FloodKey: increasing software keyboard keys by reducing needless ones without occultation

Geoffroy AULAGNER, Romain FRANÇOIS,
Benoît MARTIN, Dominique MICHEL
LITA – UFR MIM
Université Paul Verlaine - Metz
Ile du Saulcy, BP 80794, 57012 Metz cedex
FRANCE
{benoit.martin, dmic}@univ-metz.fr

Mathieu RAYNAL
IRIT – équipe IHCS
Université Paul Sabatier
31062 Toulouse Cedex 9
FRANCE
raynal@irit.fr

Abstract: - Floodkey is a software keyboard in which some keys are increased in size while needless ones are decreased. It uses a graph of sites that generates the keys. Each time a key is pressed, according to the probabilities of the key to be pressed, the sites are located by the Fruchterman-Reingold algorithm that maintains the topology of the keyboard. Then the keyboard appears with no overlapping as a weighted Voronoi diagram. A simulation shows a mean increase of 97% and 70% in size for the key to be pressed with and without the Fruchterman-Reingold algorithm. A preliminary usability test conducted with and without increasing the keys shows similar performance after 4 sessions of 3-minutes use. But the users found the combination between the Fruchterman-Reingold algorithm and the weighted Voronoi less pleasant.

Key-Words: - Text input, Software keyboards, Fitts' law, prediction system.

1 Introduction

Recent years have seen the emergence of mobile devices. They are used in various tasks that frequently require to enter texts. The availability of touchscreen devices has enabled the emergence of new solutions for text input: the gesture keyboards and the software keyboards. In this paper, we will focus on software keyboards that use the concept of keys to enter letters. The interaction is usually done by pointing to the keys with a finger or a stylus.

To facilitate the taping, some works proposed to add visual cues helping users to find and to reach the key. For example, Raynal et al. proposed new interactions based on animation as expanding targets or fisheye view [17]. Experiment shows an average gain of 25% of time. Results show that distance was 14.85% less with fisheye keyboard than standard keyboard. With a linguistic model, Magnien et al. proposed to contrast the keys that may be selected [15]: the size of the font is proportional to the probability of the key to be hit. An empirical evaluation shows a theorical gain of 50% for a novice user. On a personal assistant, a gain of 60% was measured [14].

But to improve the performance of user input, it is also possible to consider the motor efficiency of the pointing task. With a single pointing device, the pointing time *T* of the user can be modeled by Fitts' law [3]: T = a + b× log2 (D / W + 1)). The time to point depends on constants *a* and *b* that comes from the pointing device. But it also depends on the distance between the starting point and the target (*D* in Fitts' law) and the width of the target (*W* in Fitts' law). Thus, to improve performance in text entry, we can influence the distance between keys and the width of the keys.

From this law, several improvements have been proposed. Static or dynamic rearrangements have been introduced to reduce the distance between the keys to hit. Likewise, resizing keys has been proposed to increase their width. But resizing keys yields the problem of overlapping. Indeed, the keys are usually glued to each other and inflating some keys leads to overlap with the adjacent ones. This raises problems of readability and accessibility of these keys.

This paper proposes a new approach for resizing keys by modelling the keyboard as a Voronoi diagram. According to the probabilities of the keys being pressed, we move the sites that generate the keys and we apply a weighted Voronoi diagram. This principle guarantees the non-overlapping of the keys and a key area proportional to its probability of being pressed.

2 Improving Software Keyboards

2.1 Distance between keys

Many works proposed static optimization of « 26 characters » layout [21][22]. That would decrease the input time by reducing the distance covered between sequential keystrokes. Experiments prove the gain of performance [12]. But an expert user with a standard layout becomes novice again with the new layouts. So,

these software keyboards are confronted to the standardization of AZERTY/QWERTY's family keyboards. Because of the learning cost, these new layouts do not easily reach the general public.

Dynamic reorganization can also be used to reduce the distance between keys as with Sibylle [20] and FOCL [2][13]: when a key is pressed, the layout changes according to a linguistic model. Theorical performance increases significantly but such system can disorient the users who can not anticipate the next keystroke and must perpetually look for the characters among the new character distribution.

To solve this problem and to keep the AZERTY/QWERTY's family layouts, Jahveri proposed to add local optional menu to a software keyboard [10]. Isokoski later re-designed the system, modeled its performance and ran two experiments to measure its performance [7]. A QWERTY layout was used: after putting the stylus onto a key, the user can continue with a stroke to eight directions around the key. A marking menu [11] is used to display the characters associated to the directions. The character of the key would be the first character entered and the character associated with the direction of the stroke would be the second character entered. For the text entry rate, an improvement of 26% is expected for expert users with a QWERTY layout [7].

Isokoski's menu was static. The same menu with the vowels, y, backspace and space, is available on every key. The rationale is that if the user is to learn the menu layout, it cannot change. Learning to efficiently utilize the 8 menu items is difficult enough as seen in Isokoski's data. However, further work has suggested to add linguistic optimization to the system so that the menu would present the most likely characters [18]. This proposal is known as KeyGlass. The first level of the menu has only 4 items that are displayed as keys between the keys. The menu is recursive making it possible to enter more than one character with one menu selection path. When entering text similar to the text that was used to build the prediction database, the stylus movements with this design are more efficient than in the design by Isokoski. Unfortunately, the number of key-menu combinations is large making it difficult to learn the menu locations. Expert performance with a soft keyboard is so good that systems that rely on visual search after each pressing a key probably cannot beat it except for special needs.

2.2 Size of the keys

CATKey proposes to use the Voronoi diagrams for a customizable keyboard [5]. In mathematics, a Voronoi diagram is a tiling of a metric space determined by distances to a specified discrete set of objects. In the simplest case, we are given a set of points *S* in the plane, which are the Voronoi sites. Each site *s* has a Voronoi cell, also called a Dirichlet cell, *V(s)* consisting of all points closer to *s* than to any other site. According to this, figure 1 (left) shows the initial tiling of CATKey and figure 1 (right) shows the tiling customized statically by the user by moving sites.

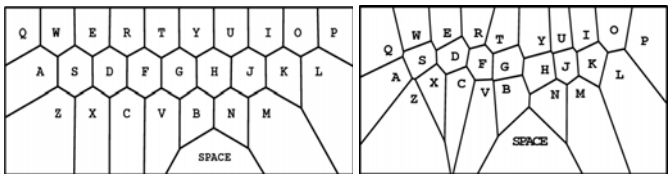

Figure 1. CATKey: the QWERTY layout with a Voronoi diagram and the layout customized by a user.

This solution is static but other methods propose to do this dynamically during the taping. While keeping the same layout, BigKey increases the size of the 4 keys that may be selected [1]. The pointing is easier and the scanning time should be reduced for novice users. The growth is limited by the layout: a key must not occult one neighbour key.

To solve this problem, SpreadKey dynamically recycles the improbable keys according to a prediction system [16]. The remapping of these keys creates ambiguous keys used to increase the key size of the most probable characters. Figure 2 shows examples of remapping of an AZERTY layout. A simple interaction enables to disambiguate the keystrokes in case of prediction error.

The hypothesis of the previous propositions is that a user might well prefer to enter text on a known and standard character layout. But to increase the size of the keys, one solution is also to reduce the number of the keys. One way is to display the entire keyboard on different pages accessible by tapping on a special key. For example the DotNote keyboard uses a 2-page layout [12]. The most frequent letters appear on the first page and a button allows the failover to the second page. On the two half-keyboards, the letters are arranged alphabetically. The keyboard is "multitap" because it needs two buttons to access to certain letters: one to change the page, then the one containing the desired letter. With this method, keys are bigger but novice users waste time to find characters on the different pages.

Reducing the number of keys can also be done by combining several characters to a key: it is called ambiguous keyboards. Certainly the best known is the 12-key phone keyboard. Disambiguation can be done explicitely. It is usually done by pressing many times the same key. But the use of a prediction module can mainly automate this process. This can be done at the letter level or at the word level word. At the letter level, LetterWise [12] or PNLH [6] rearrange the letters of a key to propose them in order of probability. At the word level,

WordWise of Eatoni Ergonomics [12] or T9® of Tegic Communications [9] offer one keypress per letter and perform a prediction of words after typing punctuation.

Many variants have been developed based on different groupings and different disambiguation techniques. Another system based on a traditional QWERTY layout, TouchPal [19], even offers a mixed disambiguation technique. Automatic disambiguation works with a linguistic model but the user can disambiguate by explicitly shifting to the desired letter.

3 FloodKey

The goal of FloodKey is to offer a resizable keyboard that sizes automatically the keys according to their probabilities to be pressed: some keys would grow while some other ones would shrink to save space and to avoid occlusion. Then FloodKey would be less limited by the number of resized keys compare to BigKey and the occlusion problem would be limited compared to SpreadKey. FloodKey is intended to use the standard AZERTY/QWERTY layout to ease visual search and to ease taping especially at learning stage. Our hypothesis remains that a user might well prefer to enter text on a known character layout rather than on a new soft keyboard where he must sometimes search for a character or press several keys to enter one given character.

As CATKey, FloodKey proposes to use the Voronoi diagrams but the tiling is modified dynamically after each keypress by sizing the keys according to their probabilities to be pressed. For the probabilities, we used a 5-grams. The model of N-grams is a technique to predict the next letter in a sequence, from N-1 preceding letters. As in CATKey, we use the Voronoi diagrams to modelize our keyboard. We consider the centre of the keys as the sites of a diagram of Voronoi. To modify the surface of the keys, we use two techniques:

- to use a weighted Voronoi diagram;
- to move the sites.

In mathematics, a weighted Voronoi diagram is a Voronoi diagram for which the Voronoi cells are defined in terms of a distance defined by some common metrics modified by weights assigned to the sites. In FloodKey, the distance between a point *pt* and a site *s* of probability *p* is equal to *dist(s, pt) + delta(p)* with *dist* the Euclidean distance and *delta* the following function:

$$delta : p \rightarrow \begin{cases} -p * F & \text{if } p > 0 \\ F & \text{if } p = 0 \end{cases}$$

with F a constant to tune the magnification. After informal testing, the constant F was set to *22*. But the magnification is limited by the sites: to maintain the topology of the keyboard, a site cannot belong to a cell of another site, otherwise a key disappear.

Then we combine it with moving the sites. We considered to use an existing algorithm, the algorithm from Fruchterman and Reingold [4]. This algorithm is a force-based or force-directed algorithm. Its purpose is to position the nodes of a graph in a two dimensional space by assigning forces among the set of edges and the set of nodes. In FloodKey, we transformed the probabilities in stiffness on the edges. The edge between s_1 and s_2 will have a stiffness of $2*Max(delta(p_1),delta(p_2))$ with p_1 and p_2 the probabilities of *s1* and *s2* respectively. The entire graph is then simulated as if it were a physical system. The forces are applied to the nodes, pulling them closer together or pushing them further apart. This is repeated iteratively until the system comes to an equilibrium state; i.e., their relative positions do not change anymore between two iterations. But in our case, to ensure a good reactivity of the visual feedback, we do not wait the equilibrium. The number of iterations of the algorithm was set to *60*. The main advantage of this algorithm is a high running time but it produces only a local minimum.

Figure 2 shows our implementation of FloodKey: the neighborhood graph (top) and the resulting Voronoi diagram (bottom). At the beginning of a word, the algorithm of Fruchterman and Reingold is not applied and a standard Voronoi diagram is used. The window is 800x220 pixels. Keys "", '.' and ',' are 60x60 pixels. Keys 'Bsp', 'Enter' and 'Shift' are 100x60 pixels and the space bar is 800x40 pixels. The dynamic area for letters, the hyphen and the apostrophe is 640x180 pixels.

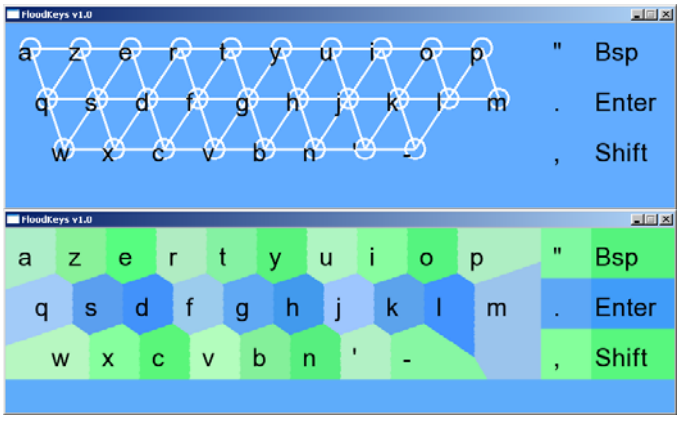

Figure 2. FloodKey without linguistic information at the beginning of a word.

Figure 3 shows FloodKey after entering "joue". The result of the 5-grams model is: n (0.0476), s (0.2619), r (0.5238), u (0.0714), m (0.0476) and t (0.0476). The other characters have zero probability. The sites moved (top) and the surfaces of the keys varie according to their probability of being hit (bottom). As visual cues, we used a proportional font to the probability as suggested in [15].

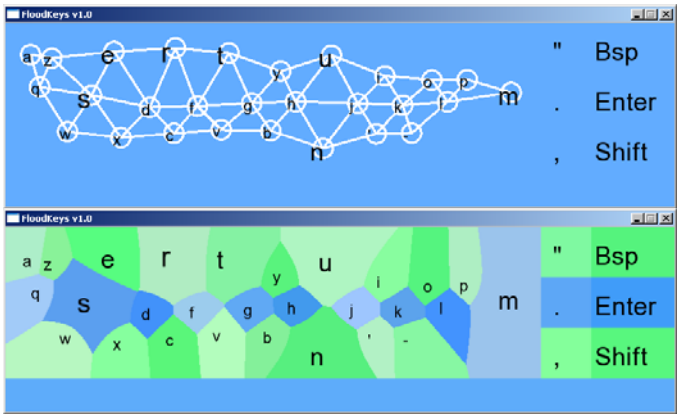

Figure 3. FloodKey after entering "joue": the graph of sites and the resulting weighted Voronoi diagram.

In addition, we used the same hue (green and blue) to visualize the rows of the keyboard despite the shifts of the keys. Into these rows, we alternated three different lights to distinguish the keys. If a key appears too small for the user, a local zoom is usable. While pressing a key, the computation of the distance is modified for that key and its direct neighbours: the function *delta* tends to *0* by an increment of 1 every 150 ms. A surface with a probability of zero increases while the others decrease. Then, the user can refine his/her selection.

4 Simulation

We conducted a simulation to evaluate the impact of FloodKey on the key size. We assume that in most cases the size has an effect on the width of the target (*W* in Fitts' law). We measured the surfaces of the keys in the Voronoi condition where the size of the keys do not change (*V*), the weighted Voronoi condition without the algorithm of Fruchterman and Reingold (*VP*) and in the weighted Voronoi condition with the algorithm of Fruchterman and Reingold (*F-VP*). This simulation was performed with a corpus of 110 French sentences for a total of 3144 letters. These sentences included only lowercase characters without accents. The distribution of word lengths in our corpus was 0.8% for words with one letter, 20.7% with two letters, 15.1% with three letters, 16% with four letters and 47.4% with at least 5 letters. This is important for the 5-grams. Each sentence has been entered automatically once, letter by letter.

Figure 4 shows the results for the surface of the key to be pressed. We note S the surface function. The line "x = 1" indicates the condition *V*. A mean increase of 97% appears with *F-VP* but only 70% with *VP*. That confirms the validity of applying the algorithm of Fruchterman and Reingold to move sites. In only 2.4% of cases, the key to hit is smaller than in condition *V*, ie. Ratio < 1. The main reason is a faulty prediction: the mean probability of such cases is only 0.07.

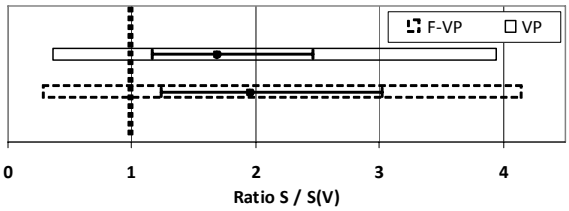

Figure 4. Ratio for the surface of the key to hit. For condition *V*, the ratio is 1.

Then we analyzed the surfaces of the keys according to their likelihood of being hit. Figure 5 shows the results. The line "y = 1" indicates the condition *V*. The condition *F-VP* appears better than the condition *VP* in all intervals. The variability of the resulting surfaces is partly due to the probability intervals that are used on the abscissa. The condition *F-VP* can be approached by a linear regression with a good correlation coefficient R^2 = 0.93: "y = 0.0844x+1.1432". The surface increases with the probability despite several peaks to explore.

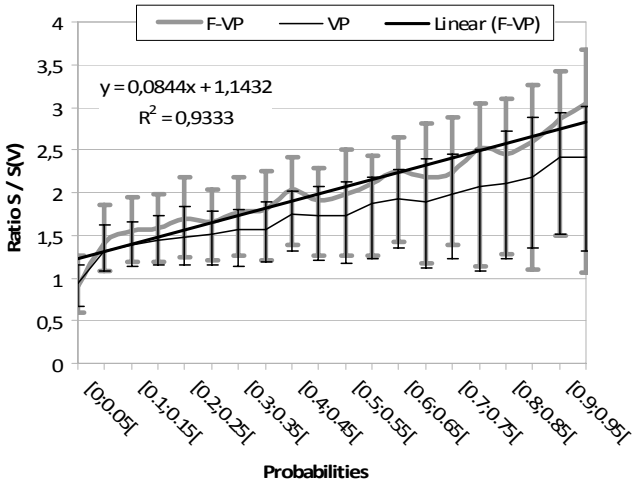

Figure 5. Ratio of the surfaces according to the probabilities. For condition *V*, the ratio is 1.

5 First Evaluation

The purpose of this experiment was to perform a reality check of the simulation results. To investigate the effects of the morphing on the user behaviour we collected data on FloodKey in the three conditions used in the simulation described before: *V*, *VP* and *F-VP*. Below, we describe this experiment in detail.

5.1 Description

Four participants, two female and two male, participated in the experiment. They were all right-handed and regular computer users. Each subject completed 4 sessions of transcription task. Within a session there were three three-minute blocks, one for each condition. The French sentences to transcribe were chosen

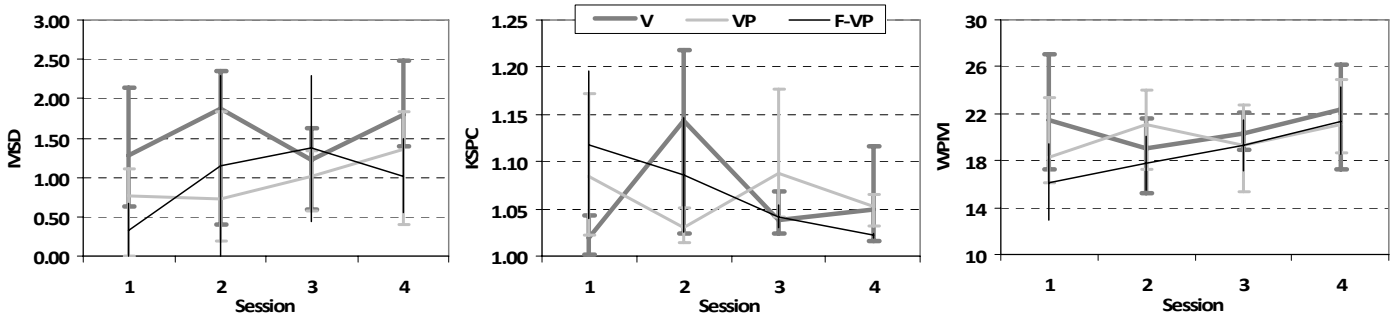

Figure 6. The minimum string distance, the keystrokes per character and the text entry rate.

randomly among the same corpus that was used in the simulation before. The sessions were self-scheduled and separated by at least two hours, but no more than one day. There was no practice before or in between the sessions. The order of the conditions was balanced between participants and sessions. They were instructed to transcribe the sentence as fast as possible while correcting errors that they noticed. Correcting errors was allowed using the backspace or manipulating the cursor.

The experiment was conducted using an Asus R1F TabletPC running Windows Vista. The computer had a 13.3" display with 1280x800 pixel resolution. The size of the keyboard was 178 x 49 millimeters. The stylus was used to point the keys. FloodKey was coded in C++ and the TimTester [8] application displayed the sentences, received the input and saved data for later analysis.

Given the small number of participants, tests for statistical significance would have been pointless. Our reporting below will be purely descriptive.

5.2 Error rate.

There were errors left in the transcribed sentences. Minimum string distance (MSD) is the number of character additions, substitutions, and deletions needed to make two strings identical. Even if the MSD of the *F-VP* condition yielded to a lower MSD in session 4, results are close for all conditions (figure 6 – left). No session effect appeared. For all conditions, between 0.5% and 2% of the characters in the transcribed sentences were erroneous. Users have well controlled their errors along the evaluation.

5.3 Effort

Due to errors and corrections participants entered more characters (13987) than the presented sentences (13292). This extra effort was measured as keystrokes per character (KSPC). The KSPC and the standard deviation of the *F-VP* condition decreased over the session and yielded to the lower KSPC and standard deviation in session 4 (figure 6 - centre). All users performed with a very low effort in condition *F-VP*.

5.4 Text entry rate

The results for text entry rate are summarized in figure 6 - right. The text entry rate was computed using the transcribed sentences except for the first character of each sentence and the "enter" at the end of the sentence which were excluded. Time spent on corrections was included. One word per minute equals 5 characters including spaces. In session 1, dynamic conditions (*F* and *F-VP*) give lower performance. That shows the surprise or difficulties for beginners when the size of the keys change especially with the *F-VP* condition. The algorithm of Frushterman and Reingold increases the difficulty of use. But a session effect appears with this *F-VP* condition that yields to similar performance for the three conditions in session 4. Further tests are needed to measure performance after the first 12-minutes of use.

5.5 User Impressions

The end questionnaire consisted of 2 assertions specific to conditions *VP* and *F-VP* and 3 assertions that were common to the three conditions (*V*, *VP* and *F-VP*). The users answered from 1 to 5, 1 for a total disagreement and 5 for a total agreement. Figure 7 shows the results.

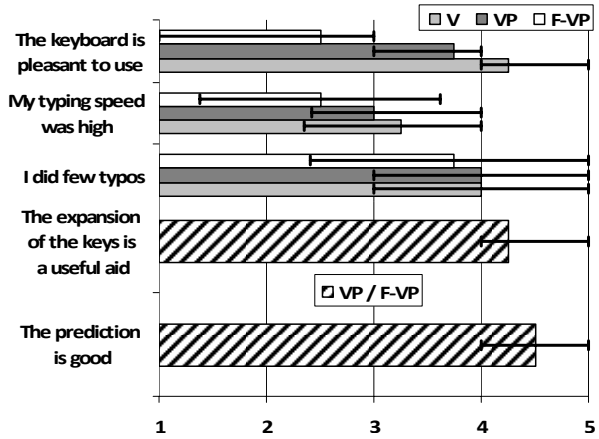

Figure 7. Questionnaire results.

In dynamic conditions, the users found the prediction good and the expansion of the keys helpful. In the three conditions, they felt to have done few typos which is confirmed by the logs. They also felt to be moderately fast especially in condition *F-VP*. That is not confirmed

by the logs in the last session since the performance is equal. Finally, they found the condition *V* the more pleasant but even if the logs do not show differences between conditions, they found the condition *F-VP* the least pleasant while logs show similar performance for the three conditions in session 4.

4 Conclusion and perspectives

We proposed a new approach for resizing keys by modelling the keyboard as a Voronoi diagram. It guarantees the non-overlapping of the keys and a key area proportional to its probability of being pressed. A simulation shows a mean increase of 97% in size for the key to be pressed. A preliminary test conducted with novice users doesn't show any improvement when the keys growth after 4 sessions of 3-minutes copy task. But the learning effect seems to favor FloodKey which must be confirmed by a longitudinal study.

For the future, we plan to enhance the algorithm that moves the sites and the distance computation in the weighted Voronoi diagram. Another way of improvement is to study other tilings as the TreeMap. Finally, users seemed disturbed by the dynamic of the keyboard: we must think about techniques to offer smoother transition between layouts.

References:

[1] Al Faraj, K., Mojahid, M. and Vigouroux N. *BigKey: A Virtual Keyboard for Mobile Devices.* In Proc. of HCII 2009, Springer-Verlag, LNCS, 2009, pp 3-10.

[2] Bellman, T. and MacKenzie, I.S. *A probabilistic character layout strategy for mobile text entry*. In Proc. of GI'98, 1998, pp 168-176.

[3] Fitts, P.M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 1954.

[4] Fruchterman, T. M. and Reingold, E. M. Graph drawing by force-directed placement. Softw. Pract. Exper. 21 (11), 1991, pp 1129-1164.

[5] Go, K. and Endo Y. CATKey: Customizable and Adaptable Touchscreen Keyboard with Bubble Cursor-Like Visual Feedback. In Proc. of INTERACT 2007, LNCS, 2007, pp 493-496.

[6] Gong, J., Haggerty, B., and Tarasewich, P. *An enhanced multitap text entry method with predictive next-letter highlighting*. In Proc. of CHI'05 Extended Abstracts, ACM Press, 2005, pp 1399-1402.

[7] Isokoski, P. *Performance of menu-augmented soft keyboards.* In Proc. of CHI 2004, CHI Letters, 6(1), 2004, pp 423-430.

[8] Isokoski, P., Text entry test package, http://www.cs.uta.fi/~poika/downloads.php

[9] James, C. and Longé, M. *Bringing text input beyond the desktop*. In Proc. of CHI '00 Extended Abstracts, ACM Press, 2000, pp 49-50.

[10] Jhaveri, N. *Two Characters per Stroke – A Novel Pen-Based Text Input Technique*. In G. Evreinov (ed.), New Interaction Techniques'03, Spring, University of Tampere Finland, 2003, pp 10-15. www.cs.uta.fi/reports/bsarja/B-2003-5.pdf

[11] Kurtenbach, G., and Buxton, W. *The limits of expert performance using hierarchic marking menus*. In Proc. of INTERCHI 1993, ACM Press, 1993, 482-487.

[12] MacKenzie, I. S., and Soukoreff, R. W. Text entry for mobile computing: Models and methods, theory and practice. In *Human-Computer Interaction*, Vol. 17, 2002, pp 147-198.

[13] MacKenzie, I. S. *Mobile text entry using three keys*. In Proc. of NordiCHI 2002, 2002, pp 27-34.

[14] Magnien, L., Bouraoui, J. L. and Vigouroux, N. *Mobile devices: soft keyboard text-entry enhanced by Visual Cues*. In Proc. of UbiMob '04, ACM Press, 2004, pp 158-165.

[15] Magnien, L., Bouraoui, J-L. and Vella, F. *Utilisation d'indices visuels pour l'aide à la saisie de texte sur PDA*. In Proc. of IHM 2003, ACM Press, 2003, pp 252-255.

[16] Merlin, B. and Raynal, M. *SpreadKey: increasing software keyboard key by recycling needless ones*. In Proc. of AAATE 2009, IOS Press, 2009, pp 138-143.

[17] Raynal, M. and Truillet P. *Fisheye Keyboard: Whole Keyboard Displayed on PDA*. In Proc. of HCII 2007, LNCS, 2007, pp 452-459.

[18] Raynal, M. and Vigouroux, N. *KeyGlasses: Semi-transparent keys to optimize text input on virtual keyboard*. In Proc. of AAATE 2005, 2005, pp 713-717.

[19] TouchPal. Available at: http://www.cootek.com/

[20] Wandmacher, T., Antoine, J-Y. and Poirier, F. *SIBYLLE: A System for Alternative Communication Adapting to the Context and Its User*. In Proc. of ASSETS'07, 2007, pp 203-210

[21] Zhai, S., Hunter, M. and Smith, B.A. *Performance Optimization of Virtual Keyboards*. Human-Computer Interaction, Vol 17 (2&3), 2002, pp 229-269.

[22] Zhai, S., Hunter, M. and Smith, B.A. *The Metropolis Keyboard - an exploration of quantitative techniques for virtual keyboard design*. In Proc. of UIST'00, 2000, pp 119-128.