SwiftPad: Exploring WYSIWYG TEX Editing on Electronic Paper

Elliott Wen
The University of Auckland
jwen929@aucklanduni.ac.nz

Gerald Weber
The University of Auckland
g.weber@aucklanduni.ac.nz

***Abstract*—Electronic paper (i.e., e-paper) is a display technology that aims to imitate and substitute the conventional paper. Previous studies of e-paper mainly focus on evaluating or making practical use of its readability. However, there is little research to explore the potential of e-paper on input-oriented applications. In this paper, we introduce a document composition system named SwiftPad for e-paper. Specifically, SwiftPad renovates the famous TEX typesetting system, enabling users to compose high typographic quality documents on e-paper in a WYSIWYG (what you see is what you get), offline-first, and collaborative fashion. Building such a system on resource-constrained e-paper with low screen refresh rate creates unique challenges. In this paper, we identity these challenges and provides workable solutions. We also provide a preliminary evaluation of the new system.**

I. Introduction

E-paper is a display technology that mimics the appearance of ordinary ink on printed paper. Owning to its excellent viewing experience (e.g., high contrast, wide viewing angle and non-glowing panels), e-paper has been widely adopted for various scenarios demanding high readability such as book readers and post signs.

More recently, many manufacturers are producing 13.3-inch e-paper. They advertise it as a digital device for writing and reading that feels like standard A4 paper. This design philosophy, however in reality, is not entirely implemented. Many devices indeed deliver decent reading experience to users, nevertheless few of them ever attach importance to the writing experience. Their built-in documentation composition applications such as sketching or note-taking are mostly rudimentary, in a sense that they lack necessary typesetting capabilities to generate documents with aesthetics and formality. Therefore, they are seldom applicable for scenarios such as education or scientific work.

To bridge this gap, we introduce SwiftPad, a novel document composition system for e-paper devices. SwiftPad renovates the acclaimed TEX (we use TEX and LATEX interchangeably) typesetting system, aiming to meet the following user experience goals:

High Typographic Quality: The system delivers an excellent typographic quality in the generated documents. They can be used in scenarios that demand formality such as scientific publications.

WYSIWYG: The system provides a WYSIWYG user interface similar to a word processor, allowing users to concentrate on document composition rather than tedious TEX typesetting procedures.

Offline-first and Collaborative: Users are able to edit and compile documents even if there is no Internet connection. Users are also allowed to edit documents collaboratively with colleagues.

Implementing such a system entails multi-fold challenges. The first challenge is that WYSIWYG editing has a strict latency requirement: users expect to see resulting documents in the order of milliseconds after they make some edits. However, existing decade-old TEX engines work in a slow batch processing fashion and may take the order of seconds to compile a long document in e-paper devices with low-spec CPUs. The second challenge stems from the fact that off-the-shelf e-paper devices lack a standardized operating system (OS). The diversity of the OSs renders a portable implementation of SwiftPad rather difficult. The last challenge derives from two notable drawbacks of e-paper screens: low refresh rate and ghost effects. They make e-paper not directly suitable for displaying our WYSIWYG editor, whose contents change frequently.

In this paper, we present practical solutions to cope with the above challenges. Firstly, we conduct a ground-up rewrite of the TEX engine and incorporate a compilation checkpointing technique to meet the latency requirement of WYSIWYG editing. Secondly, to make our implementation portable, we compile SwiftPad into the WebAssembly binary format, which can be directly executed in every major browser in a near native speed. Lastly, SwiftPad exploits the concurrent update feature of e-paper screen controllers to combat the low refresh rate and ghost effects. We consolidated the above techniques and implemented a prototype of SwiftPad on off-the-shelf 13.3-inch e-paper devices, based on which we conducted a preliminary user study involving six participants to evaluate the usability aspects of the system. The participants reacted positively to the innovative WYSIWYG editor for e-paper.

II. System Overview

Fig.1 demonstrates the general hardware setup of SwiftPad, consisting a 13.3-inch e-paper device and an optional wireless bluetooth keyboard. We argue that keyboards are so far the most reliable input instrument for SwiftPad because of their popularity and users' familiarity. Nevertheless, we will investigate the possibility of other input technologies such as speech recognition and handwriting input in the near future.

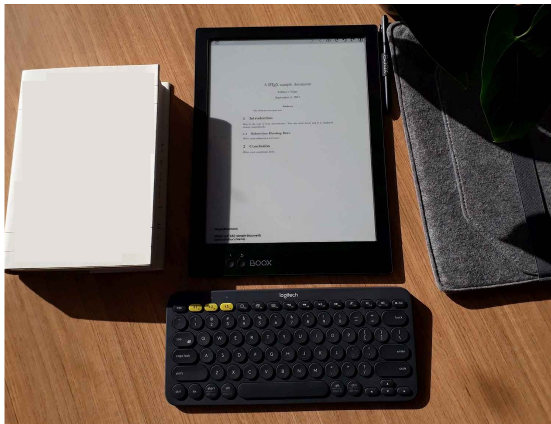


Fig. 1. General Hardware Setup

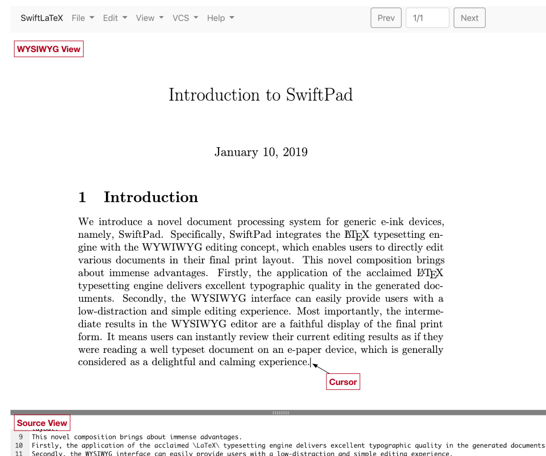


Fig. 2. User Interface of SwiftPad.

SwiftPad is equipped with a simple-yet-powerful user interface as shown in Fig. 2. Note that the screenshot is directly captured from a e-paper device's graphics memory in pursuit for better presentation purposes. It can be seen that SwiftPad offers two different editors. The first one is the source editor, which allows advanced users to directly manipulate TeX source code in a classical ASCII editor. The second one is the WYSIWYG editor, which allows the user to directly edit a document in its print form, but with effect on the source. More specifically, the WYSIWYG editor possesses the following features:

1) At editing quiescence (i.e., a moment when the editor has processed all previous edits of the user), the editor shows the print layout of the document, i.e., acts as a faithful print viewer.
2) At editing quiescence, the user can position the cursor anywhere in the document with the mouse/touchscreen, arrow keys or a combination thereof.
3) The user can perform edits at the cursor position by simply typing the keys or backspace and get instance visual feedback.
4) Meanwhile, the user's editing operations will also be applied at the corresponding position of the TeX source code.

SwiftPad also provides a file browser to facilitate users' file management (e.g., uploading or creating source files). The file browser incorporates an underlying storage protocol named *RemoteStorage*, which provides offline data access, automatic cloud synchronization and collaborative editing functionalities.

III. Implementing a Performant TeX Engine

Implementing SwiftPad on e-paper devices entails unique challenges. One main challenging issue is to achieve high responsiveness of our WYSIWYG editor, in order words, allowing users to instantly see what the end result will look like while a document is being edited. It is not trivial because compiling documents like a 10-page scientific paper may cost the order of seconds by conventional engines. The long turn around time may be attributed to the following two reasons. Firstly, most existing engines were invented decades ago and may have applied certain obsolete computing technologies. For instance, the PdfTeX engine, which has most user base, was implemented in a legacy and undocumented programming language *WEB* and adapted 7-bit character encoding. To keep using PdfTeX in a modern computer, software compatibility layers have to be enabled at the cost of performance. Secondly, TeX is a batching system, which implies that every time a compilation is initiated, the TeX engine must process the input files from the very beginning till the end. Such behavior is undesirable considering that, in most cases, users only append or modify characters located at the end of the input file, while leaving the preceding contents unchanged. It can be seen that recompiling the unchanged contents leads to a considerable amount of repeated computation.

To mitigate these drawbacks, we decided to conduct a group-up rewrite of a TeX engine. Our engine adopts up-to-date computing technologies such as high performance programming language C++, Unicode encoding and multi-threading. This mitigates the use of software compatibility layers from old engines and thus boosts the overall performance. Nevertheless, the main contribution of our engine lies in the introduction of checkpointing, which allows us to save and reload the engine's states (i.e., snapshots). This functionality can be used to accelerate compilation by skipping repeated computation. Specifically, the engine can create checkpoints periodically (e.g., after outputting each page) and mark down the corresponding input file positions in the first compilation. When an user modifies somewhere in the input file, the engine then can determine the closest checkpoint and start the next compilation from there. We can further reduce compilation time by instructing the engine to stop right after generating the page the user is currently viewing or editing. For instance, when the user is working on page 5, the engine can start from the checkpoint on page 4, only re-typeset the page 5 and ignore

page 6 onwards. This ensures the compilation time remains nearly constant regardless of the page number of a document.

One core step for the checkpointing functionality is to save/reload application memory, specifically, static memory (i.e., global variables) and dynamic memory (i.e., mostly heap). Static memory has a pre-determined addresses and sizes during the compile time. Thus, checkpointing the static content is merely a memory copying process. In contrast, the heap memory is allocated by OSs and has constantly varying memory addresses and sizes during the runtime. Saving or recovering the heap memory layout requires a series of system calls, which are non-portable and error-prone. To solve this issue, our engine does not use the OS heap allocator. Instead, it exploits another dynamic memory management scheme, namely, memory pool. Memory pool pre-requests a fixed-size memory chunk at a known memory address and later allocates it to programs in user space. This deterministic memory layout enables us to treat the memory pool almost the same as the static memory, thus simplifying our implementation.

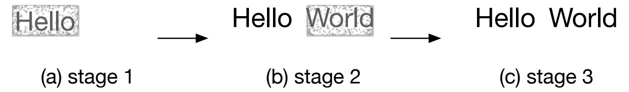
IV. Broswer-ify SwiftPad

Another hurdle for us to overcome is that off-the-shelf e-paper devices lack a standardized OS and programming interfaces. In other words, we may have to adapt and optimize SwiftPad for different devices, which is labor-intensive. SwiftPad addresses this issue by exploiting an observation that almost every e-paper device is now bundled with a web browser, which provides a standard runtime environment for web applications. Therefore, if we can implement SwiftPad as a web application, we then can execute it in every e-paper device in an installation-free fashion.

To browser-ify SwiftPad, the first step is to trans-compile our TEX engine from C++ to browser languages, specifically, WebAssembly or Javascript. WebAssembly is preferable, as it is a modern binary instruction set designed to run in a main-stream browser in a near native speed. Javascript has a relatively worse performance (approximately 2 times slower), but can be served as a fallback for old Javascript-only browsers. The trans-compilation can be achieved using Clang compiler along with LLVM WebAssembly backend.

After getting a runnable TEX engine in a browser, we can now proceed to implement the WYSIWYG editor. One core functionality is to display the typesetting result, in other words, PDF files. However, we notice that many browsers do not provide intrinsic PDF format support. For these browsers, we have to rely on external Javascript libraries (e.g., PDF.js) to parse PDF, which can cost the order of seconds. To address this performance issue, our engine is enhanced to emit HTML rather than PDF files, considering that browsers have always been highly optimized to render HTML pages even for low-specs devices.

One crucial step for HTML emission is to express vector graphics elements in original PDF with efficient HTML markup. For instance, to display embedded PDF fonts, we could take advantage of a CSS rule named ‘font-face’. To support PDF geometric commands, we translate them into Scalable Vector Graphics markup, which then can be directly rendered by a browser. Regarding the text elements, they are positioned with absolute coordinates in a PDF document. To preserve the locations, one simple method is to convert them into CSS absolute position rules. This, however, would render the resulting page bulky because each text element is now associated with a unique CSS rule. Instead, we convert the absolute coordinates to relative position rules. Specifically, we first attempt to merge PDF text segments to text lines based on their geometric metrics. Afterwards, we measure the space width between words in each line and turn them into CSS rules. Finally, these rules can be used to construct HTML spacer elements (i.e., empty span elements in a certain width) to help position each text element from left to right in each line. The advantage of this approach is that the number of generated CSS rules tends to be tiny since there are usually limited number of spacers with different widths in a PDF page.



(a) stage 1 (b) stage 2 (c) stage 3

Fig. 3. Concurrent Update and Repainting

V. Optimization for E-paper Display

An e-paper screen is internally controlled by a specifically-designed circuit: Electrophoretic Display Controller (EPDC). It is responsible for driving corresponding electrical signals to the e-paper panel upon receiving draw commands from the CPU. Each draw command contains not only image data and position, but also a parameter called update mode, whose possible values include 2, 4, 8, or 16 graylevels. The 16-graylevel update mode delivers the best display quality (i.e., highest contrast and little ghost effect) but consumes the longest timespan to finish (approximately 1 second). In contrast, the 2-graylevel update mode enables fast animation of screen contents (approximately 150 ms) but may generate poor contrast texts and significant ghost effects.

We can notice that there exists a trade-off between display quality and latency, which are equally important to our editor. In this paper, we propose a method to automatically strike balance between them. The core idea is that we process a draw command twice; we first paint a region use 2-graylevel mode in pursuit of screen responsiveness, afterwards we repaint the same region using 16-graylevel mode to improve display quality. This process is parallelizable thanks to the concurrent update feature of the EPDC, which allows multiple draw requests to be processed at the same moment if their regions do not overlap with each other. This condition frequently holds true for our editor application. A demonstrating example has been depicted in Fig. 3; at stage one, the user types a word ‘hello’, which is drawn in 2 graylevel mode. At stage two, the user types another word ‘world’, this word is still drawn

in 2 graylevel mode while the previous word 'hello' can be repainted using 16 graylevel mode at the same time since the painting regions of the two words do not collide. At the last stage, the user stops typing, so the word 'world' is repainted and all the words are now in 16 graylevel mode.

VI. Preliminary Experiments

We conduct a preliminary performance evaluation of our system on a popular off-the-shelf 13.3-inch e-paper device *BOOX MAX*. It is equipped with a single-core 1GHz ARM CPU and 512 MB RAM. We first use a predefined set of TEX code snippets to generate sample documents with 100 pages. We compile each sample document for 100 times. Before each compilation, we insert/modify some texts randomly to mimic users' editing behaviors. We then report the average compilation time. The experiments show that the average compilation time for our new engine is 126 ms, significantly outperforming the conventional engine PdfTeX, which takes 7812 ms. Our engine ensures the responsiveness of our WYSIWYG editor.

We also carried out a simple Discount Usability Test, which involves a small number of participants with a focus on qualitative studies on prototype design [1]. Existing studies [2] have suggested that a discount usability test with only 5 participants can offer reliable evaluation results and may identify up to 85% of the usability problems. In our work, we invited 6 participants from academia to evaluate our system. Specifically, we instructed them to compose an essay with at least 500 words describing their current research focus. Afterwards, we required them to complete a System Usability Scale (SUS) questionnaire, which is a commonly-used reliable tool for perceived usability evaluation even with a small sample size. The mean SUS score of SwiftPad is 71 (in a range spanning from 0 to 100). It exceeds the threshold score of 68 which indicates a decent level of usability. In addition, we also computed the mean values for the usability sub-scale and learnability sub-scale, which are 70 and 75 respectively.

VII. Related Work

E-paper has been generally considered to be a promising display technology in the field of reading. A small number of research [3], [4] has been done to evaluate the readability of e-paper. The results suggest that the reading experience of e-paper is highly similar to printed paper and e-paper triggers significantly less visual fatigue compared with growing LCD screens. E-paper have been successful applied in some useful application scenarios. For instances, Chiu et al. [5] evaluates the potential of using the e-paper to encourage users to cultivate healthy reading behaviors. Blankenbach et al. [6] proposed a smart medicine package, aiming to address the issue that today's packaging for pharmaceutics provides no information about individual medicine intake. Flexkit [7] and PaperTab [8] explored the possibility of using E-paper as accessory displays for document presentation.

However, most existing research works solely focus on evaluating or making use of readability of e-paper. There is little research on applying e-paper in scenarios other than reading. To bridge this gap, we have identified that a TEX WYSIWYG editor would be a useful input-oriented extension for e-paper. Though several attempts have been made to implement TEX based WYSIWYG editors like LyX [9] and SwiftLaTeX [10], they were originally designed for PCs and have high system requirements that e-paper cannot meet. Moreover, these systems do not take special properties of e-paper screens (i.e., low refresh rate and ghost effects) into consideration, potentially leading to a poor user experience. In contrast, SwiftPad presented in this work is specially optimized for resource-constrained e-paper devices.

VIII. Conclusion and Future Work

In this paper, we present SwiftPad, a novel document composition system for e-paper. It integrates the TEX typesetting engine with the WYWIWYG editing concept, enabling users to compose high typographic quality documents on e-paper. Nevertheless, SwiftPad still bears several limitations that need further improvement. Currently, SwiftPad adopts keyboards as the main input technology. Other alternative input options, for instance, voice input and handwriting input need to be explored. Besides that, the WYSIWYG editor needs to be enhanced for structural edits (e.g., inserting sections, tables and images). We are also planing to carry out a comprehensive usability study, which involves more participants from different domains.

References

[1] J. Nielsen, R. L. Mack *et al.*, *Usability inspection methods*. Wiley New York, 1994, vol. 1.

[2] J. Nielsen, "Discount usability: 20 years," *Jakob Nielsen's Alertbox Available at http://www. useit. com/alertbox/discount-usability. html [Accessed 23 January 2012]*, 2009.

[3] E. Siegenthaler, P. Wurtz, P. Bergamin, and R. Groner, "Comparing reading processes on e-ink displays and print," *Displays*, vol. 32, no. 5, pp. 268–273, 2011.

[4] S. Benedetto, V. Drai-Zerbib, M. Pedrotti, G. Tissier, and T. Baccino, "E-readers and visual fatigue," *PloS one*, vol. 8, no. 12, p. e83676, 2013.

[5] P.-S. Chiu, Y.-N. Su, Y.-M. Huang, Y.-H. Pu, P.-Y. Cheng, I.-C. Chao, and Y.-M. Huang, "Interactive electronic book for authentic learning," in *Authentic Learning Through Advances in Technologies*. Springer, 2018, pp. 45–60.

[6] K. Blankenbach, P. Duchemin, B. Rist, D. Bogner, and M. Krause, "22-2: Smart pharmaceutical packaging with e-paper display for improved patient compliance," in *SID Symposium Digest of Technical Papers*, vol. 49, no. 1. Wiley Online Library, 2018, pp. 271–274.

[7] D. Holman, J. Burstyn, R. Brotman, A. Younkin, and R. Vertegaal, "Flexkit: a rapid prototyping platform for flexible displays," in *Proceedings of the adjunct publication of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 17–18.

[8] A. P. Tarun, P. Wang, A. Girouard, P. Strohmeier, D. Reilly, and R. Vertegaal, "Papertab: an electronic paper computer with multiple large flexible electrophoretic displays," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2013, pp. 3131–3134.

[9] D. Kastrup, "Revisiting wysiwyg paradigms for authoring latex," *COMMUNICATIONS OF THE TEX USERS GROUP TUGBOAT EDITOR BARBARA BEETON PROCEEDINGS EDITORS KAJA CHRISTIANSEN*, vol. 23, no. 1, p. 57, 2002.

[10] G. W. Elliott Wen, "Swiftlatex: Exploring the true wyswiyg editing for publication," *DocEng*, 2018.