paper-highlighter: Synchronized PDF-to-Video Narration with Neon Highlighting and News-Style Camera Motion

Shyamal Suhana Chandra
Sapana Micro Software

April 2026

Abstract

We present *paper-highlighter*, a macOS-native command-line system that converts portable document format (PDF) documents into narrated high-definition video with synchronized visual highlighting. The pipeline extracts text via PDFKit and optional Apple Vision optical character recognition (OCR), synthesizes speech with sample-accurate word-boundary timing through `AVSpeechSynthesizer`, maps global text ranges to page coordinates, and renders H.264/AAC MP4 output with a single active neon overlay per frame—analogous to broadcast captioning and C-SPAN-style document cameras. Performance engineering centers on *unique-frame deduplication*: visually identical states are rendered once and reused across many 30 fps timeline slots, reducing compositing cost from $\mathcal{O}(\text{frames})$ to $\mathcal{O}(\text{cues})$ for static camera modes. Word-by-word mode further adds smooth pan-and-zoom camera interpolation for close-up reading. On a development workstation, a one-page smoke-test document converts in approximately two seconds at 1920×1080; a full academic paper ($\sim$6 minutes of speech, $\sim$11,500 frames) completes in under 40 seconds with camera motion enabled. The implementation is written in Swift 6, ships as an open-core library plus CLI, and includes eleven automated unit tests covering text indexing, cue timing, highlight mapping, and camera planning.

1 Introduction

Long-form reading on screen differs materially from listening while watching a presenter trace a document. Televised legislative sessions, news-agency document cameras, and accessibility tools all combine *narration*, *spatial indication* (what is being read now), and *camera motion* (where to look). Portable Document Format remains the dominant interchange format for papers, books, and reports, yet few lightweight tools produce broadcast-quality synchronized video directly from PDF without a heavyweight video editor.

This project asks: *Can a single-machine macOS pipeline turn a PDF into a narrated MP4 with word-synchronized highlighting and optional close-up camera work, using only system frameworks?*

Our answer is **paper-highlighter**, which:

- indexes PDF text natively and falls back to Vision OCR for scanned pages;
- exports timed speech with per-word or per-sentence cues via the speech synthesizer delegate API;
- overlays one neon highlight at a time on cached page thumbnails;
- encodes video with hardware-friendly H.264 and muxes AAC audio with passthrough where possible; and

- optionally pans and zooms the viewport in word mode to emulate a news-agency document camera.

The remainder of this paper summarizes background, the end-to-end algorithm, empirical observations from development benchmarking, limitations, and future work.

2 Background

2.1 Related work and motivation

Text-to-speech (TTS) accessibility tools (e.g., VoiceOver, Read&Write) emphasize audio but rarely produce shareable video with pixel-accurate highlight alignment to source layout. Video explainers typically rely on manual screen recording or non-reproducible slide decks. Research systems for "video papers" and synchronized narration exist in academic multimedia, but often require LaTeX-specific tooling or cloud services.

Broadcast document cameras solve the *where to look* problem through optical zoom and operator control. We approximate this computationally by centering a virtual camera on the active word bounding box and smoothly interpolating between cues.

2.2 Platform and dependencies

The system targets **macOS 13+** and is implemented in **Swift 6**. Core frameworks include:

- **PDFKit** — page rasterization, selection bounds, and text extraction;
- **Vision** — OCR word boxes for pages without a text layer;
- **AVFoundation** — TTS export (`AVSpeechSynthesizer.write`), AAC re-encoding, `AVAssetWriter` H.264 encoding, and passthrough muxing;
- **Core Graphics / Core Video** — CPU compositing and pixel-buffer creation;
- **ArgumentParser** — CLI UX.

No cloud API, GPU compute shader, or third-party PDF engine is required, which keeps the tool suitable for offline and privacy-sensitive documents.

2.3 Coordinate systems and orientation

PDF uses a bottom-left origin; Core Graphics bitmaps and video frames use top-left origins. Highlight rectangles are mapped through a per-page `PDFView` layout cached once per page. Final frames undergo a composed orientation transform (rotation and axis flips) at pixel-buffer creation time so that page content appears upright in MP4 players—a recurring source of bugs in PDF-to-video pipelines that we treat as a first-class encoding concern.

3 System architecture

The pipeline comprises five stages:

1. **Text indexing** — Build a global UTF-16 string and per-page spans; optionally OCR scanned pages in parallel.
2. **Timed speech export** — Synthesize audio to PCM, record `willSpeakRangeOfSpeechString` callbacks as word cues, finalize end times, optionally merge cues to sentence granularity.

3. **Timeline planning** — Convert cues into contiguous *timeline segments* ($t_{\text{start}}, t_{\text{end}}$, page, highlightRects); merge adjacent identical segments.

4. **Parallel rendering** — Cache each page's base thumbnail; render unique highlight composites in a worker pool.

5. **Video encoding** — Apply optional camera transform per frame; write H.264; re-encode speech to AAC in parallel; mux with `AVAssetExportPresetPassthrough`.

Figure 1 summarizes data flow.

PDF → PDFTextIndex (PDFKit + Vision OCR)
→ TimedSpeechResult (TTS + SpeechCue[])
→ FramePlanner (TimelineSegment[])
→ ParallelFrameRenderer (unique CGImages)
→ VideoComposer (camera + H.264 + AAC mux)
→ **MP4**

Figure 1: End-to-end data flow in paper-highlighter.

4 Algorithm

4.1 Global text index

For each page p, let T_p be extracted text from PDFKit. If $|T_p| < \tau$ (empty or whitespace-only), a Vision OCR pass produces words $\{(w_i, b_i)\}$ with bounding boxes b_i in page space. Page spans record (p, T_p, s_p) where s_p is the global start offset in the concatenated document string $T = \bigoplus_p T_p$.

Global range lookup maps a cue's `NSRange` (ℓ, n) to page-local selections for highlight geometry.

4.2 Timed speech cues

`AVSpeechSynthesizer.write(_:bufferHandler:)` streams PCM while the delegate receives `willSpeakRangeOfSpeechString` with the word about to be spoken. Each callback appends a **SpeechCue** $(\ell, n, t_{\text{start}})$ where t_{start} is derived from cumulative samples written divided by sample rate (44.1 kHz). End times are finalized as the next cue's start, or audio duration for the final cue.

In *sentence mode*, word cues are aggregated to sentence ranges using `NSString.enumerateSubstrings(_:byS`

4.3 Highlight geometry

For native text, PDFKit's `page.selection(for:)` and `selectionsByLine()` yield axis-aligned bounds per line fragment. For OCR text, word boxes from Vision are used directly. Bounds in PDF space are converted to canvas coordinates via a cached `PDFView` layout matching the thumbnail scale.

Only **one active highlight** is drawn per frame—prior words are not simultaneously colored—matching the "single laser pointer" semantics requested for clarity.

4.4 Unique-frame optimization

Let $F = \lceil 30 \cdot (T_{\text{audio}} + \delta) \rceil$ be total frames at 30 fps with tail hold $\delta = 0.5$ s. Naïve rendering costs $\Theta(F)$ full-frame composites.

Instead, segments are merged when (*page*, highlightRects) are identical. Let U be the number of unique visual specs ($U \ll F$ typically). The worker pool renders U images; encoding duplicates pixel buffers for repeated timeline slots.

For a 4 s test clip with 13 word cues, $F \approx 129$ but $U \approx 14$, yielding roughly an order-of-magnitude reduction in compositing work.

4.5 Camera planner (word mode)

When camera movement is enabled (default in word mode), each segment defines a target **CameraState** $(\mathbf{f}, z)$ with focus point $\mathbf{f}$ at the highlight bounding-box center and zoom

$$z = \min\left(z_{\max},\ \min\left(\frac{W}{w_{\text{padded}}},\ \frac{H}{h_{\text{padded}}}\right)\right),$$

clamped to $z_{\max} = 3.75$, with padding for broadcast-style margins. Segments without highlights use a wide shot ($z = 1$) centered on the page.

Between consecutive segments, focus and zoom interpolate over

$$\Delta t = \min(0.35\,\text{s},\ 0.4 \cdot (t_{\text{end}} - t_{\text{start}}))$$

using smoothstep easing $s(t) = 3t^2 - 2t^3$. Each output frame applies

$$\mathbf{x}' = z(\mathbf{x} - \mathbf{f}) + \mathbf{c}_{\text{viewport}},$$

then orientation correction and H.264 encoding.

Camera mode renders per frame ($\Theta(F)$ compositing) because panning is continuous; static mode retains unique-frame reuse through the encode loop.

4.6 Pseudocode

Algorithm 1 Convert PDF to synchronized MP4

Require: PDF path P, options (mode, voice, size, workers)
Ensure: MP4 path O

$D \leftarrow \text{LoadPDF}(P)$; $I \leftarrow \text{BuildIndex}(D)$ ▷ parallel OCR if needed
$S \leftarrow \text{ExportSpeech}(I.\text{fullText})$ ▷ TTS + SpeechCue[]
$(\text{segments}, \text{cache}) \leftarrow \text{FramePlanner.build}(I, S.\text{cues})$
$\text{images} \leftarrow \text{ParallelRenderUnique}(\text{segments}, \text{cache})$
Start AAC re-encode of S.audio asynchronously
for $k = 0$ **to** $F - 1$ **do**
    $t \leftarrow k/30$; $\sigma \leftarrow \text{segmentAt}(t)$
    $\text{img} \leftarrow \text{images}[\sigma.\text{spec}]$
    if camera enabled **then**
        $\text{img} \leftarrow \text{ApplyCamera}(\text{img}, \text{CameraPlanner}(t))$
    end if
    Append H.264 frame PixelBuffer(img) at time t
end for
Mux passthrough AAC; **return** O

5 Benchmarks and observations

All figures below are *development measurements* on Apple Silicon macOS during implementation; they are indicative rather than a formal benchmark suite.

Table 1: Representative conversion times (single machine, `swift run paper-highlighter convert`).

Document	Resolution	Audio	Frames	Wall time
1-page smoke PDF	640×360	$\sim$4 s	129	$\sim$2 s
1-page smoke PDF	1920×1080	$\sim$4 s	129	$\sim$2 s
Academic paper (CSN)	1920×1080	$\sim$6.4 min	$\sim$11,458	$\sim$36 s

5.1 Unique-frame deduplication

For the smoke PDF, unique composites $U \approx 14$ versus $F \approx 129$, confirming that cue-driven deduplication dominates static-camera performance. Encoding with passthrough mux avoids a second full video re-encode compared to highest-quality export presets.

5.2 Parallelism

OCR for scanned pages and unique-frame compositing use a worker pool sized to CPU core count (`-jobs`). AAC re-encoding runs concurrently with video rendering via Swift `Task` overlap.

5.3 Correctness testing

The Swift Testing suite contains eleven unit tests: page-span lookup, cue finalization, sentence splitting, video-size parsing, PDF highlight mapping on synthetic PDFs, and camera zoom/transition/wide-shot behavior.

5.4 Bottlenecks

- **TTS synthesis** is sequential and dominates wall time for very long documents.
- **Camera mode** requires per-frame compositing and scales with F.
- **OCR** is invoked per scanned page; accuracy depends on scan quality.
- Highlight vertical alignment remains sensitive to PDFView vs. thumbnail layout on some fonts.

6 Future work

- **GPU compositing** via Metal or `vImage` to accelerate camera-mode frame generation.
- **Keyframe caching** for camera motion: quantize $(\mathbf{f}, z)$ to reduce redundant per-frame renders.
- **Neural TTS** (on-device or API) for higher-quality narration while preserving word timestamps.
- **Multi-column and footnote-aware** highlight geometry beyond axis-aligned line boxes.
- **Cross-platform** export (iOS preview, Linux headless render via alternative PDF engines).
- **Subtitle tracks** (WebVTT / embedded CEA-608) derived from the same SpeechCue timeline.
- **Rigorous benchmark harness** with fixed hardware, dataset of PDFs, and quality metrics (sync drift in ms, OCR word error rate).

7 Conclusion

paper-highlighter demonstrates that a self-contained macOS CLI can produce narrated, highlight-synchronized video from arbitrary PDFs using system frameworks alone. The central design insight is *timeline segmentation*: speech cues induce a piecewise-constant visual state sequence, enabling unique-frame caching that makes short and medium documents convert in seconds rather than minutes. Word-by-word mode extends this with smooth pan-and-zoom camera planning that approximates news-agency document presentation. Remaining work focuses on GPU acceleration, broader PDF layout coverage, and formal evaluation.

References

[1] Apple Inc. *PDFKit Framework Reference*. https://developer.apple.com/documentation/pdfkit (accessed 2026).

[2] Apple Inc. *Vision Framework*. https://developer.apple.com/documentation/vision (accessed 2026).

[3] Apple Inc. *AVSpeechSynthesizer*. https://developer.apple.com/documentation/avfaudio/avspeechsynthesizer (accessed 2026).

[4] Apple Inc. *AVAssetWriter*. https://developer.apple.com/documentation/avfoundation/avassetwriter (accessed 2026).

[5] Apple Inc. *The Swift Programming Language (6.0)*. https://docs.swift.org/swift-book/ (accessed 2026).

[6] C-SPAN. *Video Library and Document Coverage*. https://www.c-span.org/ (accessed 2026). Broadcast reference for synchronized document reading style.

[7] Perlin, K. *Improving Noise*. SIGGRAPH 2002. Classic smoothstep interpolation used in camera easing.

[8] Chandra, S. S. *paper-highlighter*. Sapana Micro Software, 2026. https://github.com/Sapana-Micro-Software/paper-highlighter

Acknowledgement

The author thanks **Kailash Chandra** for his support and guidance at Sapana Micro Software throughout the development of this project.