Burghardt, M. Fjeld, C. Speirs, S. Ziegenspeck, H. Krueger, and T. Läubli. Touch&type: a novel pointing device for notebook computers. In *Proc. NordiCHI '06*, pages 465–468, 2006.
5. S. Greenberg and M. Boyle. Customizable physical interfaces for interacting with conventional applications. In *Proc. UIST '02*, pages 31–40, 2002.
6. D. Grolaux, J. Vanderdonckt, and P. V. Roy. Attach me, detach me, assemble me like you work. In *Proc. INTERACT'05*, pages 198–212.
7. T. Grossman, P. Dragicevic, and R. Balakrishnan. Strategies for accelerating on-line learning of hotkeys. In *Proc. CHI 2007*, pages 1591–1600, 2007.
8. B. Hartmann, M. Ringel Morris, H. Benko, and A. D. Wilson. Augmenting interactive tables with mice & keyboards. In *Proc. UIST '09*, 2009.
9. M. Helander. *Handbook of Human Computer Interaction*. North-Holland, 1988.
10. E. Hoggan, S. A. Brewster, and J. Johnston. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *Proc. CHI '08*, pages 1573–1582, 2008.
11. P. Holleis, J. HŁkkilŁ, and J. Huhtala. Studying applications for touch-enabled mobile phone keypads. In *Proc. TEI'08*, pages 41–44, 2008.
12. A. Howes, S. J. Payne, and A. Woodward. The trouble with shortcuts. In *CHI '00 Extended Abstracts*, pages 267–268, 2000.
13. P. Isokoski. A minimal device-independent text input method. Technical Report A-1999-14, Department of Computer Science, University of Tampere, 1999.
14. D. M. Lane, H. A. Napier, S. C. Peres, and A. Sandor. Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts. In *IJHCI, 18(2)*, pages 133–144, 2005.
15. H. McLoone, K. Hinckley, and E. Cutrell. Binamual interaction on the microsoft office keyboard. In *Proc. Interact '03*, pages 49–56, 2003.
16. D. L. Odell, R. C. Davis, A. Smith, and P. K. Wright. Toolglasses, marking menus, and hotkeys: A comparison of one and two-handed command selection techniques. In *Proc. Graphics Interface 2004*, pages 17–24, 2004.
17. J. Rekimoto, T. Ishizawa, C. Schwesig, and H. Oba. Presense: interaction techniques for finger sensing input devices. In *Proc. UIST '03*, pages 203–212, New York, NY, USA, 2003. ACM.
18. J. Rekimoto, B. Ullmer, and H. Oba. Datatiles: a modular platform for mixed physical and graphical interactions. In *Proc. CHI '01*, pages 269–276, 2001.
19. H. Roeber, J. Bacus, and C. Tomasi. Typing in thin air: the canesta projection keyboard - a new method of interaction with electronic devices. In *CHI Extended Abstracts 2003*, pages 712–713, 2003.
20. A. Sears. Improving touchscreen keyboards: Design issues and a comparison with other devices. In *Interacting with Computers, 3(3)*, pages 253–269, 1991.
21. W. Stuerzlinger, O. Chapuis, D. Phillips, and N. Roussel. User interface façades: towards fully adaptable user interfaces. In *Proc. UIST '06*, pages 309–318, 2006.
22. C. Tomasi, A. Rafii, and I. Torunoglu. Full-size projection keyboard for handheld devices. In *Com. of the ACM, 46(7)*, pages 70–75, 2003.
23. M. Weiss, J. Wagner, Y. Jansen, R. Jennings, R. Khoshabeh, J. D. Hollan, and J. Borchers. Slap widgets: Bridging the gap between virtual and physical controls on tabletops. In *Proc. CHI '09*, pages 481–490, 2009.
24. J. Zhang and D. A. Norman. Representations in distributed cognitive tasks. In *Cognitive Science, 18*, pages 87–122, 1994.