Far Eastern Cultures and the Console User Interface

Antoine Bossard

Graduate School of Science, Kanagawa University, Yokohama, Japan
Email: abossard@kanagawa-u.ac.jp

***Abstract*—Even as of 2024, the support of Far Eastern writing systems by computers remains uneven, to say the least. For example, vertical right-to-left user interfaces, as it could naturally be expected in countries like Japan and China, are extremely rare. In fact, it is the opposite that has been happening since the broad adoption of computer systems: people of these cultures have got accustomed to western, left-to-right horizontal interfaces. In previous research, we have started to technically investigate the current situation in the case of Graphical User Interfaces (GUI); the results obtained showed that although noticeable improvements have been made, the support provided by current technologies (e.g., operating system features, browser features) in the realisation of Far Eastern interfaces is still nowhere near that of western ones. In this paper, we propose and evaluate an extension to the Console User Interface (CUI) to improve its support of Far Eastern writing systems. Concretely, the vertical right-to-left application scenario is considered for the CUI. The measured results are positive in that they match those obtained in the case of the conventional, left-to-right horizontal layout. Precisely, our proposal is slightly faster when it comes to the text scrolling speed (between 3% and 9% in our experiments).**

***Keywords*—cultural heritage, writing system, Chinese, Japanese, Korean (CJK), vertical, Right-to-Left (RTL), kanji, hanzi, hanja**

I. Introduction

We start by reviewing several cultural patterns for the users of Far Eastern cultures which are at the centre of our research. Far Eastern cultures involve different countries, and thus different writing systems. For instance, mainland China relies on simplified Chinese while the writing system of Taiwan and Hong-Kong is based on traditional characters. Furthermore, the Japanese culture combines Chinese characters (*kanji*) with the characters of local syllabaries (*hiragana* and *katakana*) and Korea even mingles Chinese characters (*hanja*) with its own script, *hangul*.

Several common patterns can be inferred from these transnational cultural facts: first, specific sets of characters are required; second, those sets are comparatively large; third, several writing directions can be necessary: vertical (top-to-bottom) right-to-left writing is traditional, but not exclusive: horizontal (top-to-bottom) left-to-right writing is common [1].

The user interface of computer systems has seen major improvements since the advent of commercial, general-purpose computers. First developed by and for western cultures, adjusting computers to Far Eastern needs has been challenging: although hardware limitations have now significantly eased, especially memory, deep software issues remain, starting with character encoding [2, 3]. Considering user applications, and although desktop publishing now supports advanced vertical typography and typesetting [4, 5], user interfaces and the underlying development technologies (e.g., Application Programming Interfaces) still have much room for improvement [6]. Besides, the importance of software localization has been recently recalled in [7].

In this paper, we describe an extension to the Console User Interface (CUI) to improve its support of Far Eastern writing systems. A CUI, also known as command-line interface, is one special sort of Text User Interface (TUI) that is typically used with terminal (emulator) software. It is ubiquitous in computing, albeit for power users, and IT system administration; console software is thus actively developed by major operating system manufacturers [8]. Technical details are investigated and the performance of the proposed system is then both qualitatively and quantitatively evaluated. The obtained results are positive: the achieved performance of the proposed system matches that of conventional approaches. Precisely, the proposed system succeeds in realising a vertical right-to-left console user interface and the quantitative results show no performance degradation in scrolling speed when comparing to conventional left-to-right horizontal terminal layouts.

The proposal is an extension, rather than a replacement, to the console user interface, that enables to rely on existing environments, which is critical for usability, notably user-friendliness. In addition, although the system has been designed and experimented on a Japanese system, we show that it applies to other Far Eastern scripts as well, like simplified and traditional Chinese.

The rest of this paper is organised as follows. Related works are audited in Section II. The proposed system is described in Section III and empirically evaluated in Section IV. The experimental results are discussed in Section V. Finally, this paper is concluded in Section VI.

Manuscript received January 20, 2025; revised March 8, 2025; accepted April 24, 2025; published September 5, 2025.

II. Related Works

Shen *et al.* [9] have investigated culture-centred design, notably applied to computer user interfaces. It is interesting to note that in this related research work, a computer interface specifically designed for Chinese users was developed. The authors introduced the idea of a garden metaphor to replace the traditional computer desktop interface. Icons depicting items typical of the Chinese garden culture were used. This previous work focuses on graphical user interfaces, whereas our proposal considers another user interface, the console user interface.

Still considering the graphical user interface scenario, Gil and Collazos proposed to consider this time user emotions and how they relate with interface issues. For instance, they tried to connect user interfaces and music to reach user emotions [10]. This is yet another cultural pattern on which could rely user interface designers. These two authors continued their investigations of cultural patterns for graphical user interface realisation and they tried to identify design patterns more specifically targeted at web interfaces, such as translation, colour, and directly related to our research, (writing) directions [11].

In general, cultural issues of human-computer interaction design are reviewed and discussed in [12].

More directly related to the cultural patterns identified for our proposal, we have noted that other related works include right-to-left *horizontal* support, and in general bidirectional text (i.e., bidi). Bidirectional text support is uneven amongst terminals; a standardization effort has been made though [13]. In addition, this standard draft discusses bidirectional text support in the case of PuTTY, Konsole, Mlterm, Terminal.app, Bicon and VTE. In our proposal, we go even further by considering the right-to-left *vertical* scenario.

Finally, there have been attempts at proposing a web-based console user interface [14]. Relying on the browser for rendering, such proposals can be considered steps towards localization of the console user interface; they relate to our previous work [6]. We can cite the anyterm, ajaxterm and xterm.js projects, to only give a few. In this paper, we consider a native console user interface, not its emulation inside, say, a browser.

III. Methodology

The proposed system has been implemented in standard C without relying on system-specific functions.

A. Input and Memory Issues

First and foremost, the terminal has to be setup for Chinese, Japanese, Korean (CJK) support. Next, the current locale of the program has to be set for Japanese and Chinese characters support with the character-handling functions of the standard C library (e.g. `fputws`). For example, we have called `setlocale (LC_ALL, "ja_JP.UTF-8")` to this end. Then, we rely on the standard wide character data type `wchar_t` for multi-byte character handling.

Because the amount of input data remains unknown, we acquire them on the fly in blocks: the input data are stored as is inside memory pages which are dynamically allocated upon needs, that is, until all the input data has been stored. (Precisely, input data are acquired until an End-of-File (EOF) condition is satisfied.) Memory pages are implemented as a linked list of character strings: see Fig. 1. The page size does not really matter: it is the typical trade-off between memory and time. On the one hand, the larger the pages, the higher the (internal) fragmentation (only at the last page in our case though), and on the other hand, the smaller the pages, the more the memory allocation requests.

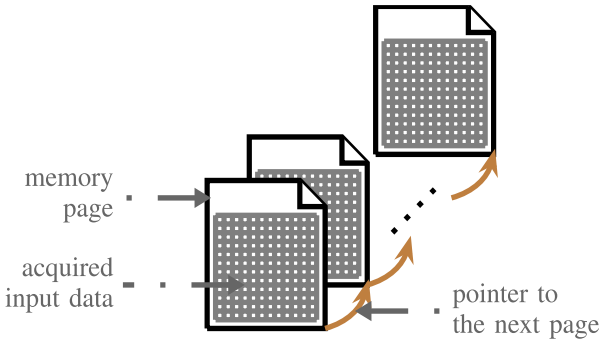

Fig. 1. Dynamic memory allocation with pagination for input acquisition. Memory pages are implemented as a linked list.

Data are read from the standard input stream (`stdin`), which enables typical use cases involving piping. Just as the conventional system utility `more` acquires its data from `stdin` and provides user-friendly interfacing, the proposed system `vmore` typesets the input data according to the vertical right-to-left layout. Hence, amongst others, the following sample use cases are supported:

- `cat file.txt | ./vmore` (piping),
- `echo sample text | ./vmore` (piping), and
- `./vmore` for direct input, typically terminated with Ctrl-D to emit EOF.

B. Transposition

Whereas in the first step, input data are acquired and stored without modification, the second step, transposition, starts transformation from the horizontal left-to-right layout to the vertical right-to-left layout.

Concretely, characters of the input data are copied into a new buffer, called the transposition buffer. The viewport is smaller than or equal to the transposition buffer as detailed next.

The height of the viewport is said to be fixed in the sense that it always matches the height of character columns (i.e., the number of rows of the transposition buffer), which is constant (it does not depend on the viewport content). There is no vertical scrolling (just as there is no horizontal scrolling for typical console applications).

Unlike the viewport height, the viewport width is smaller than or equal to the number of columns of the transposition buffer. The first character column displayed by the viewport is an adjustable parameter which is used for horizontal scrolling. An illustration is given in Fig. 2.

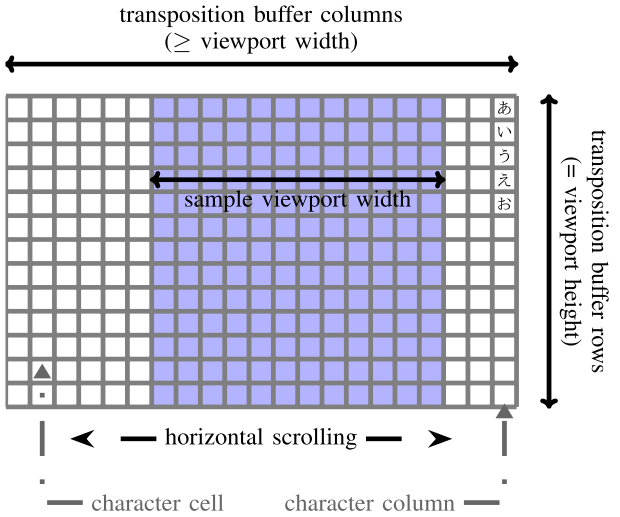

Fig. 2. 1-dimensional array that is the target of the horizontal-vertical transposition operation. Some or all of its columns act as viewport for rendering. The blue area is a sample viewport. Sample characters are shown in the first (i.e., rightmost) column.

The height of character columns (i.e., the number of rows) as well as the number of columns of the viewport (i.e. the viewport width) can be set by passing parameters when launching `vmore`, which is how the viewport can be adjusted to the terminal dimensions as explained in Section III-D.

In order to precisely calculate the amount of memory required for transposition, the number of character columns is first computed; here, the handling of new line characters is key. Precisely, columns are counted as follows: if the input data include at least one character (i.e., other than EOF), set the column counter to 1, and to 0 otherwise. Then, for each memory page, iterate the stored characters according to the following set of rules:

- a carriage return character (`\r`) is in any case ignored;
- a new line character (`\n`) that is at the top of the column and not following another new line character is discarded;
- a new line character (`\n`) that is not at the top of the column or at least the second in a sequence of new line characters triggers incrementing the column counter;
- for any other character, increment the column counter when reaching the last row of the column, unless the current character is the final one.

(It should be noted that, unrelated to `vmore`, text processing software often automatically appends a new line at the end of the file, at least on Linux, and that it may not always be rendered when opening the file in, say, a text editor. This can be easily checked with the system utility `od`.)

The total size of the memory allocated for transposition is then directly obtained by multiplying the calculated number of columns times the number of rows, plus one for the terminal null character. After memory allocation, the viewport is filled with full-width spaces (Unicode code point `0x3000`); cf. Section III-C.

C. Character Conversions

First issue: CJK characters are typically full-width characters: considering a monospace font, the width of one such character is twice that of, say, a Latin character. The former are called full-width characters and the latter half-width characters. In order to retain proper alignment of the character columns, any half-width character is converted to its full-width variant. This is typically the case of ASCII characters, digits and symbols. Thanks to the definition of the Unicode code points for half-width and full-width characters, this is rather easily done as shown in Listing 1.

Listing 1. Half-width to full-width character conversion

```
wchar_t to_fullwidth(const wchar_t wc) {
  if(wc == 0x20) // half-width space
    return 0x3000; // full-width space
  else if(0x21 <= wc && wc <= 0x7E)
    return wc - 0x20 + 0xFF00;
  else // outside of the convertible range
    return wc;
}
```

Second issue: transposing glyphs as detailed in Section III-B produces satisfactory results albeit with some remaining refinements expected. It is well known that vertical typography has its own set of challenges for text processing software and font foundries. Some characters effectively need to be rotated, or at least adjusted, in accordance with vertical typesetting rules. For example, transposing parentheses '(', ')' as is into the viewport is not satisfactory for vertical typesetting. Instead, these parentheses need to be rotated by 90 degrees clockwise. Other symbols, like the comma, need to be shifted from left to right. Lunde speaks of "Vertical character variants" [1]. Such sample conversions are illustrated in Table I. The support of such features of vertical typography depends on the character sets supported by the terminal application.

TABLE I. SAMPLE CHARACTER CONVERSIONS FOR VERTICAL TYPESETTING: PARENTHESES, ANGLE BRACKETS, ELLIPSIS

Character Type	Parentheses		Angle brackets		Ellipsis
Horizontal typesetting	(	)	〈	〉	…
Vertical typesetting	﹵	ﹶ	ﹿ	﹀	⋮

In practice, we have defined for `vmore` a conversion function that is mostly based on the two Unicode ranges "Vertical Forms" (0xFE10–0xFE1F) and "CJK Compatibility Forms" (0xFE30–0xFE4F), the latter being used especially for rotated brackets (including parentheses). Other than these two character ranges, we have used the range "CJK Unified Ideographs" (4E00–9FFF) in order to convert the glyph U+30FC (*kana* prolonged sound mark) to the character U+4E28 (vertical stroke). It should be noted that it is critical that the glyph resulting from such a conversion be full-width too.

The rendering by `vmore` of two lines of text including characters to which apply such adjustments is illustrated in Fig. 3. Brackets, the ellipsis and the long vowel Japanese symbol are rotated and the comma and period are moved

from the bottom left-hand corner to the top right-hand corner.

Fig. 3. Sample vertical typography adjustments: brackets, the ellipsis and the long vowel Japanese symbol are rotated and the comma and period are moved from the bottom left-hand corner to the top right-hand corner.

D. The User Interface

First, so as to maximise system portability, the user interface (i.e., CUI) has been realised solely with standard functions. Precisely, we emphasise the solutions to the following three main issues.

- User input to control the program is realised with functions of the standard C library such as `getwchar`. Non-standard functions such as `_kbhit` of Microsoft's `conio.h` are completely avoided.
- Elements of the user interface, also known as chrome, like toolbars are created with ANSI escape codes [15].
- Dimensions of the terminal are obtained with a shell script to avoid relying on non-standard console I/O functions to this end. The `tput` Linux command and the `mode CON` Microsoft Windows command suffice therefor.

Because input data can be acquired with the pipe system feature of terminals, the standard input (`stdin`) can be connected to a file rather than the default terminal input feature which reads data directly input by the user, say with his keyboard. So, in the case `stdin` is connected to a file, we had to reconnect it to the console input. To this end, although not breaking portability and standard compliance, we had to conditionally define the constant value for the first parameter of the `freopen` function, that is the file (path) to which `stdin` is to be reconnected after acquiring the input data. On Microsoft Windows, it is set to the character string `CONIN$` whereas it is set on Linux to the character string `/dev/tty`, as required by these operating systems. This adjustment is seamlessly realised with a preprocessor conditional directive.

IV. Evaluation

The experiments described hereinafter have been conducted on mainstream terminal applications for different operating systems.

A. System Usability

First and foremost, regarding the usability of the proposal, we have considered the three major cultural patterns identified in introduction to evaluate our proposal. First, the specific sets of characters used for the various Far Eastern cultures were successfully supported by the system: we have made experiments with both simplified Chinese text, traditional Chinese text, Japanese text and even Latin text since, for example, Japanese documents can include Latin words. We also confirmed that the large number of the involved characters did not pose any problem. Last but not least, we have confirmed that the proposed system satisfies the vertical right-to-left typesetting requirements of Far Eastern writing systems.

In practice and more concretely, this has been confirmed on a desktop computer running the Linux operating system (Debian 12, Japanese version) with various scenarios (Latin text, Japanese text, text piped from a text file, text input on the fly, scrolling, and so on). The case a sample Japanese text file is piped into `vmore` via the shell script to retrieve the console dimensions as explained is illustrated in Fig. 4.

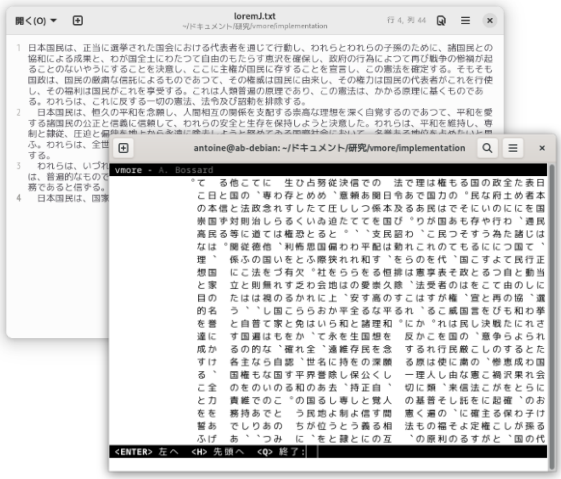

Fig. 4. Background: sample Japanese text file opened in a conventional text editor (left-to-right horizontal layout). Foreground: result in a terminal when this text file is piped into the proposed system (right-to-left vertical layout).

Importantly, Japanese is not the only writing system supported: Fig. 5 illustrates the case of Chinese, both simplified and traditional, with screenshots of outputs of the first stanza of the Chinese classic text Tao Te Ching. And writing systems can be mixed, for instance Japanese text including Chinese quotations.

Fig. 5. The proposed system does not apply only to Japanese: sample screenshots are given here for traditional Chinese (top) and simplified Chinese (bottom).

B. User Experience

Next, regarding user experience (UX), we have conducted an experiment to measure the display performance of the proposal, more precisely the performance of its scrolling feature, which is a notorious issue of both text processing software and user experience [16]. On a desktop computer equipped with an Intel Core i5-12400 CPU and 16 GB RAM and running a 64-bit Japanese Debian 12 operating system, we have compared the time required to fully scroll the data output by `vmore` the proposed system, `more` and `man`, two well-known system utilities. We have relied on the `time` command and recorded its real time value. The console used was the default GNOME Terminal, whose default dimensions are 24 rows and 80 columns. `vmore` was run via the shell script launcher to adjust to the terminal dimensions.

In this experiment, the scrolling performance of `vmore`, `more` and `man` were compared with the following commands:

Command 1 `time man ls | ./launcher.sh`
Command 2 `time man ls | more`
Command 3 `time man ls`

These time measurements were semi-automatic as scrolling was done by keeping the keyboard's Enter key pressed until reaching the end of the data to display, after which the programs were manually terminated by inputting the quit command Q (in the case of `more`, Q was input as soon as the user saw the (END) signal; for fairness, the `-e` switch was not used). Because running on a Japanese operating system, the `man` pages used in this experiment were Japanese versions. A flowchart illustrating this testing procedure is shown in Fig. 6 and the obtained results are detailed in Table II.

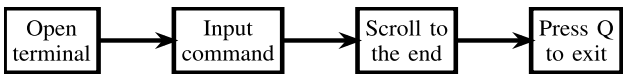

Fig. 6. Flow of the user experience (UX) experimentation. (Time measurement itself is included in the command.)

TABLE II. SCROLLING PERFORMANCE EVALUATION: `vmore` IS COMPARED TO THE CONVENTIONAL SYSTEM UTILITIES `more` AND `man` INSIDE DEBIAN'S GNOME TERMINAL

Properties and measurements	Command 1 (`vmore`)	Command 2 (`more`)	Command 3 (`man`)
number of columns	485	n/a	n/a
number of rows	n/a	255	255
full scroll time (s)	14.37	7.80	7.88
columns per second	33.75	n/a	n/a
rows per second	n/a	32.70	32.34

Because involving, even minor, manual operations, the same experiment in the same conditions was conducted several times: the obtained times remained in line with those of Table II.

We have conducted the same experiment inside a Microsoft Windows console (`cmd`, not PowerShell, run inside Windows Terminal) with `vmore` run with SSH—the SSH server is the same machine as in the previous experiment (Debian 12). The computer running the `cmd` console is equipped with an AMD Ryzen 9 5900HX CPU and 16 GB RAM and running a 64-bit Japanese Windows 11 operating system. The default dimensions of this terminal are 30 rows and 120 columns. `vmore` was once again run via the shell script launcher to adjust to the terminal dimensions. A screenshot illustrating Command 1 is given in appendix. The obtained results are detailed in Table III.

TABLE III. SCROLLING PERFORMANCE EVALUATION: `vmore` IS COMPARED TO THE CONVENTIONAL SYSTEM UTILITIES `more` AND `man` INSIDE A MICROSOFT WINDOWS CONSOLE (`cmd`) WITH SSH

Properties and measurements	Command 1 (`vmore`)	Command 2 (`more`)	Command 3 (`man`)
number of columns	400	n/a	n/a
number of rows	n/a	233	233
full scroll time (s)	12.23	7.76	7.55
columns per second	32.70	n/a	n/a
rows per second	n/a	30.02	30.87

For the sake of readability, the results of these two experiments are summarised in Fig. 7.

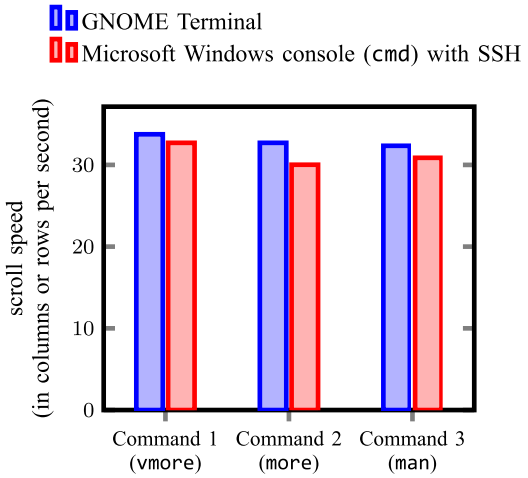

Fig. 7. A summary of the experimental results: scrolling speed inside GNOME Terminal and the Microsoft Windows console (`cmd`) with SSH.

V. RESULTS DISCUSSION

We start by considering the user experience experimental evaluation (cf. Section IV-B). First, we discuss the results obtained in the case of GNOME Terminal. From the data of Table II, although `vmore` has a slightly faster scrolling (in columns per second) than `more` and `man` (in rows per second), there is no significant difference in scrolling performance between the three programs. Precisely, a difference of 3.21% was obtained when comparing `vmore` to `more` and a difference of 4.36% when comparing `vmore` to `man`. Since these measurements were obtained after conducting a semi-automatic experiment (it is recalled that all three programs were terminated manually with the Q command as explained), this difference could amount to measurement error or deviation. Of course, due to the vertical interface and

console dimensions (24 rows, 80 columns), columns are shorter than rows (even with full-width characters: 80/2 = 40 columns but only 24 rows), which could be another explanation of the scrolling performance difference between `vmore` and `more`, `man`.

Second, we discuss the results obtained in the case of the Microsoft Windows console `cmd` with SSH. From the data of Table III, `vmore` has once again a faster scrolling (in columns per second) than `more` and `man` (in rows per second). Precisely, a difference of 8.92% was obtained when comparing `vmore` to `more` and a difference of 5.92% when comparing `vmore` to `man`. Once again, due to the vertical interface and console dimensions (30 rows, 120 columns), columns are shorter than rows (even with full-width characters: 120/2 = 60 columns but only 30 rows), which could be another explanation of the scrolling performance difference between `vmore` and `more`, `man`.

We then consider the usability experimental evaluation (cf. Section IV-A). On the other hand, these results also show the applicability of the vertical right-to-left layout: short columns are more efficient than long rows in this use case as rows are not always full.

As explained, since using full-width characters, only half of the console character columns are available: only 80/2 = 40 (resp. 120/2 = 60) columns (at most) are displayed at once in GNOME Terminal (resp. Windows console `cmd`), that is, one screen. (The number of displayed rows is not impacted.) It is unclear how this could affect performance and performance comparison though. In any case, the number of character columns is clearly established, hence the validity of our measurements.

In addition, whereas `vmore` handles itself (horizontal) scrolling, it might be the case—it obviously depends on implementations—that (vertical) scrolling in `man` or `more` is directly handled by the terminal host program (i.e. not by `man` or `more`), which would likely be faster in this case. In any case, the obtained results are a positive indicator of the high performance of `vmore`.

VI. Conclusion

Far Eastern writing systems induce non-trivial human-computer interaction design issues, such as right-to-left and vertical writing and typesetting. In continuation of previous and related work focused on Graphical User Interfaces (GUI), we have considered in this paper such issues in the case of the Console User Interface (CUI). Precisely, we have described an extension to current terminal software so as to support, even partially, right-to-left vertical layouts. Then, we have qualitatively and quantitatively shown the practicability of the proposed CUI extension for Far Eastern writing systems such as Japanese and Chinese. To this end, both the usability and user experience issues were considered.

Regarding future work, one potential improvement could be enhancing the user interface, although this would likely require using non-standard features to better control the console environment. A specialised library such as ncurses could be used to this end. In addition, although it may not apply directly to `vmore` since in most cases its input data are acquired with piping, vertical right-to-left input is an important issue for the CUI in general. Its support would however require relying on non-standard console features, with thus a library such as ncurses key to portability. Finally, although `vmore` has been developed in accordance with the C and ANSI standards as explained, cross-compiling issues, precisely issues specific to the Microsoft Windows console, require additional work, beginning with code page and encoding issues.

Appendix

A sample output of `vmore` when used inside the Microsoft Windows console (`cmd`) with SSH on a Japanese Windows 11 operating system is given in Fig. A1. This figure illustrates the case of Command 1: `time man ls | ./launcher.sh` as detailed in Section IV.

Fig. A1. Output in the case of Command 1 (`time man ls | ./launcher.sh`) inside the Microsoft Windows console (`cmd`) with SSH.

CONFLICT OF INTEREST

The author declares no conflict of interest.

ACKNOWLEDGEMENT

The author is sincerely grateful towards the reviewers for their insightful comments and suggestions which helped improved this article.

REFERENCES

[1] K. Lunde, *CJKV Information Processing*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2009.

[2] A. Bossard and K. Kaneko, "Unrestricted character encoding for Japanese," in *Databases and Information Systems X*, IOS Press, 2019, vol. 315, pp. 161–175. https://doi.org/10.3233/978-1-61499-941-6-161

[3] A. Bossard, "Proposal and evaluation of a Chinese character hash function based on strokes for fingerprinting," *International Journal of Computers and Their Applications*, vol. 29, no. 2, pp. 59–65, 2022.

[4] H. Okumura, "pTeX and Japanese typesetting," *The Asian Journal of TeX*, vol. 2, no. 1, pp. 43–51, Apr. 2008.

[5] H. Mukai, *An Introduction to Japanese Typography Based on Structure and Algorithm*, Tokyo, Japan: Seibundo-shinkosha, 2018. (in Japanese)

[6] A. Bossard, "On bridging the gap between Far Eastern cultures and the user interface," in *Trends and Applications in Information Systems and Technologies*, Springer International Publishing, 2021, pp. 415–424. https://doi.org/10.1007/978-3-030-72657-7_40

[7] A. Tray, "Why bother localizing information technology products?" *Communications of the ACM*, vol. 67, no. 2, pp. 6–7, Feb. 2024. https://doi.org/10.1145/3638537

[8] P. Bhojwani. (August 2024). Windows Terminal preview v1.22.2362.0. *Microsoft's official GitHub repository.* [Online]. Available: https://github.com/microsoft/terminal/releases

[9] S.-T. Shen, M. Woolley, and S. Prior, "Towards culture-centred design," *Interacting with Computers*, vol. 18, no. 4, pp. 820–852, 2006. https://doi.org/10.1016/j.intcom.2005.11.014

[10] R. Gil and C. A. Collazos, "Integrating emotions and knowledge in aesthetics designs using cultural profiles," in *Proc. the Second International Conference on Usability and Internationalization (UI-HCII)*, 2007, pp. 344–353. https://doi.org/10.1007/978-3-540-73289-1_40

[11] C. A. Collazos and R. Gil, "Using cross-cultural features in web design patterns," in *Proc. the Eighth International Conference on Information Technology: New Generations (ITNG)*, 2011, pp. 514–519. https://doi.org/10.1109/ITNG.2011.95

[12] R. Heimgärtner, "Intercultural user interface design—culture-centered HCI design—cross-cultural user interface design: Different terminology or different approaches?" in *Proc. Design, User Experience, and Usability. Health, Learning, Playing, Cultural, and Cross-Cultural User Experience (DUXU)*, 2013, pp. 62–71. https://doi.org/10.1007/978-3-642-39241-2_8

[13] E. Koblinger. (January 2019). A draft proposal for handling RTL and BiDi text in terminal emulators. [Online]. Available: https://terminal-wg.pages.freedesktop.org/bidi/

[14] H. Yanagisawa and K. Kondou, "Interactive interface for web-based programming environment," in *Proc. the 27th International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, 2013, pp. 168–173.

[15] Standard ECMA-48: Additional control functions for character-imaging I/O devices, 2nd ed. European Computer Manufacturers Association (ECMA), 114 rue du Rhône, 1204 Geneva, Switzerland, Aug. 1979.

[16] P. Quinn, A. Cockburn, G. Casiez, N. Roussel, and C. Gutwin, "Exposing and understanding scrolling transfer functions," in *Proc. the 25th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2012, pp. 341–350. https://doi.org/10.1145/2380116.2380161