Command Strokes with and without Preview: Using Pen Gestures on Keyboard for Command Selection

Per Ola Kristensson
Dept. Computer and Information Science
Linköpings universitet, Linköping, Sweden
perkr@ida.liu.se

Shumin Zhai
IBM Almaden Research Center
650 Harry Road, San Jose, California, USA
zhai@almaden.ibm.com

ABSTRACT
This paper presents a new command selection method that provides an alternative to pull-down menus in pen-based mobile interfaces. Its primary advantage is the ability for users to directly select commands from a very large set without the need to traverse menu hierarchies. The proposed method maps the character strings representing the commands onto continuous pen-traces on a stylus keyboard. The user enters a command by stroking part of its character string. We call this method "command strokes." We present the results of three experiments assessing the usefulness of the technique. The first experiment shows that command strokes are 1.6 times faster than the de-facto standard pull-down menus and that users find command strokes more fun to use. The second and third experiments investigate the effect of displaying a visual preview of the currently recognized command while the user is still articulating the command stroke. These experiments show that visual preview does not slow users down and leads to significantly lower error rates and shorter gestures when users enter new unpracticed commands.

Author Keywords
commands, pen gesture, shorthand, keyboard shortcuts

ACM Classification Keywords
H5.2. Information interfaces and presentation: User Interfaces – *input devices and strategies, interaction styles, theory and methods*; I5.1. Pattern recognition: Implementation – *interactive systems*.

INTRODUCTION
Advances in mobile computing hardware are rapidly increasing the processing power, wireless bandwidth, and storage capacity in handsets, electronic tablets, and other ultra-mobile computers. Palmtop computers with a 1GHz processor and 30GB of storage are already available on the market today. The increasingly limiting bottleneck to information flow in ultra-mobile computing, however, is the lack of efficient user interfaces on these devices. In comparison to desktop or laptop computers, today's handsets and tablet PCs are far more cumbersome to use.

When a desktop keyboard and a mouse are not an option, an electronic pen provides a mobile and flexible alternative means of input. The current work focuses on pen-based command selection. The basic and de-facto standard method of issuing commands on a pen-based computer is the same as on a desktop PC: hierarchical pull-down menus. The limitations of pull-down menus on desktop computers have long been recognized [9]. Pull-down menus are much more problematic on a pen-based touch-screen computer for a number of reasons. First, the pen (stylus) and its holding hand often obscure the very items on the pull-down menu the user needs to find and select. Second, pen motion on a screen has to be one-to-one in scale. This is in contrast to other pointing devices—such as the mouse—whose control-to-display gain is rate-accelerated, so that one does not have to move over a large distance to reach a far-away menu on a large display. Pull-down menus on mobile devices also pose a space management problem, forcing application developers to either remove functionality or introduce an even more complex pull-down menu navigation system than on their desktop counterparts.

What comes to pull-down menus' rescue on desktop and laptop computers are keyboard shortcuts for frequent commands such as *Copy* and *Paste*. Without a keyboard, these shortcuts are often what the user misses the most on a pen-based device. Despite the flexibility of the pen, command selection on mobile devices remains inefficient and unsatisfactory.

We present a new and practical command entry technique for pen computers called *command stroke* (CS). CSs are pen-gesture traces defined on a graphical keyboard according to the letters in the commands such as *c-o-p-y* and *p-a-s-t-e*. CSs offer users a complementing method of directly selecting any command without needing to browse a menu hierarchy.

The development of CSs followed an iterative process. An early incarnation of the concept was compared with

CHI 2007, April 28–May 3, 2007, San Jose, California, USA.

traditional pull-down menus. Encouraged by the positive results and informed by the feedback gained from the experiment we developed a second iteration of the technique we call *command strokes with preview* (CSP). The structure of this paper reflects the iterative development process: We first present the initial incarnation of command strokes and its evaluation. Thereafter we present the second iteration of the method and two experiments that investigate how the visual preview functionality impacts end-users. Last, we compare our work to previous research and point to future work.

COMMAND STROKES

The concept of CS grew out of SHARK shorthand [5, 14], a word recognition system also known as ShapeWriter. A user-drawn CS on a graphical keyboard is recognized by a pattern recognizer that compares its geometrical shape, not the keys crossed on the keyboard, with all CS templates to return the user intended command. Since this pattern recognition approach takes advantage of the constrained number of commands (as opposed to the infinite number of possible shapes or geometric patterns on a keyboard), it has an inherent error tolerance. We use the same recognition method that we previously reported for word recognition in [5], which gives an in-depth explanation of the recognition process.

Short Command Strokes

In the first design iteration CSs were divided into two classes: *short command strokes* and *long command strokes*. A short CS is a pen-gesture trace from one or more modifier keys, such as *Ctrl*, *Alt*, *Fn*, *Shift* or *Command* to a letter key corresponding to the physical keyboard hotkey conventions, for example, *Ctrl-C* for *Copy*. In a desktop environment frequently used commands such as *Copy* and *Paste* have mnemonic key identifiers indicating the hotkey shortcut.

Taking advantage of the flexibility of the pen, short CSs transform the physical keyboard hotkeys to a fluid form. A short CS is typically defined as a straight stroke. The user's input can be matched against a collection of pen-gesture templates: a user needs only to draw an approximate pen-gesture to invoke a command, as long as it is closer to the intended pen-gesture template than other templates in the database. A novice user starts out by tracing the key sequences. Over time, the pen-gesture becomes ingrained in the user's memory and can be quickly executed without looking much at the keys. The technique can be expanded to any arbitrary sequence of keys. For instance, Figure 1 shows the hotkey trace *Ctrl-Shift-E* (for *Track Changes* in MS Word).

Short CSs, being directly mapped from the physical keyboard hotkeys, have limitations due to their heritage. Hotkey commands typically consist of a very short sequence of keys (usually two) and some are designed to be easily reachable with one hand (e.g. *Ctrl+C* for *Copy* on a regular QWERTY desktop keyboard). This causes two problems. First, very short pattern sequences are confusable in the recognition process. Second, the pen-gestures of some different commands can be quite similar from a user's point of view. The pen-gestures *Ctrl-C* for *Copy* and *Ctrl-X* for *Cut* are in fact very close to each other (cf. Figure 1).

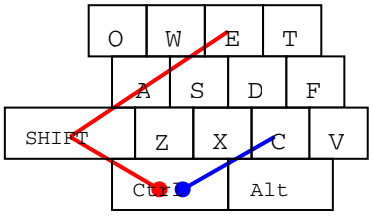

Figure 1. The line traces of *Ctrl-C* and *Ctrl-Shift-E* as short command strokes on a stylus QWERTY keyboard

Long Command Strokes

A long CS is defined by a stylus keyboard trace from a modifier key through a few or all letters in the name of a command. Figure 2 shows an example of a long CS for the action *Copy* whose template starts on *Ctrl* and intersects the letter keys *C-O-P-Y* in sequence. Since long CSs are richer in shape features, they should be more error tolerant than short CSs: as long as the user's gesture is geometrically similar to the long command pattern, the command can be recognized and executed (Figure 2).

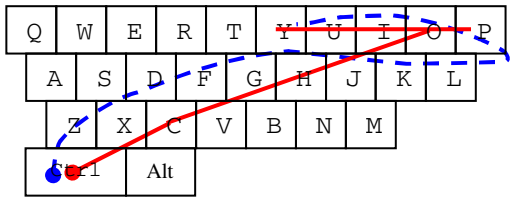

Figure 2. The solid red line shows the long CS template for *Copy* and the dashed blue line shows a user's actual pen-gesture of the command

EXPERIMENT 1

There are many issues with CSs that can be studied. In the first experiment we focused on a few basic questions: Can CSs offer any performance advantage over pull-down menus, the de-facto standard technique for entering commands on tablet PCs and smart phones? What are the quantitative tradeoffs between short and long CSs? Which technique do users prefer?

Procedure and Design

We recruited 12 volunteers for participation in the experiment. Their ages ranged between 20 and 50. The experiment was conducted on a 1GHz Tablet PC with a 12" TFT display with 1024 × 768 pixel resolution. The test software was written in Java, but the look and feel of the pull-down menus was set to the standard Windows XP look and feel which is used on the Tablet PC in other applications. The de-facto standard QWERTY keyboard layout was used for the stylus keyboard.

In the experiment the participants used a stylus to enter various commands in three conditions:

1. *Pull-down menu.* The participants entered commands by selecting them in a pull-down menu.

2. *Short command stroke.* The participants entered commands by gesturing short CSs, e.g. *Ctrl-C* for *Copy*.

3. *Long command stroke.* The participants entered long CSs, e.g. *Ctrl-C-O-P* for *Copy*.

The order of the three conditions was balanced among the participants in this within-subject experiment. With each condition the participants entered 10 (trials) × 5 (command types) = 50 commands. The trials were randomly shuffled within each condition so the participant could not predict which command would appear before the trial started. The participants were told to rest whenever they wanted between trials.

To start a new trial the participant had to click on the "Click for next command" button which brings out a new target command in the top panel of the experiment software. The participant then entered a command as fast and as accurately as possible. The command recognized by the software was displayed in another panel. If a wrong command was entered the trial had to be repeated until the correct command was entered. Incorrect entries (and their repeated trials) did not contribute when we calculated the reaction and total time.

In addition to a brief introduction and explanation to each technique in the beginning of the experiment, the participants were given a "cheat sheet" (a reminder table) in all conditions to glance at in case they forgot how to issue a particular command for the given condition. For example, the command *Copy* on the reminder table was "In Edit" in the pull-down menu condition, "Ctrl-C" in the short CS condition, and "Ctrl-C-O-P" in the long CS condition.

The five types of commands tested in the experiment were drawn from the *File*, *Edit*, *View* and *Tools* menus in Microsoft Word 2002, and their command sequences were *Ctrl-C, Ctrl-P, Ctrl-T, Ctrl-Shift-E* and *Ctrl-F* for the short CSs, and *Ctrl-C-O-P*, *Ctrl-P-R-I*, *Ctrl-T-H-E*, *Ctrl-T-R-A* and *Ctrl-F-I-N* for the long CSs. We chose to resemble MS Word in the pull-down menu condition because it is a commonly used application. All participants had experience in using the pull-down menus in MS Word. All items that had a default hotkey assigned by MS Word in the *File*, *Edit*, *View* and *Tools* menus were inserted into the set of commands the system could recognize. Consequently the recognition system contained a realistic number of distracters.

The selection of these five types of target commands in the study was biased in favor of the traditional pull-down menus. Only one of the five target commands was located in a submenu which obviously was more difficult to reach in the pull-down menu condition. CSs in contrast are not hierarchical so a submenu item in a pull-down menu is not necessarily more difficult for CSs. In comparison MS Word 2002 by default contains 117 top-level menu items and 136 submenus item (56 in the *AutoText* sub-menu).

Results

Selection Time

Selection time was defined as the time it took to issue a command, from pen down to pen up for both the pull-down menu and CS conditions. Figure 3 shows the successful selection time with each of the three methods as a function of the trial number. Repeated measures variance analysis shows that the selection time difference between the three methods was statistically significant ($F_{2, 22} = 99.9$, $p < .0001$). *Post hoc* tests indicate all pair-wise comparisons were significant ($p < .0001$). Taking the average of the last four trials as an example, short and long CSs were 3.8 and 1.6 times faster than the pull-down menu method respectively.

There was a significant interaction between different commands and the three methods ($F_{8, 88} = 39.38$, $p < .0001$, Figure 4). In particular, although the pull-down menu was slower for all commands tested, it was particularly slow with *Thesaurus*, which was a nested sub-menu item *(Tools-Language-Thesaurus)*.

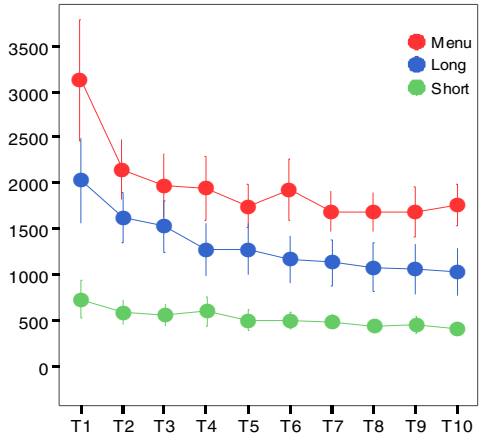

Figure 3. Mean and 95% confidence interval of selection time (ms) of the three methods as a function of trial number.

Reaction Time

It is conceivable that although CSs were faster than pull-down menus, they may take longer to prepare before the movement starts. We hence measured reaction time, which was defined as the time duration from the moment a new target command was presented to the moment of the first stylus contact with the pull-down menu or the stylus keyboard when selection time starts. Since occasionally the participants could be distracted or wanted to ask questions during this period, resulting in an excessively long reaction time, we truncated these cases (two cases) at 32676 ms, which was the largest number our statistics software could accept. There was no statistically significant difference

between the three methods in reaction time ($F_{2, 22} = 0.35$, $p = .71$). Reaction time decreased significantly with practice ($F_{9, 99} = 20.9$, $p < .0001$) particularly during the first two trials, and eventually stabilized at or just below 2000 ms. It is interesting to note that although novel to the participants, the CS methods did not have a longer reaction time than pull-down menus.

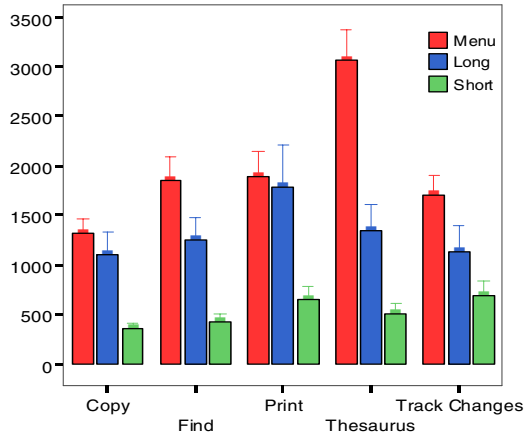

Figure 4. Selection time (ms) of the three methods and different commands.

Total Time

Total time was the sum of reaction time and selection time, from the moment a new target command was presented to the moment the system received a command.

Repeated measures variance analysis showed that the total time difference between the three methods was also statistically significant ($F_{2, 22} = 21.57$, $p < .0001$). *Post hoc* tests indicate all pair comparisons were significant ($p < .0001$).

The average total time of long CSs was similar to that of the pull-down menu in the beginning, but decreased to about two thirds of it. The differences between the three methods were quite large after a few trials of practice. Taking the average of the last four trials as an example, short and long CSs were 1.8 (1840 ms) and 1.3 (2609 ms) times faster than the pull-down menu (3380 ms) method respectively. It is remarkable that the time for command selection using the pull-down menu, whose layout and components were identical to Microsoft Word hence should be familiar to most of our participants, also decreased significantly during the first few trials but was still overall much longer than CSs both in selection time and total time.

Error

The error rates were 2.8% for pull-down menus, 6.5% for short CSs and 3.5% for long CSs. Repeated measures variance analysis shows that error rate differed significantly across methods ($F_{2, 22} = 6.12$, $p = .0077$).

Fisher's PLSD *post hoc* tests indicated that short CSs were significantly more error prone than long CSs and pull-down menus ($p < 0.05$) but the difference between long CS and pull-down menus was not significant ($p = .64$).

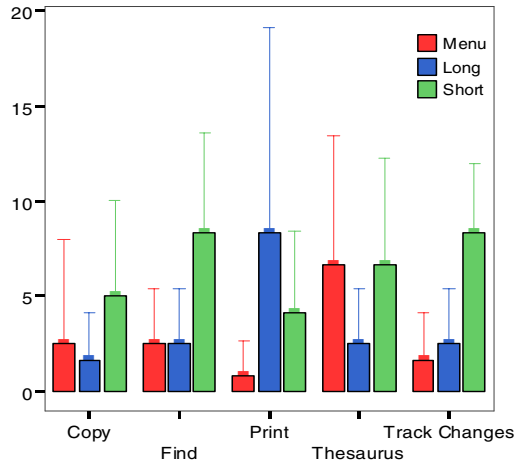

Figure 5. Error rates (%) in different conditions.

Note that (Figure 5) when selecting a sub-menu command *(Tools-Language-Thesaurus)*, the pull-down menu was no better than short CS and worse than long CS in error rate. Note also that the error rate of long CS was comparable to pull-down menu except in the case of the *Print* command. This was because another command in the lexicon, *Go To (Ctrl-G-O-T-O)*, forms a similar pattern as *Print (Ctrl-P-R-I)* on the QWERTY layout. This confusion would not arise with CSP, to be described in the next section.

User Ratings

No participants preferred pull-down menus and they all consistently rated pull-down menus as the most physically demanding technique. Five participants preferred long CS, five preferred short CS, and two stated that they preferred different modes for different situations, using short CS for very frequent actions such as *Copy* and using long CS for less frequent commands. When asked for their reasons for preferring a specific technique, participants who preferred short CS said they were already familiar with the typing pattern of traditional hotkeys. Participants who preferred long CS stated better accuracy and/or the ease of remembering the commands as the main reason for their preference.

In conclusion, CSs performed much faster than pull-down menus. While short CSs were the fastest, they were also the most error prone. Long CSs were faster than pull-down menus without significant accuracy loss. Users subjectively preferred CSs but differed in their view of short vs. long CSs. Overall CSs are clearly a viable selection alternative that can co-exist with pull-down menus for pen-based interfaces.

COMMAND STROKES WITH PREVIEW

Although CSs were rather successful in Experiment 1, through the experiment and our own use experience we realized that novel, interesting and possibly advantageous

improvements could be made. The result of our second design iteration is *command strokes with preview* (CSP).

Basic Idea

The overall goal in CSP was to give the user more flexibility and more certainty when using command strokes. For flexibility, each command can be entered with as long as, or as short as, a stroke on its complete path (e.g. *Command-C, Command-C-U* or *Command-C-U-T* can all do *Cut*), as long as the stroke is unambiguous to the pattern recognizer. Consequently CSP effectively merges short and long command strokes. For certainty, the system displays what the command would be (a *preview*, see Figure 6) if the pen would be lifted from the current location. Through testing we found it better to display a preview only if the stroke movement is relatively slow.

Example

Suppose the user wants to issue the command *Copy*. The user starts by landing the pen on the *Command* key, and then drags it to the first letter key in the command, in this case the *C* key. Since *Command-C* matches another command (*Cut*) that is shorter and/or more frequently used, *Cut* is now previewed (Figure 6 top). Other commands that also match the sequence *Command-C*, in this case *Copy, Close* and *Comment*, are shown in a list of alternatives to the left of the center panel. To enter *Copy* the user either quickly slides the pen towards *Copy* in the leftmost box (see the Quick Pick subsection below) or continues to gesture towards the second letter key *O*. Since *Command-C-O* matches *Copy* the command *Copy* is now previewed (Figure 6 bottom). When the user lifts up the pen the *Copy* command is issued. We want to emphasize that CSP still uses a pattern recognizer. For instance if the user is gesturing a pen trace geometrically close to *Command-P-R-I* the command *Go To* could appear instead because from the pattern recognizer's point-of-view *Command-G-O-T-O* is very similar to *Command-P-R-I*.

Cancellation

An important feature when previewing is the ability to cancel the gesture. By dragging and releasing the pen over the semi-transparent cancellation icon (see Figure 6) the current gesture is cancelled and no command is executed.

Dynamic Visual Preview

An important design goal was to make preview as unobtrusive as possible when the user already knows the gesture for a command. Therefore pattern recognition and subsequent visual preview is only triggered if the user moves the pen slower than an empirically determined threshold. In our implemented system any movement slower than 2.5 letter keys per second triggers pattern recognition and visual preview. This check is performed by the system every 20 ms (50 Hz).

Quick Pick

Any commands shown in the alternative list to the left of the center panel (Figure 6) can be directly selected by quickly dragging the pen towards the command name. The system can unambiguously separate gestures from alternative list selections since once the pen tip leaves the keyboard area it does not constitute a valid gesture. Since movement dynamics is taken into account when deciding if pattern recognition and update of the preview should be performed, the alternative list will not suddenly change despite the user gesturing over the keyboard while heading towards the desired command. This functionality worked very well in practice, as shown in the results later.

An important aspect of all user interfaces is behavioral consistency. For this reason it is also possible to "quick pick" the currently previewed command (e.g. *Cut* in Figure 6 top). In other words, either lifting the pen from its current location or dragging the pen to the preview box results in the same command.

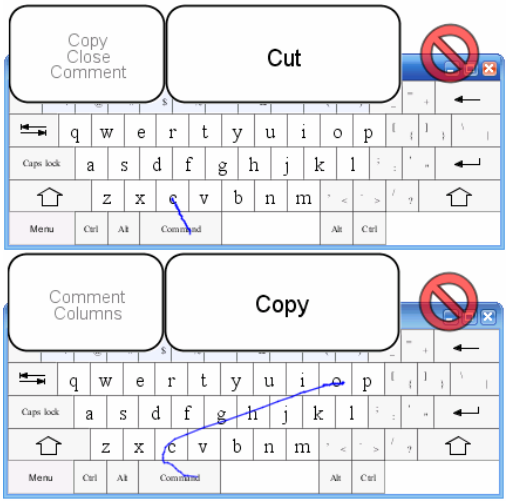

Figure 6. When the user gestures *Command-C* the command *Cut* is previewed (top). When the user gestures *Command-C-O* *Copy* is previewed (bottom).

Disambiguation

Some applications have many commands that start with the same letters. For instance, in MS Word *Cut*, *Copy* and *Comment* start with the letter *C*. When the system detects such a conflict a disambiguation procedure is invoked. First the system checks if any command has priority. For instance, if *Copy* is used more in the active application than *Cut* (as determined by the user's actions in the system) then *Copy* will be shown as the primary command previewed. If two commands have been used the same number of times the system prefers the shorter command, in this case *Cut*.

Conceptual Advantages

What You See Is What You Get

Preview allows the user to be certain about what command will be executed. This alleviates the fear of unintentionally invoking irreversible commands.

Minimizes User Effort

For many commands, one or two letters in a command would uniquely differentiate it from other commands. However without preview it is impossible for the user to discover the minimum number of letter keys needed for a given command without trial-and-error. With preview the user can learn the shortest path needed without the frustration of trial-and-error, hence the amount of motor effort needed in articulating the gestures is reduced. CSP therefore gracefully and flexibly merges short and long CSs.

Encourages Exploration

Without preview the user would need to try out commands to see if they existed in the active application. With preview frequently a command will appear either in the preview or in the alternative list after only one or two letters of the command is gestured. For instance, if the user gestures *Command-S* the command *Save* is likely shown in preview or in the alternative list along with the other best matching commands. If it is not, the user can continue gesturing on the path *Command-S-A-V-E* and the system will display *Save* in preview or in the alternative list at some point as long as *Save* is a valid command. If *Save* is still not displayed when the complete path of *Command-S-A-V-E* is finished, the user would know that *Save* is not available in this particular application. During this exploration the user can always stroke to the cancellation icon to abort the procedure and prevent an unintended command (if currently previewed) from being executed.

Reveals the Space of Possible Commands

A problem with the first iteration of CSs is discovery support: how does a user know the space of available commands except by consulting co-existing pull-down menus (which most likely exist due to legacy compatibility) or the application help system first? With preview, some commands other than the intended one will inevitably be shown in the preview or the alternative list while the user is gesturing. This increases the likelihood that the user notices other available commands. For instance, if *Print* is gestured as the sequence *Command-P-R*, *Properties* might appear as the next best matching command, informing the user that this command is available in the active application.

Benefits the Novice and the Expert

Since CSP consider the movement dynamics of gesturing, a true expert that knows how to quickly gesture the commands will not be disrupted with any visual feedback. If a user writes an unfamiliar command the natural slowdown of the pen motion causes preview to be automatically displayed, aiding novices and experts alike on just how much of the command stroke path needs to be completed for the command to be accurately recognized. Furthermore, if an expert user is very certain of a command stroke, the preview display can be ignored, reverting back to basic CS behavior.

EXPERIMENT 2

While there are many conceptual advantages with preview as outlined above which motivated the development of the technique, we were also concerned with possible adverse effects of preview. For example the display of preview could be distracting or over-demanding on the user's visual attention therefore significantly slowing down input speed and/or inflicting a higher error rate. Clearly an empirical study was needed to reveal any measurable *performance* impact of preview, particularly an adverse performance impact if any. Note that not all of the conceptual advantages outlined above, such as the ability to explore and discover, should necessarily result in measurable performance differences.

Procedure and Design

We recruited 16 volunteers for this within-subject experiment. None of them had participated in Experiment 1. The experiment was carried out on a Fujitsu-Siemens Tablet PC with a screen set to landscape orientation and with a screen resolution of 1024 × 768 pixels. The QWERTY keyboard layout was used for the stylus keyboard.

In the experiment participants used a stylus to enter commands in one of two conditions:

1. *No Preview*. The participant entered commands with the preview interface (preview, alternative list and cancellation icon) disabled. When the participant lifted the pen the recognized command was displayed above the keyboard.

2. *Preview*. The participant entered commands with the full implementation of the preview interface as described earlier.

In this experiment we decided to simulate realistic document editing in which commands were interleaved with common word processor (MS Word) tasks. Our goal was to be able to observe and measure both speed and error of the two versions of CSs. We developed two task scripts and instructions so the second condition would not repeat the same script. Our system was made to work with real applications in MS Windows. For example, a part of the first script read: "Scroll down to the bottom of the document. Invoke *Paste*." If the participant correctly activated *Paste* the contents of the clipboard would be pasted into the active document. All 114 commands in MS Word 2002 available in the main pull-down menus (e.g. *Open, Paste*, etc.) and all other menu items that had an assigned hot key (e.g. *Thesaurus, Visual Basic Editor*, etc.) were implemented and could be recognized by the experimental system. 10 commands were used in the first script: *Open, Properties, Copy, Paste, Select All, Word Count, Date and Time, Undo, Track Changes, Print* and 10 in the second script: *New, Styles and Formatting, Symbol, Paragraph, Font, About Microsoft Word, Versions, Page Setup, Break, Close*.

If a user made a mistake in following the instructions, the user was asked to repair the mistake. For example, if the

user accidentally executed the command *Styles and Formatting* to the word processor, the user was asked to close the panel that appeared.

After a brief demonstration of the system the participant was asked to follow one of the scripts with preview either disabled or enabled. After the script had been repeated 10 times, the participant was asked to follow a second script and test command strokes with the second preview type. The order of the two methods and the two scripts were balanced among the participants.

Results

Error

The average error rate with the *Preview* condition (7.5%) was lower than with *No Preview* (11.3%). However the difference was not statistically significant ($F_{1, 15} = 2.0$, $p = 0.178$).

Selection Time

Selection time was calculated as the time duration from pen-down to pen-up when articulating a correctly recognized command gesture. Repeated measures variance analysis showed that the difference in selection time was not statistically significant ($F_{1, 15} = 0.207$, $p = .656$), see Figure 7. There were considerable individual differences in performance. For instance, the fastest participant had an average selection time of 1126 ms with *Preview* and 1583 ms with *No Preview* while the slowest participant had an average selection time of 2451 ms with *Preview* and 3900 ms with *No Preview*.

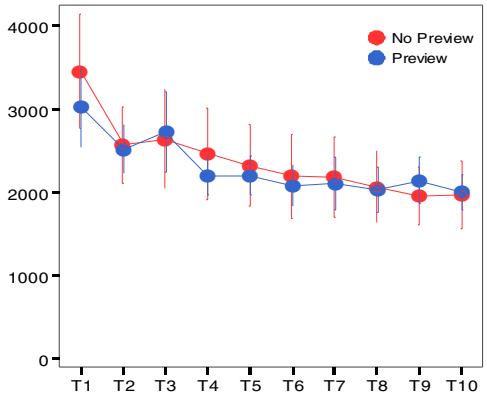

Figure 7. Mean and 95% confidence interval of selection time (ms) as a function of trial number.

Trial Completion Time

Trial completion time was defined as the time taken to complete one repetition of one of the scripts with 10 commands. Average trial completion time was 105.2 seconds with the *No Preview* method and 105.6 seconds with *Preview*. The difference was not statistically significant ($F_{1, 15} = 0.004$, $p = .952$).

Trace Lengths

Trace length, the distance the pen traveled over the keyboard, was measured in multiples of key width. If only the minimum paths were gestured, the average trace length of the commands tested would be 9.0 keys. If the complete paths were gestured the average trace length of the tested commands would be 27.3 keys. Results show that the traces in the *Preview* condition were significantly shorter (10.7 keys) than the traces in *No Preview* (15 crossed keys). The 40% difference was statistically significant ($F_{1, 14} = 31.7$, $p < .0005$). Evidently participants took advantage of the visual feedback and did not over-specify the gestures much.

Quick Pick

In *Preview* two participants used quick pick almost exclusively (96% and 93% of the responses respectively). One participant used quick pick ¼ of the time. The other participants used quick pick considerably less (0-5%). Pearson's r showed no significant correlation between quick pick usage and error rate ($r = -.139$, $n = 16$, $p = .608$, two-tailed).

User Ratings

After each condition participants were asked to rate their confidence on a 1-7 scale (1 = "Very unconfident", 7 = "Very confident"), and after the experiment they were asked to rate their preference of each method on a 1-7 scale (1 = "Strongly dislike it", 7 = "Strongly prefer it"). Friedman's repeated measures non-parametric test showed that neither confidence ($\chi^2 = 2.273$, $df = 1$, $p = .132$) nor preference ($\chi^2 = 2.571$, $df = 1$, $p = .109$) varied significantly between the methods.

The comments from the participants in the study were positive towards both interfaces. One participant declared "Wow! This is amazing! How can it know what I want to write?" One participant that really liked the preview version stated that "without preview I felt unsure if I was doing the right thing. With it enabled I felt I was guided [by it]" (translated from Swedish).

In summary, Experiment 2 did not show any adverse effect with CSP that concerned us. It also revealed that participants could take advantage of some features of CSP, such as using quick pick and taking a shorter stroke path due to the guidance of the preview display. On the other hand, when the same task procedure was repeated in succession as in this experiment, hence intensifying the participants' familiarity with the sets of command strokes to an "expert" level, neither speed nor accuracy was significantly different between the two conditions. We hypothesized that users might take greater advantage of the preview functions when they encounter commands that are new or unfamiliar, which frequently happens in a real use situation.

EXPERIMENT 3

We decided to conduct a follow-up study with the same participants as in Experiment 2 and investigate how users

familiar with the technique tackle new commands they have not previously gestured – with and without preview.

Procedure and Design

The same 16 participants in Experiment 2 were recruited the week after to take part in Experiment 3. They were asked to enter commands in one of two conditions: *Preview* or *No Preview* with the same properties as described in Experiment 2.

Two sets of 10 commands were used. The sets of commands and the experimental order of the two conditions were balanced. The commands were randomly chosen from MS Word with the constraint that they had not been previously used in Experiment 2. The commands in the first set were: *Fullscreen, Office Clipboard, Table AutoFormat, Macros, Theme, Text Box, Borders and Shading, Drop Cap, Word Perfect Help, Ruler*, and in the second set: *Object, Thesaurus, Draw Table, Reveal Formatting, Office on the Web, Heading Rows Repeat, Hide Gridlines, Bullets and Numbering, Find, Paste as Hyperlink*.

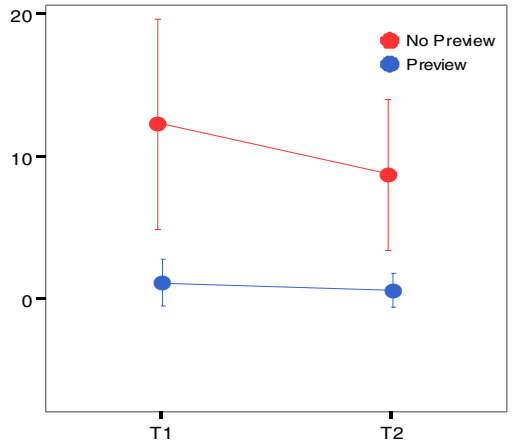

Figure 8. Mean and 95% confidence interval of error rate (%) as a function of trial number.

In one condition the participants entered all commands from a first set of commands and repeated the set once again. If a mistake was made the participants were asked to try again. Next the procedure was repeated in the other condition with the second set of commands.

Results

Error

Error rate differed dramatically between the conditions with average error rates as low as 1% in both trials in the *Preview* condition (Figure 8) but on average 10.5% in the *No Preview* condition. Repeated measures variance analysis showed that the difference in error rates between the conditions were statistically significant in the first trial ($F_{1, 15} = 8.99$, $p = .009$) as well as in the second ($F_{1, 15} = 9.22$, $p = .008$). The results show that users benefited from preview when executing unfamiliar command strokes.

Selection Time

There was no significant difference found in selection time (as defined in Experiment 1) in either trial 1 ($F_{1, 15} = 2.815$, $p = .114$) or trial 2 ($F_{1, 15} = 3.624$, $p = .076$), see Figure 9. The slower selection times compared to the ones in Experiment 1 and Experiment 2 are not surprising given that the majority of the commands tested in this experiment were longer and less frequently used.

Trace Lengths

The trace lengths (as defined in Experiment 2) in *Preview* condition were significantly shorter than in *No Preview* condition in both trial 1 ($F_{1, 14} = 25.043$, $p < .0005$) and trial 2 ($F_{1, 15} = 25.671$, $p < .0005$), see Figure 10. Clearly visual preview aided the participants in gesturing shorter CSs.

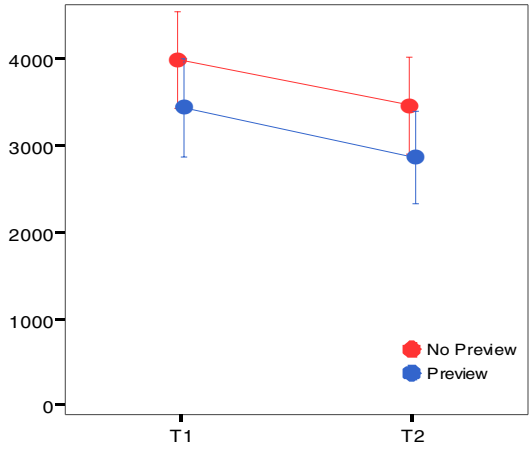

Figure 9. Mean and 95% confidence interval of selection time (ms) as a function of trial number.

User Ratings

After Experiment 3 the participants were asked which method they preferred and why. All participants preferred the *Preview* method. When asked to explain their preference all participants stated that with the *Preview* method they knew when they could stop and lift up the pen. One participant stated that dyslexia made it difficult for him to spell out the commands without visual guidance.

In summary, Experiment 3 clearly demonstrated the advantage of CSP when dealing with unfamiliar commands, as indicated by shorter stroke path and higher accuracy.

DISCUSSION

Experiment 1 showed that both short and long CSs are significantly faster than pull-down menus. Between the two, short CSs were faster but more error prone than long CSs. Short CSs based on traditional keyboard shortcuts are only applicable to menu items that have a keyboard shortcut assigned, whereas long CSs can be applied to all commands in an application. Long CSs should also be easier to remember and use, since they are based on the name of the command instead of a randomly assigned keyboard sequence such as *Ctrl+Y* for *Redo*.

With the preview display, CSP simultaneously takes advantage of both short and long CSs. CSP is still based on command name traces on the keyboard, but with the visual preview the user knows how much of the entire trace needs to be gestured for a command to be recognized. Experiments 2 and 3 showed that users' trace lengths were significantly shorter when using preview (also Figure 10).

A critical concern with visual preview is whether it demands so much visual attention that it interferes with expert users' speed performance. Experiment 2 showed that the CSP preview mechanism, which was carefully designed not to distract fast users, did not impede users' performance for well practiced familiar commands. For unfamiliar commands tested in Experiment 3 the visual preview did not only leave participants' speed performance unchanged, but also significantly reduced error rates. Overall, our experiments show that visual preview has real significant benefits and does not slow down input speed.

RELATED WORK

Many researchers have previously tackled the pen-based command selection problem. For instance, Kobayashi and Igarashi [7] demonstrated a technique that makes pull-down menu traversal easier when navigating through sub-menus, and Kurtenbach et al. [8] have developed a technique called Hotbox which combines linear, radial and pop-up menus to create a graphical user interface that can handle a large number of commands for the Maya modeling application.

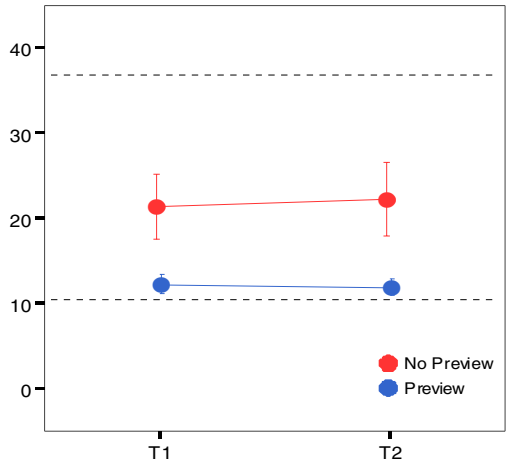

Figure 10. Mean and 95% confidence interval of trace length (in key widths) as a function of trial number. The dashed bottom and top reference lines indicate the minimum and maximum possible trace length.

Pie menus have been demonstrated as a competitive alternative to pull-down linear menus [3]. Marking menus, studied and advocated by Kurtenbach and Buxton [6], further improve pie menus. Embodying a critical thought in UI design, marking menus are pie menus augmented with a pen-gesture recognizer that encourages behavior transition from novice to expert use. Novice users select items in the pie menu structure as if using a regular pie menu (with delayed display). Over time, users learn the angular pen-gestures for selecting a command. This allows expert users to quickly flick the pen-gesture of the command without needing to visually traverse a pie menu hierarchy. To encourage users to learn the fast mode of gesturing commands instead of using slower visual feedback-based navigation, marking menus do not "pop-up" until after a time delay. A problem with marking menus observed by Zhao and Balakrishnan [15] is that some selections are ambiguous when marks are articulated independent of scale. Zhao and Balakrishnan investigated the use of consecutive single line marks instead of compound marks in marking menus and found that single line marks are unambiguous and more compact. However, it remains an open question whether users will memorize a sequence of disconnected single line marks as easily as compound marks that can be perceived and remembered as a whole. Another variant of the marking menu is FlowMenu [4]. Originally made for wall-sized displays, FlowMenu combines the Quikwriting text entry method [11] with marking menus. Although a practical method for many specialized applications, the down-side of the method is that the pen-gestures are long and complicated.

The contrast between CSs and marking menus is interesting and multifaceted. First, marking menus replace linear menus while CSs are designed to co-exist with or complement linear menus. With CS interfaces, the user can use the traditional pull-down menus to explore the existence of certain functions or commands (although with CSP one can also often discover CSs by trying out tracing the letters of a command and observe the visual preview) but rely on CSs to efficiently evoke known commands. Second, the learning mechanisms in the two systems are different. In CSs the user incrementally learns the gestures on the keyboard with use, starting by slowly tracing the keys with visual guidance then, over time, gradually (partially) transitioning into open-loop gesturing by recall. The graphical keyboard is always present as a visual map. In contrast, using the marking menu novice-expert bridging technique, the transition is binary – either with the displayed (and delayed) menu hierarchy or gesturing the mark without any visual assistance. This may force the user to memorize the gesture faster at the cost of reduced novice performance due to the delay. On the other hand, marking menus have a relatively easy transition from browsing commands via the displayed menu to selecting commands by gestures. In this regard, CSs requires an additional conscious step by the user to transition from menu browsing to gesturing CSs, although the visual preview system in CSP may encourage exploration.

In comparison to all menu selection methods, one advantage of CSs is consistency. Commands tend to be named in a similar manner in all user interfaces but are often placed differently in menu structures, shifting locations from application to application. Hotkeys for the same function can also vary between applications. The flat

hierarchy with CSs allows hundreds of commands to be specified directly by the user rather than being accessed from browsing a hierarchy. Large capacity in a small space is another advantage with CSs. Since a stylus keyboard does not require much screen estate, CSs can be used in specialized domains, such as Unmanned Aerial Vehicle (UAV) control [12], where a large number of commands need to be entered on a size-constrained handheld computer.

Another class of command articulation techniques is free-form pen-gestures, such as the Rubine recognizer [13]. Free-form pen-gestures are often "arbitrary" pen-gestures that denote different actions in a user interface. The similarity aspects of pen-gestures have also been studied [10]. In general, free form pen-gestures are limited in the number a user can remember and reproduce, due to their arbitrary definition to the user. In contrast to free-form pen-gestures, CSs use the keyboard as a mnemonic map.

It is clear that each technique has its own pros and cons. Users are familiar with pull-down menus and they are backwards compatible with virtually all existing desktop and mobile programs. Special techniques such as the Hotbox [8] can be used when an application needs to provide access to over 1,000 commands but it takes a large amount of screen space. Free-form pen-gestures are most advantageous in application domains that have a strong convention, such as copy editing, or in places where pen-gestures can be highly metaphorical, such as crossing a series of words to cause the words to be deleted.

CONCLUSIONS

Based on the conceptual analysis and the empirical study presented in this paper, we expect CSs to be a very useful complement to pull-down menus in future mobile computing devices. Pull-down menus tend to be slow and tedious, but offer an effective way for the user to discover the commands available in an application. Users will continue to use pull-down menus to access a large number of infrequent functions. For a known command, particular frequently used commands such as *Cut*, *Copy*, *Paste*, *Find*, and *Print*, CSs provide fast, fluid and convenient access when used in conjunction with a stylus keyboard. Our evaluation shows that on average CSs can be much faster than pull-down menus, particularly if the commands are located in sub-menus. In the first experiment we found a speed-accuracy trade-off between long and short CSs. The latter is faster but more error prone. We further developed command strokes with preview (CSP) which conceptually could improve certainty, reduce effort, and encourage exploration and command discovery. Empirically CSP proved capable of reducing users' gesture lengths without impacting speed or accuracy for familiar CSs. For unfamiliar commands, preview dramatically reduced error rate. These results indicate that as long as the input interface does not force users to look at the visual feedback, high input speed and low error rates can be obtained even though visual feedback is introduced that guides novice users towards the expert mode.

ACKNOWLEDGMENTS

This work was in part sponsored by the Santa Anna IT Research Institute (SICS Linköping) and IBM Research. We thank Alison Sue for her assistance.

REFERENCES

1. Accot, J. and Zhai, S. Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. *Proc. CHI 1997*, pp. 295-302.
2. Bernard, P., Hammond, N., MacLean, A. and Morton, J. Learning and Remembering Interactive Commands. *Proc. CHI 1982*, pp. 2-7.
3. Callahan, J., Hopkins, D., Weiser, M. and Shneiderman, B. An Empirical Comparison of Pie vs. Linear Menus. *Proc. CHI 1988*, pp. 95-100.
4. Guimbretière, F. and Winograd, T. FlowMenu: Combining Command, Text and Data Entry. *Proc. UIST 2000*, pp. 213-216.
5. Kristensson, P.O. and Zhai, S. SHARK^2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. *Proc. UIST 2004*, pp. 43-52.
6. Kurtenbach, G. and Buxton, W. The Limits of Expert Performance Using Hierarchic Marking Menus. *Proc. CHI 1993*, pp. 482-487.
7. Kobayashi, M. and Igarashi, T. Considering the Direction of Cursor Movement for Efficient Traversal of Cascading Menus. *Proc. UIST 2003*, pp. 91-94.
8. Kurtenbach, G., Fitzmaurice, G., Owen, R.N. and Baudel, T. The Hotbox: Efficient Access to a Large Number of Menu-items. *Proc. CHI 1999*, pp. 231-237.
9. Landauer, T.K. and Nachbar, D.W. Selection from Alphabetic and Numeric Menu Trees Using a Touch Screen: Breadth, Depth, and Width. *Proc. CHI 1985*, pp. 73-78.
10. Long, A.C., Landay, J.A., Rowe, L.A. and Michiels, J. Visual Similarity of Pen Gestures. *Proc. CHI 2000*, pp. 360-367.
11. Perlin, K. Quikwriting: Continuous Stylus-Based Text Entry. *Proc. UIST 1998*, pp. 215-216.
12. Quigley, M., Goodrich, M.A. and Beard, R.W. Semi-Autonomous Human-UAV Interfaces for Fixed-Wing Mini-UAVs. *Proc. Intl. Conf. Intelligent Robots and Systems 2004*, pp. 2457-2462.
13. Rubine, D. Specifying Gestures by Example. *Proc. SIGGRAPH 1991*, pp. 329-337.
14. Zhai, S. and Kristensson, P.O. Shorthand Writing on Stylus Keyboard. *Proc. CHI 2003*, pp. 17-24.
15. Zhao, S. and Balakrishnan, R. Simple vs. Compound Mark Hierarchical Marking Menus. *Proc. UIST 2004*, pp. 33-42.