Bubble Keyboard: A Gesture-Based Non-Physical Text Entry Method Designed for Minimalized Motion and Increased Speed Utilizing Microsoft Kinect Sensor

**Chinmay Nirkhe, Jacob O. Wobbrock**

AIM Research Group

The Information School

University of Washington, Seattle

Box 352840 Mary Gates Hall, Ste 370

Seattle, WA 98195-2840

chinmay.nirkhe@lakesideschool.org; wobbrock@uw.edu

**Abstract**

**Introduction**

We are investigating the successes and failures of non-physical keyboards in terms of their utilization of gesture-based inputs. We are using the Microsoft Kinect Sensor to monitor gestures and track motion of limbs. This study aims at advancing the techniques used in the industry for text entry with non-physical keyboards for the general populace; it also looks at the applications of such devices on populations who have a difficulty with regular desktop QWERTY keyboards such as those with difficult with fine motor control such as users affected with Cerebral Palsy or Parkinson’s.

Currently, the most common method of non-physical text entry used in the industry is a QWERTY based keyboard that is explained in the following section. This keyboard does not incorporate the capabilities of gesture based interactions which are more intuitive for the user.

Our research intends on constructing a revolutionary system that incorporates gesture based interactions to increase speed and minimalize motion. This keyboard, called the Bubble Keyboard, is a non-static keyboard that aims to overcome the following restrictions resultant of the QWERTY based keyboard.

The size of the

***Minimalized Motion***

The inner ring of probable letters are constructed such that the user has to move less to access the next letter, thus reducing difficulty specifically for physically impaired users.

***Reduced Jerk and Precision from User***

The rapid motion nature of the keyboard allows users to move in more fluid and continuous motions, reducing the jerk exerted by the user as the move for the next key. Furthermore, the user can afford to be less accurate with their selection. Also, as more likely letters are nearer and larger, less precision is required.

***Fitt’s Law Benefit***

As the user need not stop above the letter they wish to choose, the time required to complete the movement is far greater as the realistic size of the target is considerably larger than the QWERTY keyboard counterpart.

***Increased Speed***

The culmination of all improvements, the Bubble Keyboard is inherently faster.

**An Overview of QWERTY Keyboard**

**An Overview of Bubble Keyboard**

**Related Work**

**The Design and Engineering**

This section explains in more detail the design and implementation of Bubble Keyboard. Due to the recursive and iterative nature of the Bubble Keyboard, an explanation of the structure is insightful to its successes.

***The Positioning of the Letters***

The letters in Bubble Keyboard are positioned in a manner that optimizes time required to find letters and minimalizing motion. The placement of letters in two rings allows users to select letters from either ring and reduces search time for letters as more likely letters are placed in the inner ring and all possible letters are in the outer ring. In the case of letters that occur in both rings, either one may be selected. Collected data shows that users prefer to select the letter from the inner ring [INSERT RESEARCH].

The outer ring is a static set of all letters of the alphabet in a clockwise alphabetical placement. This allows the user to always know where the next letter is irrespective of whether the letter is in the inner ring and if they can find it.

The inner ring is the collection of the eight most likely letters as predicted by the previous two words typed and the set of letters currently typed as part of the current word. The prediction of the most likely letters for the inner ring for the first letter of the word is based on trigram prediction[Citation]. Using the set of words most likely to follow the previous two words typed, we calculate the most likely initial letter for the word. If there are not eight distinct letters received from the words produced by the trigram algorithm (see below), we place the rest of the letters according to their probability of occurrence as the starting letter of any word as calculated by a tree dictionary (see below) also preloaded into the program.

The construction of the inner ring of letters when the letter being typed is not the preliminary letter is very similar to that of the construction of the preliminary letter. The previously constructed set of words by the trigram algorithm is used for the trigram algorithm for further letters. However, the algorithm only considers that words that start with the string currently typed. Similarly, the tree dictionary is traversed according to the string currently typed. The tree dictionary data with probability of word occurrence and that of the trigram algorithm will be used to determine the most probable letters with the trigram algorithm data given priority to that of the tree diagram letters.

***The Arrangement of the Inner Ring Letters***

The eight most likely letters are already organized by their likeliness of being the next letter. However, the likeliness of any letter is not obvious to the user; instead, they need to have an organization system that they easily recognize and can simply find.

The arrangement of the letters is pseudo-alphabetical. However, we have noticed that users tend to remember the general position of a letter but lack an intuitive sense of where is will show up next in a purely alphabetical arrangement as the position of the letter in the ring is based on the collection of the other seven most likely letters. Since a letter could be in virtually any position in the ring during the first iteration and then in any other position in the next iteration, we have made the placement of the letters “sticky”. The alphabetical arrangement is placed second in priority to the “sticky” nature of the letters. A letter is “sticky” if its position is in or near the position it was in the previous iteration. The order of which letters are placed is based on their likeliness of being the next letter.

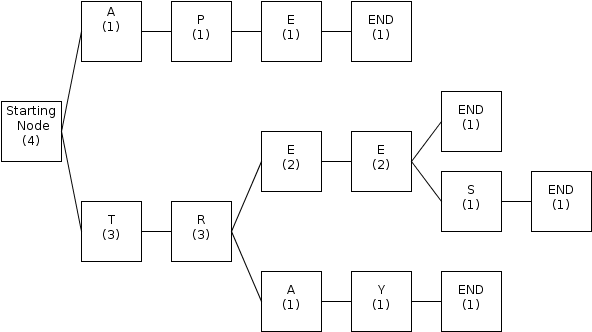
The resulting structure simplifies the process required by a user to find the next letter in the next iteration.

***The Trigram Algorithm and Dictionary***

A trigram is a prediction algorithm for the next word typed; it uses the previous two words to determine the next word typed. The trigram dictionary is a data storage system designed to reduce search time and computations; it organizes all trigram trios into an easily searchable system so that the list of possible trigram trios can be retrieved given the previous two words. All words in the trigram dictionary are given an integer index corresponding to its alphabetical location so that any word or its index can be found with a binary search if the other is known. An example of the Trigram Algorithm is shown in Figure 1. The far left column displays the data storage of all words in the trigram dictionary. Each word node contains a list of nodes that correspond to second words in a trigram trio given that the word node corresponds to the first word in the trigram trio.

These secondary nodes are simply indexes of possible secondary words in the trigram trio. Each of these nodes also contains a list of possible third words in the trigram trio. These are nodes contain indices and occurrence statistics

For example, the trigram trio “across the pond” would be represented by the indices 3, 16687, 8021, respectively. Searching for the first two indices would yield the list shown on the right. The occurrence statistics show that the phrase “across the bar” is more common than “across the pond” so the algorithm would place “bar” as a more likely next letter.



***The Tree Dictionary***

The Tree Dictionary is a data system that is optimized for predicting next letters given the initial character(s). The dictionary is displayed in Figure 2 with the words APE, TREE, TREES, and TRAY. As more letters are typed in, a pointer traverses down the tree. At each node, there is a count of how many words start with that string. Thus, we can effectively predict the next letter by using the occurrence statistics of the substring including the next letter.

Also, the tree dictionary is very simple to construct. The generation of the count at each node is done as the dictionary is being built. As a program reads through a set of words, it starts to construct the tree; when it reaches the end of word, it places an END node and triggers an iteration of the count of each node to the top of the tree.

**An Evaluation of Bubble Keyboard**

**Future Work**

**Conclusions**

**Acknowledgments**