WORKING NOTE AC98

Alan Creak
17 May 1996

FASTER COMMUNICATION THROUGH MULTIPLE KEY OPERATIONS

It is suggested that, in designing communications interfaces for people who are unable to use conventional devices in conventional ways, it is important to explore means of adapting existing interfaces to modes of use which are convenient for the people. A keyboard interface which can accept depressions of adjacent key pairs as well as single keys is described.

Technology used in the communications industry has advanced in two ways. The more obvious, and the more immediately spectacular, direction is towards new modes of communication; new communications media, new transmission techniques, and new hardware devices break the ground for more efficient, more convenient, or more attractive communications techniques. The second direction is less obvious, but is at least comparable in its effects; research in this area is directed at making the best use of available communications machinery. A prime example is the simple telephone line, which now carries information at rates which would have been unthinkable in earlier days.

The principle of making best use of available systems is important in rehabilitation computing for at least two reasons. First, many people are, for one reason or another, unable to use devices in the conventional ways, and they can be severely impeded if they are constrained to use only conventional methods. Second, though alternative interfaces are available, they cost money, while if a conventional device can be adapted a cheaper software solution might be possible.

In this article, I describe work on using a conventional computer keyboard in unconventional ways. The justification for this investigation is that, in effect, the keyboard is free : it usually comes with the computer whether you want it or not, and in any case you usually want it so that people accustomed to the conventional input modes can use them. If the keyboard can be adapted to alternative input techniques, the cost of an additional device might be avoided. It is probable that additional software will still be required, but its cost is likely to be comparable with that for the software needed to drive an alternative device, so the hardware cost for the alternative device might be avoided.

USING A CONVENTIONAL KEYBOARD.

Why is it that many people with various forms of motor disability find it difficult to cope with a typewriter keyboard ? The simple, obvious, and correct answer is that which also applies to corresponding difficulties with a knife and fork, opening doors while in a wheelchair, and many other everyday tasks : the rules for using the keyboard (or the knife and fork, or the door) have been written for people who are not constrained by the same difficulties. The rules for a typewriter say that you must press one key at once, or – for functions such as uppercase shift – two keys at once, and the keys to be pressed are precisely defined. Such rules are difficult to follow for people who can only press one key at a time, or have difficulty in pressing one key without touching adjacent keys.

A computer keyboard, though quite like a typewriter keyboard, is not the same. Because of this difference, the typewriter answer stated above, while still simple and obvious, could be wrong. The significant difference is that, while the keys of a typewriter are more or less directly connected to the printing mechanism, the same is not true of many computer keyboards. Their connection to the output, be it screen, printer, or other device, is indirect, passing through the computer's processor, and being therefore at least to some extent programmable. This possibility opens up a new approach to keyboard use : instead of having people trying to follow rules which are inconvenient and restrictive, why not rewrite the rules so that they fit what the people can do ?

One example of this type of development, the notion of "sticky keys", is well known. In this method, shifted characters can be formed by pressing the shift key and then the character key, so that it is no longer necessary to hold down the shift key while pressing the character key. This method has been widely implemented, and works much better than earlier mechanical methods which depended on some form of weighted lever which could be set to press on the shift key during the operation, and usually then had to be reset to return to the lowercase typeface. Despite its effectiveness, though, it does not seem to have been regarded as suggesting that the wider generalisation discussed here might be fruitful.

Given sufficient flexibility in the access by the computer to the keyboard signals, it would be possible in principle to recognise any combination of key depressions. The keyboard would then become a sensitive pad which could be programmed to recognise whatever combinations of keys were pressed by a hand in different positions and different poses. People who have difficulty in pressing a single key could thereby use the keyboard without the awkwardness of adopting an uncomfortable pose, or the inconvenience of a pointer attached to the hand. It would also be possible to distinguish the combinations pressed with the hand in different orientations, so in effect introducing an additional set of keys which could be used – for example – as alternative case keys or as programmable keys.

The software described here is intended to explore a rather different approach. It is intended for people who type with one finger, or other probe, and can use a keyboard quite accurately, but rather slowly. The intention is to provide what amounts to an enlarged keyboard by accepting simultaneous depressions of pairs of adjacent keys. (This does not exhaust the potential of the method; with a finger-sized probe, it is not difficult to press up to six combinations of up to four keys for most of the keys on a computer keyboard of ordinary design[2]. This study is restricted to combinations of two keys because it was easier to programme using the hardware available, and because the results will be sufficient to gauge the promise of the method.)

DESCRIPTION.

The software provides for both multiple key operation and sticky key operation, so that direct comparison is straightforward. The choice is made on starting the software, and can easily be changed during a session if required.

In sticky key mode, only the Shift key operates in sticky mode; the first key pressed after pressing shift is interpreted as its uppercase character. The Caps Lock key is interpreted as a shift lock, so shifts all keys while set. Otherwise, the keyboard behaves normally so far as text entry is concerned, though special functions associated with keys may not be preserved.

In multiple key mode, possible combinations of key depressions are any key alone, or a key together with the key to its right, or above and to the left, or above and to the right. (It is convenient to describe these pairs as horizontal, grave, and acute combinations, respectively.) The combinations are only defined where they make sense, so while all are defined for keys well within the conventional character keyboard, there are anomalies at the edges, and other keys (function keys, control keys, etc.) may not fit into the pattern. As different keyboard patterns have different geographies, the keyboard maps incorporated in the software may require rewriting if a different keyboard is used. Apart from these inevitable departures from generality, there is one exception to the rule : the Shift key continues to operate as normal, so that the keyboard remains usable by people accustomed to the conventional pattern.

For multiple key mode it is also necessary to determine a time interval within which key depressions are to be considered as simultaneous. This parameter will differ for different people, so it can be defined by choosing a number when the session starts. The number itself has no useful meaning, and its precise interpretation in terms of time will change from machine to machine, so it must be set by trial and error. If the number is too large, a character will not appear on the screen for a long time after pressing the key; if it is too small, double key depressions will be displayed separately instead of as a modified key.

One set of combinations is predetermined : the horizontal combination is interpreted as uppercase – so simultaneous depression of f and g is interpreted as F. These definitions only operate where they make sense (so uppercase F4, for example, though an acceptable combination, is not predefined as anything).

In either mode, keys without predefined meanings can be programmed to produce any desired sequence of keystrokes. Pressing the F2 key initiates a procedure in which the key to be defined is selected, then the required key sequence is entered. At the end of the session, the stored sequences can be saved, and reloaded for a later session if required.

The two modes have been kept compatible as far as possible, so that switching between them is easy. In both cases, Esc is used as a general escape from the special mode, so that repeatedly pressing Esc will eventually lead back to the ordinary programme. Uppercase Esc (Esc + Shift, or Esc + its right-hand neighbour) is a quicker way out of the programme. Key sequences stored in one mode can be used in the other, if they're accessible – again, a horizontal combination in multiple key mode is equivalent to a shifted character in sticky key mode, but the other adjacent sets have no such straightforward equivalence.

One potentially tricky question with any system in which non-standard keycodes are used is the encoding problem : how do you choose key combinations to represent defined sequences so that you can remember them when you want to use them ? Some clues are available, primarily the labels on the keys of the keyboard, but anything else must be remembered somehow. Systematic rules are helpful : a shift key always does the same thing, so is easy to remember; the multiple key convention of horizontal adjacency is similar applied consistently, and also easy to use. Apart from that, though, you have to make up your own clues. The programme can help to the extent that it can tell you what definitions you have made, and a list of such definitions is appended to the saved "keyset" file whenever a new one is made. The form of the list depends on the mode in use when the file is stored; shifted characters or horizontal combinations are used as appropriate.

DEFECTS.

This multiple-key programme is designed as a prototype purely for experimental use, and would not as it stands be suitable for general application. Here is a list of some obvious deficiencies.

Only two keys : The restriction to two keys is a consequence of the computer configuration used in developing the programme. The programme depends on being able to detect both the key depressions and the key releases as they occur, and their availability is determined by the system architecture. Even with conventional use, keys may be depressed almost simultaneously, and for a skilled typist this may not be a mistake, so it is usual to allow software to detect states in which more than one key is depressed. This number is commonly limited, though, and is called the *rollover* of the system. In the system used for development, the rollover value was 2 (an unusually low value), so only two simultaneous key depressions could easily be detected. The software is written so that it should be easy to extend to cope with more keys, though no advantage is taken of this feature in the current implementation.

It is not necessarily impossible to detect more events, as the limitation might not be determined by the keyboard hardware itself. (On the other hand, some ways of constructing keyboards can give misleading results if several keys are simultaneously depressed[3].) The disadvantage of any such development is that it is not necessarily easily portable. The programme as it stands adheres to MS-DOS conventions, and should run, more or less, on any MS-DOS system; to detect more keyboard events it would be necessary to go outside MS-DOS in order to catch the signals before they are handled by the input-output system, and there is no longer any guarantee of portability.

Timing : The time delay used for the definition of simultaneity is at present implemented as a busy-wait, with the processor executing a small loop of instructions for a number of times determined by the number entered to determine the speed when the multiple key mode is chosen. This is reasonably consistent in its timing, but unsatisfactory for several reasons.

First, the calibration varies from processor to processor. I tried to avoid this problem by using the standard MS-DOS time primitives, but they proved even less reliable for the rather small fractions of a second required; though the time is given in hundredths of a second, it is only reset a few times each second, so it is useless for measuring short times. It should be feasible to devise and implement an automatic calibration procedure, in which the system measured the number of loop cycles executed in a clock time long enough to be reasonably accurate – say, one second – and adjusted a scale factor accordingly; it is not implemented in the current version of the software because guessing the numerical value seems to work reasonably well in practice.

Second, for a Windows system, the constant busy waiting in effect disables the window operations, such as resizing or closing. This would be a nuisance for permanent use, but is tolerable for an experimental prototype.

In the longer run, both problems would be solved if the precise time could be read at each interrupt, when no busy wait would be necessary. (A non-Windows system would still need an idle loop, but that's a different matter.) The clock time is available in certain system locations, but again requires an excursion outside the realm of strict MS-DOS[4].

Keyboard functions : All key depressions (except for the Fn key, which is only perceptible in MS-DOS as modified values for other keys) are caught by the software, and not passed on to the system. While this is appropriate for most keys, keyboard functions such as PageUp, PageDown, Delete, and so on are useful, and it would probably be worth conserving them. Control keys are in the same category : there is no automatic detection of combinations such as Ctrl+h.

Isolation : Perhaps the major deficiency is that the interface software can at present be used only with the demonstration programme. For practical use, it should be installed as a device driver, when it would become available for any programme used with the system.

Keyboard geography : Adapting the system to a keyboard of different layout is not easy with the present construction, which depends on a number of tables which are constructed by hand. It would be preferable in the long run to provide a setting-up function, in which all acceptable key combinations were manually inserted, with predefined values as required. While this might be tedious, experience suggests that it might be significantly less tedious than trying to get the tables right without assistance.

Something of this nature will in any case be necessary if the system is extended to deal with any arbitrary combination of simultaneous key depressions.

Editing : At present, defined keystroke sequences can be edited to a limited extent (using backspace) while they are being defined, but cannot be recalled later and redefined. Better editing facilities will be provided in later versions, should there be any.

QUESTIONS TO BE ANSWERED.

The major question which I hope will be answered by work with this programme is whether or not the idea of multiple key operation using these simple patterns of adjacent pairs of keys will be useful in practice. To perform this evaluation, I need the help and cooperation of people who might benefit from the technique if it turns out to be useful, and from others with skills and experience in assessing and prescribing assistive devices. I hope to set up a well-structured experimental test of the system, in which the multiple-key approach will be compared with the sticky-key method.

In addition to this general question, some specific queries more connected with the design and eventual application of the method have emerged from the work done so far. A few are obviously raised by the defects listed above; here is a further selection :

Better compatibility between sticky key and multiple key systems : While I have tried to keep the two methods equivalent as far as possible (which is one reason for predefining the horizontal combinations as uppercase characters), there are no sticky-key equivalents to the multiple-key grave and acute combinations. In my own experience with the system, this has emerged as a significant defect of the sticky-key approach. While simple text entry is not affected, the deficiency reduces the number of easily accessible definable keys very significantly, so that combinations defined easily in multiple-key mode become inaccessible when the mode is switched. Should the sticky-key method be further extended to include equivalents for these other combinations ? An obvious method would be to use the Ctrl and Alt keys instead of the Shift key for the other two, or Ctrl+Shift+key and Alt+Shift+key if the control key functions are to be preserved.

Other two-key combinations : I have assumed that adjacent keys are the easiest pairs to press simultaneously, but that may be to oversimplify the position. People with hands constrained to a rigid configuration by spasticity may find other combinations accessible. To what degree would it be useful to provide means of defining arbitrary sets of two-key combinations for use with a system of this sort ?

Encoding strategies : What sorts of encoding strategy can be used to assist recall of defined key sequences ? Given that a list of defined codes can be printed, is it worth worrying about encoding ? Methods such as Minspeak[1] work by using more elaborate key labels, thereby providing a wider range of clues which can be used to associate with definitions; would more varied key-caps be useful ? Alternatively (or, for that matter, additionally), would some form of computer-resident help be useful ? (I have guessed that it wouldn't, because you could waste more time searching through the help system than it would take to enter the text in full, but that's a guess.)

Extensions and additions : What should be done better ? What should be done differently ? What should be done that isn't done at all ? What should not be done that is done ? Questions such as these are not easy for me to answer, because I have little experience in using the methods in practice. Guidance from expert users is always welcome.

REFERENCES.

1 : B. Baker : "Minspeak", *Byte* **7**, 186 (September, 1982).

2 : G.A. Creak : *Multiple keying for faster communication*, unpublished Working Note AC91, (September, 1994).

3 : J. Fulcher : *An introduction to microcomputer systems* (Addison-Wesley, 1989), page 104.

4 : See, for example, K. Heidenstrom : *FAQ / Application notes: Timing on the PC family under DOS* (SimTel FTP archive, file PCTIM003.ZIP, 1995).