DocuGram: Turning Screen Recordings into Documents

Laurent Denoue
FXPAL
3174 Porter Drive
Palo Alto, California 94304 USA
denoue@fxpal.com

Scott Carter
FXPAL
3174 Porter Drive
Palo Alto, California 94304 USA
carter@fxpal.com

Matthew Cooper
FXPAL
3174 Porter Drive
Palo Alto, California 94304 USA
cooper@fxpal.com

ABSTRACT
In this paper we describe DocuGram, a novel tool to capture and share documents originating from any application. As users scroll through pages of their document inside the native application (Word, Google Docs, web browser), the system captures and analyses in real-time the rendered video frames and reconstitutes the original document pages into an easy to view HTML-based representation. In addition to detecting and regenerating the document pages, a DocuGram also includes the interactions users had over them, e.g. mouse motions and voice comments. A DocuGram allows users to flexibly share enhanced documents across applications.

Keywords
Document capture; image processing; video processing; interactive documents.

1. INTRODUCTION
We have numerous available choices to edit and view documents, ranging from native applications such as Word and PowerPoint, to online editing tools like Google Docs and Microsoft Office 365. We also have many ways to share these documents, such as by attaching them in email, embedding them in web pages, or simply sending a link of their online location.

But individual source applications often require users to devise custom means to share that document. One could simply attach a Word document by email, hoping the recipients will have Word installed; or one might export the document as a PDF; or generate a sharable link to the document.

While already complex for our own documents, capturing and sharing is even harder for content we don't author, such as pages from a scanned book hosted on Google Books or a slide deck shown during web conferences. Even the seemingly mundane task of sharing the URL of an online article could cause problems to the recipient (e.g. a pay-wall from the Wall Street Journal).

Furthermore, sharing a document often means leaving out comments and interactions we have over the document. Without options, people often need to decontextualize their comments, e.g. in the accompanying email message.

Or export the document into a PDF editor, highlight, and then hope the recipient will also have a PDF reader that understands and renders annotations.

One last resort is for users to record their screen. Although videos can be indexed (see [1] for examples), recipients would typically be shown a video player, making it hard to page through and otherwise read the "document".

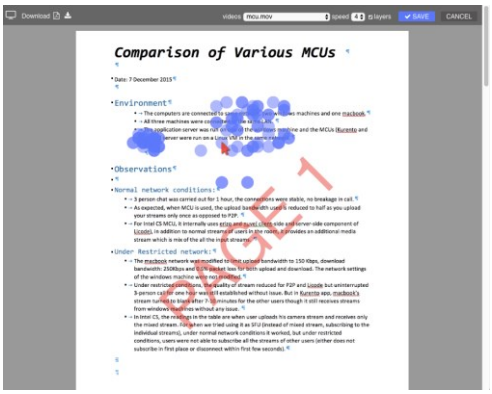

Figure 1. A sample DocuGram is a document reconstituted from a screen recording of any application showing the original document; mouse actions and pages are preserved

With DocuGram, we aim at reconstituting the original document from a screen-recording of that document captured by the user. Using image analysis techniques, the system processes the screen recording to produce a copy of the original document that is immediately available online as a Web page, viewable on any device with a basic web browser. This web-based viewer shows each page as an image, including any interaction and voice comments that the author added while recording, see Figure 1.

We describe below how the system analyses in real-time the video frames captured upon a user's request and turns them into viewable document pages, along with users' interactions over the document while recording[1].

2. INTERACTION DESIGN
To create a DocuGram, users pick one window on their desktop that displays the document to capture, e.g. the Word window showing a DOC file, and click "Start".

DocEng '16, September 12-16, 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4438-8/16/09 $15.00
DOI: http://dx.doi.org/10.1145/2960811.2967154

[1] A video is available at https://youtu.be/phS0TrwZ4Tg

They then interact freely with the document inside the application window (e.g. Word), scrolling up and down through pages they wish to capture and share. During the recording, they have full access to all functions provided by the application, such as selecting text passages and moving their pointer over areas of the document.

Because DocuGram records the screen as a video, users can also talk at the same time, e.g. to describe a figure that needs to be modified or express a feeling about a particular passage in the document. When they are done recording, users simply click the "Stop" recording button.

The system is able to recognize the two major kinds of document viewing metaphors: scrolling and pagination. All word processors or PDF viewers typically implement scrolling, while pagination is used for showing slide decks, e.g. PowerPoint. Page detection is a core problem in document image analysis [5], and has also been studied using video cameras [6]. The screen-based context here is distinct and provides opportunities to accelerate our visual analysis.

Using the same system, users can also pick a browser window showing a video of a lecture, in which the speaker is showing slides. At the end of the session, the user will have a reconstituted copy of the slides that were shown during the video lecture.

During users' interactions, the system analyses every captured frame of the window and turns them into a final DocuGram: a copy of the original document. In order to generate this copy, the system applies these important steps: Region of Interest (ROI) detection, image stitching, and interaction lifting.

All these image-processing algorithms run in real-time inside the user's web browser in JavaScript. The video frames are captured using the media devices API available to modern web apps, allowing us to capture either the full desktop or individual windows. The rest of this paper describes the algorithms required to generate a DocuGram.

3. SYSTEM DESCRIPTION

3.1 Video pre-processing

After the user has stopped recording her window, the system plays the recorded video into a VIDEO element, draws its frames at 30 frames per second into a CANVAS element and only keeps different frames using a simple image difference function comparing corresponding gray scale pixels with a fixed threshold.

3.2 ROI detection

When users start recording a given window, the system starts a new MediaRecorder that saves the video of the window into a local file later accessible to the web application.

Once the user is done recording, the system processes the first frame, binarizes it and looks for long vertical and horizontal segments. It combines them to determine the largest rectangle as the ROI for that window, shown in Figure 2.

If the system has correctly detected this ROI, the user simply clicks over the identified region and the system proceeds to the next step. Alternately, users simply drag a rectangle to manually specify the ROI to use. Some document types such as web pages do not have clear paginated layout, making it hard or impossible for an automatic ROI detector to find the correct rectangle.

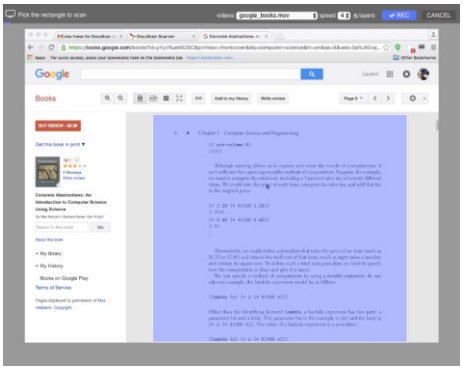

Figure 2. Detected ROI is shown in blue to the user; they can overwrite the automatic detection by dragging another rectangle

3.3 Image Stitching

Image stitching is a critical analytical step for the system. Brute force comparisons between frames to determine their vertical shift would be very CPU intensive. Instead, we borrow techniques from the image stitching literature [2] that use key point detection and matching as the basis for finding generic transformations between pairs of images and adapt it to our domain. These methods are fast and robust, making them appealing for a real-time implementation in Javascript inside a web application.

In our case, the goal of the image-stitching step is to find the amount of vertical scroll that occurred between frames. After gray scale conversion, a fast corner detector runs across the frame, yielding a set of key points. We developed a custom key point extractor that works well for textual content. For a given pixel $P_1=I(x,y)$ in the grayscale frame, we look at its three neighboring pixels $P_0=I(x-1,y)$, $P_2=I(x+1,y)$ and $P_3=I(x,y-1)$. P_1 is selected as a key point if the following condition is true:

$$|P_0-P_1| > 32 \text{ AND } |P_0-P_2| <= 6 \text{ AND } |P_0-P_3| > 32$$

The idea is to select keypoints that have a sharp increase in contrast when scanning from left to right and top to bottom (thus a threshold of 32), while having the next pixel similar (the threshold of 6). Intuitively, it corresponds to corners of character glyphs (see Figure 3). To limit the number of key-points, if a pixel P_1 is selected we skip 128 pixels to the right on the same line.

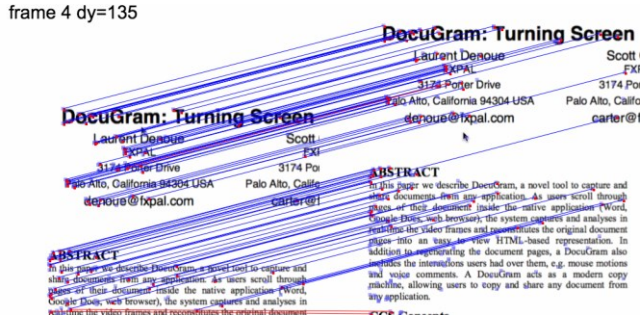

Figure 3. BRIEF vectors are matched in a vertical shift up: left previous frame, right next frame; blue lines join matching key-points; red lines show mismatches

Each key point is represented by a BRIEF descriptor, which is a binary descriptor described in [3] that represents the neighboring pixels of a given key point by a 512 dimensional vector. Vectors

can be quickly assessed for similarity with a simple Hamming distance, implemented using XOR operations.

Given two images, we thus have two lists of key-points and their BRIEF descriptors. A reciprocal matching algorithm finds the closest matches, but adds a specific constraint for our domain: a match is only allowed between 2 BRIEF vectors if their corresponding key point locations have the same X coordinate. In practice, we have observed that most people view their documents without a horizontal scrollbar, thus letting us optimize the execution of the matching to finding only vertical scroll offsets. This constraint can be relaxed as needed with a corresponding loss of efficiency.

The estimated vertical scroll value ΔY between two frames is found by keeping the most popular value between pairs of matching key points.

3.4 Handling non-scrolling document viewers

Our research has shown that ΔY can erroneously be set to zero, especially on documents that have little text content such as PowerPoint slides. Another example would be an online Google Presentation where users flip through pages of a presentation, or a presentation given by a co-worker during a live web meeting.

To recover from these mistakes, we empirically found that ΔY is correct if the number of matching key points is at least 20 percent of the average number of key points found in each image. If not, the system detects that the frames correspond to entirely different pages and sets ΔY to the frame height instead of zero. When stitching all frames later, the result will be that pages will neatly be paginated as in the original document.

On the other hand, the matching step can erroneously report ΔY as non-zero. This can also happen on PowerPoint slides where letters match with the wrong key points. We also empirically found that this happens when the popular value found in the previous step is found less than three times. The algorithm determines that the document viewer is pagination (versus scrolling) if it happens more than twice.

3.5 Detecting mouse, text and voice actions

Capturing mouse and text actions performed by the user while recording is important to preserve contextual information as users markup documents [4]. An author can for example highlight specific words for revision with a co-author, or a teacher can explain a chart or figure by moving her cursor, or a lawyer can circle a whole section with her mouse cursor and ask a question verbally.

To detect these actions, it is enough to recognize when people only move their cursor and stop scrolling or paginating through the document. Conveniently, these times correspond to when the stitching algorithm yields $\Delta Y=0$. For such frame pairs, we thus compare their absolute difference and cluster the positions of their changed pixels into blobs of 64x64 pixel areas. Clustering changed pixels onto a grid allows for a nice speed up as well as suppresses noise due to other smaller changes that might occur during the recording. The winning cluster is reported as the detected changed location, used in the next section when overlaying the action on the final DocuGram.

3.6 Generating the final DocuGram

Given the set of frames and estimated shifts between them, the system generates a single composite image whose height corresponds to the sum of the deltas plus once the ROI's height. Each frame is then copied at its corresponding Y position based on the accumulated delta offsets until that index. Special attention needs to be paid for cases when the user started recording her window at a later page and scrolls up.

Once the composite image is created, the system identifies likely page breaks. Because of the ROI step earlier, it is rather easy to identify page breaks as long horizontal lines that cross the whole ROI's width. Special attention is paid to small interrupted fragments; while recording, the mouse cursor has sometimes been found to overlap page boundaries, thus creating little discontinuities in the horizontal segments. To speed up implementation, our fast binarization step only computes the vertical gradient of the composite image so that it only detects horizontal edges.

Once page breaks have been found, the system cuts the tall image into as many smaller page images as necessary, padding the last one with white space in case the user had not completely captured it, thus giving the inferred document pages a more uniform look.

Finally, detected actions are overlaid on the corresponding page images as a DIV element that depicts a mouse pointer. When clicked, the original motion path (as recovered from the previous step) is used to play an animation of the fake mouse cursor. The corresponding voice segment is played at the same time by seeking the audio track to the start time and playing it until the end of the segment.

3.7 Editing and annotating before sharing

Unlike sharing a document as a link or by email as an attachment, DocuGrams can be edited before being published and shared. For example, an author might want feedback on slide 1 and 5 of her presentation, but during pagination all slides 1 through 5 were captured by the system. Using the edit button, she can easily delete the page images no longer desired. She can also re-record her voice, add new highlights, or record new mouse motions that she might have forgotten to produce while recording.

An optional step allows users to apply OCR on each page image of the document page images. We currently use the Javascript port of GOCR (GNU Optical Character Recognition) to perform this step. Each detected word is represented by a transparent DIV overlaid over the page image at its bounding box. It allows users to select text from the DocuGram as if it was the original document, including highlighting parts of it.

4. EXPERIMENTS

We ran two experiments to evaluate the system accuracy and speed. The first experiment tested whether document pages were correctly reconstituted by the stitching and pagination algorithms, and if the mouse actions were correctly identified. This experiment included 17 screen recordings: 5 word documents, 2 PowerPoint slide decks, 4 PDF documents (2 using smooth scrolling, 2 using normal scrolling), 2 online Office 365 Word documents, and 4 web pages, including one from Google Books.

The ROIs were correctly identified for all videos except two web pages with no obvious line separators and one PPT exported to a long PDF that contained a tall figure including all slides on a single page.

Stitching worked well on all samples. Pagination also performed well apart from three web pages which did not contain natural page breaks. The last web page (Google Books) worked correctly because the interface showed page breaks.

Mouse motion was correctly identified, but failed to handle selected text areas; this is a limitation of our current implementation that selects one winning region in the difference maps instead of looking at region shapes.

We ran a second experiment to better understand the speed of capture without the user spending time talking or moving her mouse cursor while recording. As shown in Table 1, this experiment included screen recordings of 4 PDF documents, 3 PowerPoint files, and 3 web pages. Averaging the results, PowerPoint slides can be captured at 3.4 pages per second, documents at 1.5 pages per second, and web pages at 1.08 pages per second (by counting a page as being 600 pixels tall). Again, stitching worked correctly for all recordings, but pagination failed on one web page where strong horizontals confused the page break algorithm.

Table 1. Capturing documents in popular applications

Document type	Duration of recording (seconds)	Number of pages/pixels
PDF	4.5	7
PDF	12.4	14
PDF	18.7	32
PDF	4.3	7
PPT	8.0	23
PPT	4.0	14
PPT	6.0	24
WEB	4.7	3405 (pixels)
WEB	6.2	4318 (pixels)
WEB	6.6	3644 (pixels)

5. DISCUSSION

Early experiments show that capturing a DocuGram can be fast, especially if no interactions need to be captured. For example, capturing PowerPoint slides is as fast as scrolling through all slides with a mouse scroll wheel.

Unlike sharing the original document, users can control which pages are shared (except for web pages since the system does not yet paginate them), add comments or mouse motion, and be sure their shared documents are viewable on any web browser, including mobile phones. Users can also capture documents being shared by others, e.g. slides or a PDF document shared by a colleague during a live web conference.

Compared to a pure screen-recording, DocuGrams allow the recipients to navigate the copy as if working with the original, scrolling up and down pages, even selecting text, while still being able to replay actions created while recording, such as mouse motions and voice comments. DocuGrams are also much smaller files than the video of a screen recording and can be easily exported to an image-based PDF for printing.

6. CONCLUSION AND FUTURE WORK

The algorithm runs at 20 frames per second now, meaning that it could run as the user is capturing the recording. This would dramatically improve the user experience, especially for cases where the user is viewing an online lecture and needs to use the DocuGram as it is being captured.

Real-time would however require improvements to the ROI detector. One idea is to involve the user at the beginning of the recording, another is to leverage several seconds worth of recording to determine what parts of the frames are moving. Still, more work would be required for paginated documents such as PowerPoint slides, or YouTube videos showing slides along with speaker or animated transitions [7]. Also, web pages do not have obvious page breaks: one could automatically add page breaks as when the pages are printed.

We would also like to explore the full life-cycle of DocuGrams. For example, recipients could send back comments to the original author. During a document-authoring scenario, one could imagine the DocuGram to observe the Word document in a window and synchronize its scroll to show the author the corresponding comment left by a co-author on the DocuGram, thereby facilitating document correction.

Finally, document analytics could be integrated to support awareness of use. For example, a teacher sharing a DocuGram could be shown where students have looked, what voice comments were played, what pages were read, helping them increase their awareness of student engagement, a strong problem in for online education.

7. REFERENCES

[1] Toni-Jan Keith Palma Monserrat, Shengdong Zhao, Kevin McGee, and Anshul Vikram Pandey. 2013. NoteVideo: facilitating navigation of blackboard-style lecture videos. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 1139-1148.

[2] L. Zhu, Y. Wang, B. Zhao and X. Zhang, "A Fast Image Stitching Algorithm Based on Improved SURF," Computational Intelligence and Security (CIS), 2014 Tenth International Conference on, Kunming, 2014, pp. 171-175.

[3] Calonder, M., Lepetit, V., Strecha, C. and Fua, P., 2010. Brief: Binary robust independent elementary features. Computer Vision–ECCV 2010, pp.778-792.

[4] A. Adler, A. Gujar, B. L. Harrison, K. O'Hara, and A. Sellen. A diary study of work-related reading: design implications for digital reading devices. In Proceedings of ACM CHI, 241–248. 1998.

[5] T. M. Breuel. High performance document layout analysis. Proceedings of the Symposium on Document Image Understanding Technology. 2003.

[6] C. Kim, P. Chiu, S. Chandra. Dewarping Book Page Spreads Captured with a Mobile Phone Camera. Camera-Based Document Analysis and Recognition. Vol. 8357 of the series Lecture Notes in Computer Science, pp 101-12, 2014

[1] John Adcock, Matthew Cooper, Laurent Denoue, Hamed Pirsiavash, and Lawrence A. Rowe. 2010. TalkMiner: a lecture webcast search engine. In Proceedings of the 18th ACM international conference on Multimedia (MM '10). ACM, New York, NY, USA, 241-250.