TDOM: コンパイル待ちのない LaTeX 執筆
——文書を生きた木として保つ検証つき増分組版——

TDOM: LaTeX Authoring Without the Compile Wait — Verified Incremental Typesetting over a Live Document Tree

川口晃世　幸川卓実

株式会社 Fermion

contact@fermion.company

2026 年 8 月 18 日

概要

LaTeX による執筆では、わずかな修正であっても文書全体の再コンパイルを経なければ結果を確認できず、その待ち時間は文書の成長とともに増大して執筆の反復を妨げる。本稿は、かつて Web ブラウザが静的ページの再読込を DOM(Document Object Model) に基づく増分再描画へ置き換えたのと同じ移行を、無改造の LuaTeX の上で実現する組版ランタイム TDOM(TeX Document Object Model) を提案する。TeX のマクロ言語は実行中に字句規則を変更しうるため、Web におけるような構文木を静的に定めることはできない。そこで TDOM は、解析した結果ではなく実行そのものを保持する。文書を段落単位のブロックに分け、各境界まで組み終えた LuaTeX のプロセスを、`fork(2)`——実行中のプロセスを複製する Unix の機能——によって停止させたまま常駐させておく。編集が届くと、その直前で止まっているプロセスを再開し、変更されたブロックだけを読ませる。カウンタもマクロ定義もフォントの状態も、そのプロセスがすでに保持している。影響の及ぶ範囲は有界の検証で確定し、改ページ・フロート・脚注・相互参照・目次を含む大域的なレイアウトも、全文の再コンパイルを待たずに増分的に収束する。表示が通常の LuaLaTeX 出力と一致することは、PDF コンテンツストリームに基づく 0.1 bp 精度のレイアウト照合、および無作為な編集列の後で増分状態が新規構築状態と一致することの検査を含む 4 系統の手続きで機械的に確認され、一致を保証できない構文は行・ブロック・文書の各粒度で実出力による表示へ退避する。評価では、4 ページから 30 ページの文書に対して定常の打鍵応答が p50 で高々 18.2 ms、p95 で 100 ms を下回り、文書規模にほとんど依存しないことを確認した。

Abstract. Editing a LaTeX document still requires recompiling the entire file before any change can be inspected, and the delay grows with the document. This paper presents TDOM (TeX Document Object Model), a typesetting runtime that carries over to unmodified LuaTeX the transition the Web once made from page reloads to incremental re-rendering over the DOM. Because the TeX macro language can alter its own lexical rules during execution, no syntax tree of the source can be defined statically; TDOM therefore keeps the execution itself rather than a parse of it. The document is split into blocks, and a LuaTeX process that has typeset everything up to each block boundary is kept resident, halted, by `fork(2)`. An edit resumes the process sitting just before it and feeds it only the changed block; counters, macro definitions and font state are already held by that process. The range affected is settled by a bounded check, while page breaking, floats, footnotes, cross-references and the table of contents converge incrementally rather than awaiting a full recompilation. Agreement between the display and genuine LuaLaTeX output is checked mechanically by four independent procedures, including layout comparison at 0.1 bp precision against PDF content streams and an equivalence test between incrementally maintained and freshly built engine states under randomized edits; constructs whose agreement cannot be established fall back to display taken from the real output, at line, block or document granularity. In our evaluation, steady-state keystroke response stays at or below 18.2 ms (p50) and below 100 ms (p95) on documents of 4 to 30 pages, largely independent of document size.

1 はじめに

LaTeX を用いた執筆では、原稿に手を入れるたびに文書全体をコンパイルし直し、生成された PDF を開いて該当箇所を確かめるという手順が繰り返される。コンパイルの所要時間は文書の規模とともに延び、パッケージを多用する 100 ページ規模の書籍では 1 回あたり 10〜30 秒に達することが報告されている [1]。この待ち時間は一回ごとに見れば小さいが、修正と確認のあいだに毎回挟まることで執筆の反復そのものを重くし、書き手は結果の確認を後回しにするか、確認のたびに思考を中断するかの選択を迫られる。組版品質の点で LaTeX に代わるものがない分野ほど、この編集と表示の断絶は避けがたい負担であり続けてきた。

同種の断絶が解消された先例として、Web の変遷が参考になる。1990 年代の Web ページは操作のたびに全体をサーバから読み直すものであったが、DOM(Document Object Model)[12] の標準化を境に、文書はブラウザ内に保持されるオブジェクトの木となり、スクリプトによる木の変更に対してレイアウトエンジンが影響範囲だけを再計算して再描画する構成へと移行した。ページの再読込が木の変更に置き換えられたことは単なる高速化にとどまらず、のちの対話的な Web アプリケーションが成立する基盤となった。

TeX についても、遅さの原因が組版アルゴリズムそのものにないことは実測から知られている。Knuth–Plass の行分割 [3] は 1 段落あたり 1 ms 程度で完了する [1]。時間を要しているのはむしろ、入力を読み、全体を処理し、出力を書き、内部状態を残さず破棄するという使い捨ての

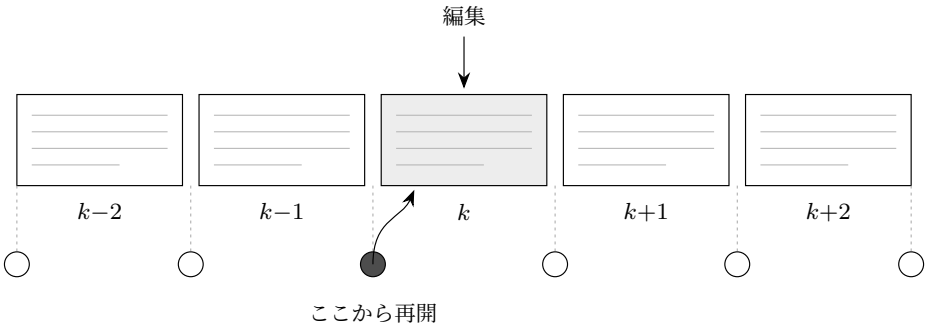

図1　文書は段落単位のブロックに分かれ、各境界には、そこまで組み終えた LuaTeX のプロセスが停止したまま常駐している (図の丸)。ブロック k の直前にある丸は、ブロック $1..k-1$ を組み終えたプロセスである。ブロック k が編集されるとこの丸を再開し、ブロック k だけを読ませる。プロセスはカウンタもマクロ定義もフォントも保持しているので、結果は文書を通しで組んだ場合と一致する。

プロセスモデルの方であって、このモデルを保ったままでは、キャッシュや並列化を導入しても所要時間は文書規模への比例を脱しない。文書を実行中のランタイムの内部に保持し、編集を保持された状態への変更として扱うことができれば、組版品質に触れることなく、編集応答だけを対話的な水準へ引き下げられるはずである。

本稿で提案する TDOM(TeX Document Object Model) は、この構成を LuaTeX の上で実現するものである。ただし、Web の DOM を TEX へそのまま持ち込むことはできない。TEX では、入力の各文字が担う役割——エスケープ文字か、群の開始か、数式の切り替えか、ただの文字か——が catcode と呼ばれる値で決まり、しかもこの値は実行中に書き換えられる。`\makeatletter` 以降では `@` が英字として読まれ、verbatim 環境では`\` さえ普通の文字になる。同じ文字列が、そこへ到達するまでに何が実行されたかによって別の意味を持つのだから、ソースの静的な解析によって構文木を得ることは一般にはできない [4]。TDOM はこの制約を、木のノードにパース結果ではなく実行状態を対応させることで回避する (図1)。文書は段落単位のブロックに分けられ、ブロック k まで組版し終えた LuaTeX のプロセスが、停止したまま常駐する。これがノードの実体である。マクロもカウンタもフォントも catcode も、そのプロセスの中に生きたまま残っているから、状態を列挙して書き出し、読み戻す仕組みは要らない。この複製を担うのが `fork(2)`、実行中のプロセスをまるごと複製する Unix の仕組みである。複製された二つは、どちらかが書き換えるまで同じメモリを共有するので、大きなプロセスを複製しても、その場の費用はほとんどかからない。

編集された箇所だけを組み直す試みは、近年独立に現れている。Lode[1] は、行分割が段落の外側に依存しないことに着目し、編集された段落ひとつだけを常駐 LuaTeX に組み直させれば約 1 ms で済むことを実測で示した。本研究の前提を裏づける結果であるが、この方式では段落を文書の文脈から切り離して組むため、扱える範囲に制約が残る。節番号や相互参照のように直

前までの実行に依存する内容は速い経路に乗らず、改ページ・フロート・脚注・ページ番号の更新は、周期的に走る全文コンパイルを待つことになる。加えて、速い表示が本物の LaTeX 出力と一致しているかを確かめる仕組みを持たない。本研究が扱うのはこの三点、すなわち文脈の保存、大域的なレイアウトの増分的な収束、そして出力の機械的な検証である。

本稿の貢献は次の四点である。第一に、素の LuaTeX に 71 行の C コードを読み込ませるだけで、停止したプロセスの列として文書の実行状態を維持する構成を示す (§4)。エンジンの改造を必要とした TeXpresso[2] と異なり、既存のパッケージ資産にも将来の TeX Live 更新にもそのまま追従する。第二に、影響がどこまで及ぶかを自ら確かめながら止まる増分再組版を示す (§4)。編集のたびに走る同期処理は決まった個数のブロックを確かめた時点で打ち切られ、残りは手が止まってから収束するので、相互参照・目次・改ページ・フロート・脚注が全文コンパイルを待たずに更新される。第三に、表示の正しさそのものを検査の対象とする構成を示す (§5)。一致すると判定できた行だけを文字として描き、判定できない行は本物の出力から作った画像に置き換えたうえで、表示と LaTeX 実出力の一致を 4 系統の手続きで機械的に検査する。第四に、実測により、定常の打鍵応答が文書規模にほとんど依存せず、すべての規模と編集位置で 100 ms を下回ることを示す (§6)。

2 背景

2.1 使い捨てのプロセスモデル

通常の LaTeX 実行は、一回かぎりのバッチ処理である。`lualatex` はソースを先頭から末尾まで読んで組版し、PDF を出力したのち、内部状態を保持せずに終了する。所要時間の内訳は、プロセスの起動とフォーマット——マクロ定義を読み込み済みの状態で固めたもの——の読み込みに約 0.3 秒、プリアンブルの処理に 0.1〜5 秒であり、本文の組版はその後に始まる。前者は編集内容に依存しない固定費であり、内部状態が実行ごとに破棄される以上、次の実行でも同額が再び生じる。

ここに多重パスの問題が重なる。相互参照や目次のページ番号は、組んでみるまで確定しない。そこで LaTeX は 1 回目の実行で参照情報を補助ファイル (aux) に書き出し、2 回目の実行でそれを読み戻して正しい番号を印字する。正しい出力を得るには最低 2 回、目次がページ数を動かす場合には 3 回の全文コンパイルが要る。修正のたびにこの手順を繰り返す点で、今日の LaTeX 執筆は、Web がかつて続けていたページ再読込と同じ形のループの中にある。

2.2 関連研究

主な先行アプローチを表1に整理する。TeXpresso[2] は XeTeX エンジンに手を入れ、プロセスの複製を保存点として編集位置以降を再実行する。実行状態をプロセスとして残す着想は本

表1　リアルタイム LaTeX への先行アプローチと TDOM

	エンジン改造	組み直す単位	文脈の保存	大域レイアウトの即時更新	出力の機械検証
latexmk 等の自動再実行	不要	文書全体	—	(全文コンパイル)	なし
TeXpresso[2]	必要	編集位置以降	あり	再実行で収束	なし
SwiftLaTeX/BusyTeX	不要	文書全体	—	(全文コンパイル)	なし
Typst[6]	(新エンジン)	式単位・$O(n)$	あり	毎回全体を再収束	なし
Lode[1]	不要	段落・$O(1)$	なし	全文コンパイル待ち	なし
TDOM(本稿)	不要	ブロック (有界)	あり	あり	あり (4 系統)

研究と共通するが、エンジン本体の改造を要する点が異なる。SwiftLaTeX[8] と BusyTeX[9] は TeX を WebAssembly に移してブラウザ内で走らせるもので、配布や起動の形は変わるが、コンパイルそのものは毎回全文である。Typst[6, 7] は増分計算を言語の設計に組み込んだ新しい組版系であるが、LaTeX と互換性がなく、増分コンパイルの時間も文書規模に対して線形に伸びると報告されている [1]。Lode[1] は前節のとおり無改造の LuaTeX で段落単位の組み直しを実現したが、文脈の保存、大域的なレイアウトの増分更新、出力の検証は扱っていない。

3　設計

TDOM の名称は Web の DOM に由来する。DOM が HTML を解析した結果を保持するのに対して、TDOM が保持するのは実行状態であり、この違いは前節で述べた TeX の性質からくる。それを除けば編集の流れは同型で、木の一部が変わると、変わったノードとその影響が及ぶ範囲だけが計算し直され、表示には差分だけが送られる。対応を表2に示す。

表2　Web の DOM と TDOM の対応

	Web ブラウザ	TDOM
文書の内部表現	DOM ツリー (解析結果)	ブロックの列＋停止した TeX プロセス (実行状態)
変更の単位	ノード・属性	ブロック (段落の境界。環境は分割しない)
影響範囲の追跡	スタイル無効化・変更フラグ	組版結果のハッシュ＋組み終えた値の組＋依存表
再レイアウト	影響範囲だけの再計算	有界の同期処理＋手が止まってからの収束
再描画	差分の描き直し	ページ単位の描画命令の差分
最終的な出力	(描画エンジン自身)	普通の LuaLaTeX 出力 (正本層)

設計の方針は三つある。

第一は文脈を保つことである。ブロックを文書から切り離して単体で組むことはしない。ブロック k を組むのは、ブロック $k{-}1$ までを処理し終えたプロセスから複製された子であり、組む場所も本物の主垂直リスト——組み上がった行がページに割られる前に積まれていく縦の列——の上である。この作り方をすれば、カウンタも、マクロ定義も、フォントの状態も、直前の行の

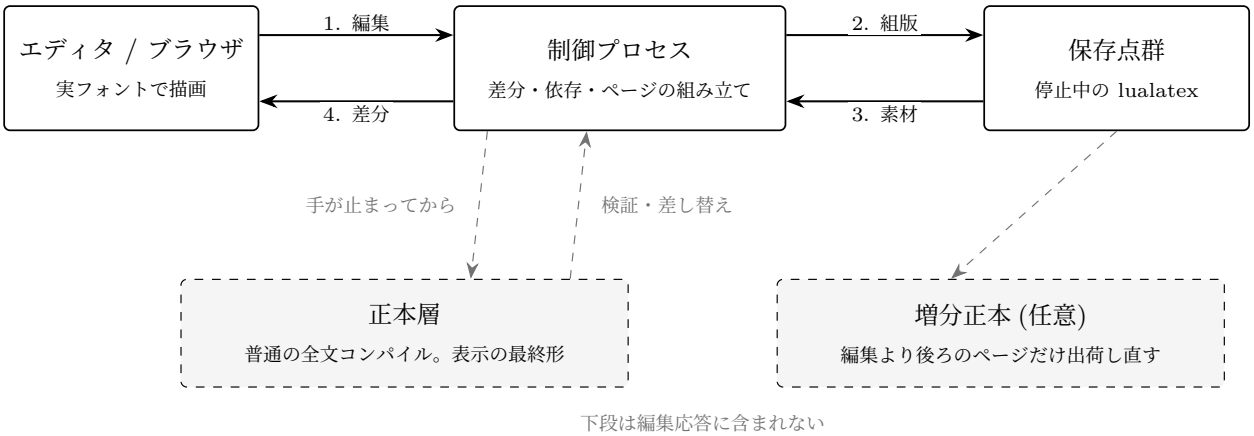

図2　全体の流れ。打鍵のたびに走るのは (1) から (4) の経路だけであり、この往復が十数ミリ秒である。(1) 編集が届き、(2) 手前の保存点を複製してそのブロックだけを組ませ、(3) 組版素材を受け取り、(4) 変わったページの描画命令だけを返す。下段の二つは編集応答に含まれない。手が止まってから走り、画面を本物の LaTeX 出力に置き換えるとともに、検証の基準を与える。

深さも、段落間の空きの合成規則も、通しで組んだ場合と同じになる。近似したのではなく、実際に通しで組んできたプロセスの続きだからである。

第二は表示を検証することである。速い表示は、本物の LaTeX 出力と一致すると判定できた範囲に限って使う。判定できない行は本物の出力から作った画像に置き換える。さらに、あとから照合して食い違いが見つかった領域は自動的に格下げする。表示が一時的に古いことは許すが、本物と違う表示は許さない、という規律である。

第三は段階的に退けることである。40 年分のパッケージ資産のすべてを速い経路で扱うことはできない。扱えないものは、行 (数式の行を画像にする)、ブロック (その塊だけ別に組む)、文書 (本物の出力だけを見せる) の 3 段階で退ける。どの段階に退いても、書き手はそのまま編集を続けられる。

4　アーキテクチャ

4.1　保存点の構成

ブロック k までを処理し終えた LuaTeX のプロセスを、以後「保存点 k」と呼ぶ。編集が届くと、編集位置の手前にある最寄りの保存点を複製する。複製された子は、編集後のブロックを読んで組む。組み終えた子が、そのまま次の保存点となる。複製元は停止したまま残るので、次の編集でも同じ位置から利用できる。

この構成に必要なのは、プロセスを複製する手段を LuaTeX に与えることだけである。常

駐する親プロセスは、通常の `lualatex` として起動する。組版の途中で Lua を実行させる命令`\directlua` から、71 行の C コードを読み込む。このコードが提供するのは `fork`、`waitpid`、`_exit` など数個のシステムコールにすぎない。エンジンの実行ファイルにも LaTeX カーネルにも変更は加えない。ただし共有ライブラリの読み込みは通常禁じられているため、親プロセスの起動にのみ`--shell-escape` を要する。

保存点の個数には上限を設ける。ひとつが実際に動作するプロセスであり、数百メガバイトを占めるためである。上限に収まるよう、保存点は文書全体へ等間隔に配置する (既定は最大 64 個、後述の計測では 8 個)。ただし編集中の位置の前後だけは、間隔と無関係に保持する。間隔を粗くした分は、再開時に手前のブロックを組み直して補う。その費用はブロック 1 個あたり約 3 ms である。

4.2 組版素材の取り出し

各ブロックを組んだ結果は、PDF を経由せずに取り出す。取り出し元は TeX 内部の主垂直リストそのものである。このあいだ、ページの組み立ては止めておく。具体的には、1 ページに入る高さを表す`\vsize` を最大値に設定してページが決して満ちないようにし、脚注は `\holdinginserts=1` によって本文の流れに保持したままにする。そのうえで、組み上がった行のボックス、伸縮量と次数を含むグルー (伸び縮みする空き)、ペナルティ (そこで改ページすることの罰点)、脚注の本体、フロートの位置印を、そのまま受け取る。グルーが実際に何ポイントに伸びたかは LuaTeX の `node.effective_glue` が答えるので、得られる文字の座標は、出荷される PDF のそれと一致する。

この取り出しが、行分割の後、かつ改ページの前である点が設計の要である。改ページの判断材料であるグルーとペナルティが手つかずで残っているからこそ、次節の処理が TeX と同じ基準で改ページをやり直せる。

4.3 ページの組み立て

取り出した素材に対して、JavaScript 側の処理が、TeX のページ分割と、LaTeX の出力ルーチン——ページが満ちたときに呼ばれ、本文と脚注とフロートを組み付けるマクロ群——に相当する仕事を、編集のたびに実行する。改ページ位置を選ぶ評価値の計算は、Knuth の実装 (tex.web §108) をそのまま移したものである。フロートの配置、脚注の組み付け、ページ上端の空き、柱とノンブル、`\cleardoublepage` に伴う偶奇の調整と白ページの挿入までを扱う。配置を左右するパラメータはすべて実行中の TeX から実測値として受け取っており、この層が独自の寸法を持ち込むことはない。

ページの境界ごとに途中状態を控えているので、組み直しは編集の近くのページに限られる。Lode[1] が改ページの更新を周期的な全文コンパイルに委ねたのに対して、TDOM では改ページ

ジも打鍵と同じ応答の中で更新される。

4.4 影響範囲の見きわめ

各ブロックは、組版結果のハッシュと、組み終えた時点の値の組を持つ。後者に含まれるのは、全カウンタの値、直前の行の深さ、直前で改ページを禁じていたかどうか、末尾に残っている空きの量である。編集のたびに、編集されたブロックに続くブロックを組み直し、結果と値の組が前回と完全に一致することを確かめる。一致すれば、そこから先には影響しない。

この確認を無制限には続けない。レイアウトが結びついている場合は 8 個、値の波及だけの場合は 4 個で打ち切る。打ち切った時点の判定は三つに分かれる。一致を確認できた場合は、そこで終わりである。カウンタなどの値だけが動いている場合は、番号の追従を後回しにする。マクロ定義の変更のように下流の意味が変わりうる場合は、下流の組み直しを後回しにする。後回しにした処理は編集が 300 ms 途絶えてから走り、次の打鍵が来れば直ちに中断して、途中から再開する。

相互参照は補助ファイルを介さない。LaTeX の相互参照は `\r@` ラベル名という名前のマクロとして実現されているから、これを実行中に直接定義してしまえば、ファイルへの書き出しと読み戻しは要らなくなる。どのブロックがどのラベルを参照しているかの表を持ち、ラベルの値が動いたときには、それを参照しているブロックだけを組み直す。目次も同様に、ページ番号が動くたびに目次を使うブロックを組み直し、最大 3 回の繰り返しで落ち着かせる。

4.5 表示の判定と実出力への退避

ブラウザで文字を描くにあたっては、TeX が実際に使ったフォントファイルをそのまま配信し、ブラウザ側の字詰めと合字処理を止めたうえで、行内の位置をフォントの字送りから再現する。それでも一致を保証できない行が残る。数式を含む行と、TikZ のように生の PDF 命令を含む行である。数式は、既定のフォントがブラウザの読めない旧来の形式であるうえ、大型記号の部品はフォントの私用領域に置かれて文字コードでは指し示せない。そのため数式の判定はフォント名に頼らず、組版中に TeX が数式へ出入りした境界の記録によって行い、数式に由来する行は、使われたフォントを問わず対象とする。該当する行は行単位で、本物の出力から作った画像に置き換える。画像は、その位置の保存点から複製した子プロセスに、その部分だけを単ページの PDF として出荷させ、変換したものである。

ページの組み立てそのものに介入する環境——`multicols`、`longtable`、分割可能な `tcolorbox` など——と、ページの残り高さを読んで自分の配置を決める環境——`wrapfigure` や横倒しのフロートなど——は、行ではなくブロック単位で退ける。これらは別のプロセスで単独に組み、その際、ページ上の開始位置を 0.25 bp(1 bp は 1/72 インチ) の精度で再現したうえで、配置と分割の判断は本物の出力ルーチンに行わせる。文書全体の紙面に影響する構文——出力ルーチンの

差し替えや二段組など——を含む場合は、文書単位で本物の出力だけを見せる状態に退く。どの段階に退いても、編集そのものは中断されない。

4.6 正本層と増分正本

表示の最終的な姿は常に、`lualatex` を補助ファイルが安定するまで実行した、普通のコンパイル結果である。以後これを正本層と呼ぶ。正本層のコンパイルは、編集応答に含めないだけでなく、編集の合間ごとに走らせることもしない。最初の一度だけは短い遅延 (既定 2.5 秒) で基準を作り、以後は、編集が続けて途絶えたこと (既定 30 秒) と、直前のコンパイルに要した時間に比例する間隔の双方を満たしたとき、あるいは PDF の書き出しが明示的に要求されたときに限って実行する。画面の即時性は速い経路が担っているのだから、正本層は、書き手が一区切りつけた時点での確認と、次節で述べる検証の基準として働けばよい。コンパイルが終わると、変更のないページは LaTeX 自身の出力に置き換わり、同時に検証が走る。

任意機能として、もうひとつの常駐実行を持つ。こちらは本物の出力ルーチンをそのまま使い、1 ページ出荷するごとに、その時点の保存点を残していく。編集が届いたら、最初に変わったブロックより手前で止まっている保存点から、末尾側のページだけを出荷し直せばよい。正本の更新に要する時間も、全文の再コンパイルではなく、編集位置以降のページ数に比例する量になる。以後これを増分正本と呼ぶ。

5 検証

「表示は本物と一致する」を主張ではなく検査できる条件にするため、独立な 4 系統の検証を用意した (表3)。

中心にあるのは、増分で保った状態と、新しく開き直した状態が等しいという条件である。どんな編集列を処理したあとでも、全ブロックの組版結果と値の組は、最終的なソースを新しいエンジンで開いた結果と一致しなければならない。乱数の種を固定したランダム編集がこの条件を編集のまとまりごとに確かめており、開発の過程では、参照値の記帳の誤りや、ページ上の位置に依存するブロックが古い状態のまま固まる不具合が、いずれもこの検査で見つかって直った。ただし、入力途中の壊れた TeX のようにコンパイルできないソースには、収束すべき正解が存在しない。この場合は最後の正常な状態を凍結して番号の揺れを止め、ソースが直れば自動的に元へ戻す。凍結と復帰の経路も、同じ検査の対象である。

検証は実行中にも働く。正本層のコンパイルが終わるたびに、画面に出ている文字と本物の出力の文字を照合する。照合は文字の 2 字組の一致率で行うので、欧文と和文を同じ規則で扱える。食い違いがはっきりしたブロックだけを自動的に格下げし、以後そのブロックは文字として描かず、本物の画像か正本の表示に任せる。ブラウザがフォントの読み込みに失敗した場合も、その

表3　TDOM の検証系統。いずれも普通の LuaLaTeX 出力を基準とする。

検証	内容	精度	実行
レイアウトの照合 (`verify-layout`)	全行の基線の座標を、本物の PDF の内部命令から直接読み出して突き合わせる	0.1 bp	コーパス 12 文書・CI で必須
ページ分割の照合 (`compare-breaks`)	本物の出力ルーチンが呼ばれるたびに素材と寸法を書き出し、こちらの組み立てと突き合わせる	0.02 bp	診断時
増分と新規の一致 (`fuzz`)	ランダムな編集のあと、増分で保った状態と新しく開いた状態を全ブロックで比較する	完全一致	CI(種を固定)
増分正本の照合 (`shipping`)	出荷し直した各ページの文字が、全文コンパイルの結果と一致する	ページ単位	CI

報告が戻ってきて、該当するフォントを使う行が画像による表示に切り替わる。検証は、表示の外側に置かれた検査ではなく、何をどう表示するかを決める情報として、系の内側に組み込まれている。

6　評価

6.1　設定

計測環境は Apple M5、メモリ 16 GB、LuaHBTeX 1.18.0 (TeX Live 2024) である。文書には `article` クラスの英文書を 3 規模用意した (S: 4 ページ・49 ブロック、M: 20 ページ・241 ブロック、L: 30 ページ・361 ブロック。各節は 4 つの段落、番号つきの数式ひとつ、前方参照と後方参照を含む)。プリアンブルは `amsmath` のみの軽い構成で、全文コンパイル時間としては下限に近い条件にあたるから、実際の文書では以下に示す差はさらに開くことになる。

エンジンは保存点の上限を 8 として起動した。既定の 64 より大幅に粗い、メモリを節約する設定であり、後述のとおり編集位置を移した直後の 1 打を遅くする方向、すなわち本手法にとって不利な側の設定である。正本層と、画像を作る処理は、設計上どちらも編集応答に含まれないため (§4)、本計測では止めてある。打鍵は文書の 25%、60%、92% の 3 箇所で各 30 打 (1 文字の挿入) を行い、同期の応答時間の分布を取った。比較の基準としては、同じ文書を普通に全文コンパイルした時間 (`lualatex` を 2 回) を、同じ機械で計測した。計測スクリプトと生データは実装とともに公開している [14]。以下、p50 は中央値、p95 は遅い側 5% を除いた値である。

6.2　結果

結果を表4に示す。まず、定常の打鍵応答は文書規模にほとんど依存しなかった。規模が S から L へ 7 倍になっても、p50 で 16.6〜18.2 ms、p95 でも最悪 73.6 ms にとどまり、すべての規

表4　文書規模と編集応答 (打鍵は 1 文字の挿入、3 箇所 ×30 打の最悪箇所の値)。「定常」は編集位置が定まったあと (2 打目以降)、「初回」は編集位置を移した直後の 1 打目である。全文コンパイルは `lualatex` を 2 回実行した実測値。

	S(4p)	M(20p)	L(30p)
全文コンパイル (2 回)	1.1 s	1.1 s	1.0 s
TDOM の起動 (1 回のみ)	2.0 s	3.1 s	3.1 s
定常の打鍵 p50	16.6 ms	13.9 ms	18.2 ms
定常の打鍵 p95	19.5 ms	73.6 ms	41.5 ms
編集位置を移した直後の 1 打	20.2 ms	92.6 ms	141 ms
節の挿入 (以降の番号が動く)	43.1 ms	90.1 ms	98.5 ms
マクロ定義の変更	20.6 ms	41.9 ms	35.5 ms

模と編集位置で、対話的な応答の目安とされる 100 ms[5] を下回った。これに対して全文コンパイルは、この最軽量の文書群でも約 1 秒を要する (1.1〜1.0 秒)。両者の比はこの下限に近い条件でも約 50 倍であり、実際の文書では全文コンパイルが 10〜30 秒に達する [1] ことを思えば、差はさらに大きい。編集のたびに要していた時間は、起動時に一度だけ要する時間 (L で 3.1 秒) に置き換えられた。

応答の遅い側の裾は、要因を特定できる。編集位置を移した直後の 1 打だけは最大 141 ms(L、文書の中ほど) を要した。新しい位置での 1 打目は、粗い間隔で置かれた保存点から手前のブロックを組み直す費用を負担するためで、2 打目以降は編集位置が固定されて定常値に戻る。組み直す距離は保存点の上限数に反比例するから、既定の 64 であればこの距離は本計測の約 8 分の 1 になる。観測された裾は原理的な限界ではなく、メモリ使用量と初回応答のあいだの、設定で選べるトレードオフである。

影響範囲の大きい編集も、有界の時間で応答した。節の挿入は以降のすべての節番号と参照を動かす編集であるが、応答は 98.5 ms(L) であった。番号の伝播は後回しにされ、編集応答はそれを待たない。マクロ定義の変更は下流全体の組み直しを要する最悪の編集であるが、同期の応答は 35.5 ms にとどまり、組み直しは背後で収束した。編集の体感を決めるのは平均だけではないから、最悪の編集で応答が有界であることには、平均と同等以上の意味がある。

最後に、本計測と同じ設定で検証系統を実行した。編集経路の不変量を固定した試験は 20/20 を通過した。数式・フロート・脚注・参照・和文混植・別プロセスで組む環境を含む 12 文書のレイアウト照合では、基線 318/331 行が一致し、11 文書では全行が一致した。残る 1 文書は本稿の執筆と並行して追加された検証用文書であり、乖離が残っている。

6.3 先行システムとの関係

Lode[1] は、段落単体の組み直しが 0.1〜2 ms、Typst の増分コンパイルが 10、100、300 ページに対して 12.6、76、206 ms であったと報告している (いずれも同論文の計測環境による値である)。TDOM の打鍵応答も、文書規模に依存しないという同じ階級に属するが、計測に含まれる処理の範囲が異なる。TDOM の数値には、プロセスの複製による文脈の保存、改ページのやり直し、フロートと脚注の配置、参照の追跡までが含まれている。Lode の速い経路が対象外とした大域的なレイアウトを処理に含めたうえで、なお応答が同じミリ秒台にとどまる。これが本稿の主要な実測結果である。

7 応用

文書が実行状態を保つ木として管理されることの意味は、待ち時間の短縮にとどまらない。ここでは、この構成の上に成り立つ応用の方向を三つ挙げる。

第一は直接編集である。TDOM の描画命令は、そのひとつひとつが、どのブロックのどの位置から来たかを保持している。したがって、画面上の任意の文字から対応するソースの範囲を引ける。これをミリ秒の編集応答と組み合わせれば、組版結果の上で直接テキストを編集し、組版がその場で追いかけてくるというワードプロセッサ型の操作が、LaTeX の組版品質のまま成り立つ。数式についても事情は同じで、数式の行は行単位で画像とソース範囲に対応づけられているから、数式を選んでその TeX 記述を編集し、検証済みの組版で確かめるという操作が、同じ仕組みの上に載る。見た目と最終出力が食い違うという WYSIWYG 編集の弱点は、§5の検証層が抑える。

第二は、大規模言語モデルに渡す文脈を構造化することである。現状、モデルに LaTeX 文書を扱わせるときの入力は、生の `.tex` 文字列か PDF であり、どちらも文書の構造を失っている。TDOM の内部表現は、ブロック単位の構造、各ブロックが何を参照し何に影響するかの関係、編集ごとの正確な変更集合、そして描画座標とソースの双方向の対応を備えている。変更に関係する部分だけをその組版状態とともに渡すことも、モデルの提案を編集として適用してミリ秒で組版と検証の結果を返すこともできる。バッチコンパイルの所要時間は、この種の反復をこれまで実用の外に置いてきた。

第三に、描画命令の差分は複数の閲覧者へ同時に配れるから共同編集の基盤となり、増分正本は、印刷と同一のページが編集を追いかけてくるという性質から、出版工程における校正の反復にも使える。

8　制限

現在の実装には次の制限がある。

第一に、`microtype`(文字を版面から少しはみ出させ、字幅をわずかに伸縮させる微調整) に対応していない。この機能を使う文書では、文字として描いた行の座標が微小にずれうる。検出したときの退避と、対応そのものが今後の課題である。

第二に、二段組、出力ルーチンを差し替えるパッケージ、紙面全体に描画を加えるパッケージは、文書単位で本物の出力だけを見せる状態に退く (編集は続けられる)。

第三に、脚注がページをまたいで分割される場合に未対応である。分割が必要な状況では早めの改ページで近似し、最終的な姿は正本層が示す。

第四に、常駐するプロセス群はプロセスの複製に依存するため、この機能を持たないネイティブの Windows は対象外である。

第五に、保存点は実際のプロセスであるから、メモリ使用量は文書規模と上限設定に比例する (本評価の設定で、プロセス群の見かけ上の常駐メモリ量の合計は約 748 MB である。ただしこれは複製元と共有している領域を各プロセスに重複して数えた値で、実効はこれより小さい)。

第六に、ひとつの複製系譜の深さに限界がある。120 節・721 ブロック (60 ページ) の予備実験では、系譜が深くなるにつれて組版が遅くなり、既定の打ち切り時間 (12 秒) に達した。起動にも 331 秒を要している (生データは同梱の `measurements-l120.json` に収めた)。本評価の範囲 (361 ブロック) はこの限界の手前にある。

これらのうち第二と第三は段階的な退避の枠組みの中で対応でき、第一は文字を描く層の拡張、第四は単独コンパイル型の縮退動作、第六は保存点を定期的に作り直して系譜を浅く保つことによって、それぞれ対応できると考えている。

9　結論

本稿では、文書を実行状態の列として保つ組版ランタイム TDOM を提案し、無改造の LuaTeX の上で、編集応答を文書規模からほぼ独立なミリ秒台にできることを、実装と実測によって示した。編集はブロック単位の組み直しとして処理され、改ページ・フロート・脚注・相互参照・目次は、全文の再コンパイルを待たずに更新される。表示と本物の出力の一致は 4 系統の検証によって機械的に検査され、検証できない構文は本物の出力による表示へ段階的に退く。評価では、プリアンブルが軽い下限に近い条件でも、全文コンパイル 1.0 秒に対して定常の打鍵が p95 41.5 ms となり、全文コンパイルが 10〜30 秒に達する実際の文書 [1] では、この差は 3 桁を超える。

この短縮は、組版アルゴリズムを変えたり近似したりして得たものではない。使い捨てのプロ

セスモデルを、状態を保つランタイムに置き換えたことの帰結である。Web において文書が生きた木となったことが対話的なアプリケーションの基盤になったように、組版においても同じ移行が可能であることを、本稿の実装と測定は示している。

参考文献

[1] C. Lode. Real-Time LuaTeX: Recompiling Large Documents in 1 ms. *TUGboat*, draft, 2026.

[2] F. Bour. TeXpresso: Live rendering and error reporting for LaTeX. *TUGboat*, 44(2):185–192, 2023.

[3] D. E. Knuth and M. F. Plass. Breaking Paragraphs into Lines. *Software—Practice and Experience*, 11(11):1119–1184, 1981.

[4] D. E. Knuth. *The TeXbook*. Addison-Wesley, 1986.

[5] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[6] M. Haug. Fast typesetting with incremental compilation. Master's thesis, Technische Universität Berlin, 2022.

[7] Typst. `https://typst.app/` (2026 年 8 月閲覧).

[8] SwiftLaTeX: The visual LaTeX editor. `https://github.com/SwiftLaTeX/SwiftLaTeX`.

[9] TeXlyre. BusyTeX: TeX live compiled to WebAssembly. `https://github.com/TeXlyre/texlyre-busytex`, 2026.

[10] H. T. Thành. *Micro-typographic extensions to the TeX typesetting system*. PhD thesis, Masaryk University Brno, 2000.

[11] R. Schlicht. The `microtype` package: Subliminal refinements towards typographical perfection. `https://ctan.org/pkg/microtype`, 2024.

[12] W3C. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, 1998.

[13] LuaTeX development team. *LuaTeX Reference Manual*. `https://www.luatex.org/`.

[14] TDOM Engine (ソースコード・検証スイート・本稿の計測スクリプト). `https://github.com/Fermion-company/tdom-engine`, 2026.