Read#: A System for Active Reading

Sam Bartleman
UC San Diego
9500 Gilman Dr.
La Jolla, CA 92093 USA
sbartlem@ucsd.edu

Julia Lin
UC San Diego
9500 Gilman Dr.
La Jolla, CA 92093 USA
jyl011@eng.ucsd.edu

Neema Mahdavi
UC San Diego
9500 Gilman Dr.
La Jolla, CA 92093 USA
mahdavi@ucsd.edu

Amer Sinha
UC San Diego
9500 Gilman Dr.
La Jolla, CA 92093 USA
amsinha@ucsd.edu

CHI'13, April 27 – May 2, 2013, Paris, France.
ACM 978-1-XXXX-XXXX-X/XX/XX.

Abstract
In this paper, we introduce Read# a new system for document interaction and active reading activities. Read# allows for digital interactions synchronously with paper. The system allows a computer to be invisibly integrated into the reading environment and naturally track and record a reader's gestures while minimizing the number of deliberate actions a user needs to make to annotate and comprehend a piece of reading.

Author Keywords
Interactive Workspace; Active Reading; Tabletop Computing

ACM Classification Keywords
H.4 Information Systems Applications; H.5 Information Interfaces and Presentation; H.5.2 User Interfaces

Introduction
Current trends in technology have been transforming analog objects into digital objects. However, paper is one artifact that people still prefer as their medium rather than its digital counterparts. We have developed a system to allow the user to continue using paper as their preferred medium while being able to enjoy the features of digital documents; features such as defining words, digital highlights, and annotating a digital copy. Read# is designed to promote digitizing reading material for later collaboration, online or offline,

something which a hard copy could not accomplish. We also take into account the major complaints that users typically have when digitizing paper, including having to stop the reading process to define words and having to manually recreate highlights page by page at a later time. The system encourages higher productivity by speeding these processes along and leaving a user free to focus on the reading material.

Motivation and Related Work

One of the driving forces behind our work was based on Sellen and Harper's work of the "Myth of the Paperless Office." Rather than getting rid of paper, technology has shifted the point at which paper is used for doing work [2]. Instead of providing another interface that users would have to learn, our challenge is to support how people already do their work. However, there is a reason that digital technologies have not replaced paper. The affordances of paper are about what people can do with paper [2]

Presented by Sellen and Harper, one of the disadvantages of paper is the interactional problem it provides [2], namely the limitations of the use of paper. One of our primary motivations was the problem that paper and the work done on paper cannot be accessed remotely. Another motivation for continuing to use paper was the ubiquity that paper affords and that paper is still the medium of choice for reading, even though most high-tech technologies are to hand [2].

The microanalysis of active reading performed by Hong et al. provides us with a framework with which to base our gestures and also provides considerations for designing an augmented desktop workspace [3].

Using paper and the user's natural workplace, we are also able to use the user's natural work environment without having to worry about screen real estate and resolution. The spatial dynamics of this environment is also part of the user's cognitive process that simplify internal computation, as how people manage the spatial arrangement of items around them is an integral part of the way we think, plan, and behave [4].

System Description

Hardware and Software Requirements

Read# requires multiple components to function: one Microsoft Kinect, the Interactive Spaces SDK 12/14/12 8:47 PM, a projector, and the use of a webcam. For our environment, we used the Microsoft LifeCam. The implementation of Interactive Spaces was critical in order for seamless integration into the environment for the Read# system.

Gestures

The gesture detection offered in Interactive Spaces was very minimal; after calibration, the SDK would detect fingertip presses via event handlers, and nothing more. The presses were also required to be spot-on towards the Kinect and have a forty-five degree angle between the finger the surface.

The highlight gesture specifically looks for six fingers down on the paper (or five fingers and a writing utensil). One contact would be made by the writing hand and five on the opposite hand, which is a gesture noted in the analysis done by Hong et al [3]. Six contacts is a rarity during the usual writing process so it made it easier for the program to ignore noise. When six simultaneous contacts are detected, the program begins waiting for the highlight gesture to end before

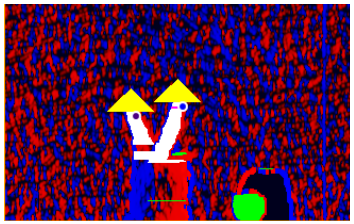

Figure 1: The "Define" Gesture as seen through the Kinect.

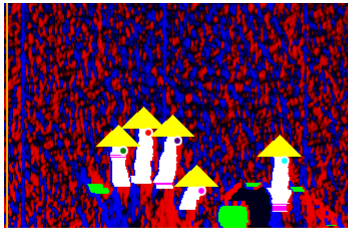

Figure 2: The "Highlight" Gesture as seen through the Kinect. The Kinect cannot distinguish between a finger or a pen as the 6th contact point.

It's assumed that the highlight ends when there are removed was the pen (or finger) that was touching the highlighted area.

The "define" gesture consists of two contacts for an extended period of time. The program is calibrated to two seconds, as two seconds of constant, uninterrupted contact from two fingers appears to be rare and doesn't leave room in input ambiguity. As soon as the last contact is made the timer resets itself and continues counting upwards, and if there are two recorded contacts when the timer increments to two the OCR is activated and attempts to define the word.

Highlighting & OCR

Every time a gesture occurs, the program takes an image of the workspace and processes the data to find highlights and definition requests. To process the image, we require a high-resolution camera, as the one on the Kinect does not provide enough information to be successfully processed into text using OCR technology. The system, in its most basic sense, works by first capturing an image from webcam, then detecting the document, then detecting the highlighted area, and then finally removing the noise and then processing the image through an OCR engine to obtain a text string.

To get the initial image, we capture a video stream from the Microsoft LifeCam using DirectShow and OpenCV libraries. These libraries are implemented as part of the EmguCV library wrapper that we use for C#. The video stream received from the EmguCV library is a series of RGB images that are continuously updated to the newest frame, which are stored in an EmguCV wrapper for a Bitmap image so we can simply get the newest frame as a Bitmap by calling a function. As the image is a Bitmap, it is lossless and allows for simple execution of filters on it.

The first step is to convert the image into Hue, Saturation and Brightness (HSB) format so it is easier to process for highlights. After this, we split the image into 3 streams; hue, saturation and brightness. We then detect the highlights by detecting running a binary filter to detect a range of +/- 10 from the saturation value of the color we want to detect. We run the same type of filter on the hue stream and get two binary images with the highlight separated from the image. We can also detect the exact color of the highlight by using the hue stream. Now, we run an Erosion filter with 2 passes on both the images to remove any noise that was also detected as the same hue or brightness. We take a union of the binary images to get a more accurate area of where the highlight is. Afterwards, to further improve the image we run a Convolution filter with n as 3. Following the Convolution filter, we run a four pass Dilation filter to remove noise inside the highlighted area if the area was not detected properly. After that, we run a binary filter on the brightness stream so that we only have the text remaining on the brightness stream. When we are left with just the highlights, we remove all the information other than the text from the image by taking its union with the highlight-filtered image so that all the color from the highlight is removed and the OCR engine has an easier time processing the image. We can then run a block detection algorithm to detect the separate highlights and their dimensions. We then crop out the separate highlighted areas that are greater than a certain amount of area. After we have the separated highlights, we send these images to the Tesseract OCR

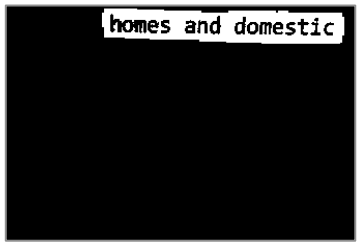

Figure 3: Three words as they appear in the system after extraction.

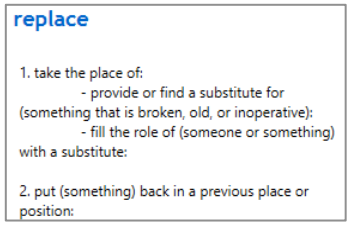

Figure 4: The definition of a word as it would appear when projected.

engine to extract the highlighted text from the original image.

The Tesseract OCR Engine then returns an array of characters and its accuracy confidence level. We simply return the string right now but there is a possibility to run the whole process on another frame until we get acceptable confidence levels. We can then return a Regular Expression instead of just a string so it will be easier to match even if some words are harder to read for the OCR Engine

Dictionary

The Dictionary component performs a search of a keyword on http://oxforddictionaries.com and returns the definition given by the website. We implemented a dictionary over the web because it is easily extendable to searches over Wikipedia, Google, and other user preferred websites. The definition is then projected onto the workspace to the left of the documented being read.

Interface

Our choice in removing any graphical interface for user interaction was also based on Csíkszentmihályi's notion of flow [1], as we did not want the user to break their state of absorption in an activity to interact with the system instead of their work.

PDF Editor

The PDF Editor's task is to modify the digital copy of the PDF to reflect physical modifications of the paper. A user could highlight key sentences, add jpeg annotations, and add text-based annotations to the digital PDF as she works with the physical paper. The PDF may be located in a user's Dropbox folder.

A highlight is added to the digital PDF by passing in a regular expression of a highlighted sentence extracted by the OCR component to the PDF Editor. The editor looks through the entire document and highlights all matching patterns. Image annotation is added by passing an image along with the regular expression of the anchor to the PDF Editor. The image will be included as a file annotation.

Analysis

Environment

Since we were limited by the video resolution of the Kinect camera, a LifeCam was used to read the text. Although the camera was directly above the document, for the system to work we could only work with one page at a time and the font and size of the document were restricted to certain OCR friendly fonts. We experimented using the Consolas font. The set up of the environment is also very time consuming. The Kinect camera was mounted on top of the projector while the LifeCam was hung over the document. This set up is most likely not preferred in a professional setting.

Gestures

The SDK would detect knuckles, wrists, loose wires, tripod legs, papers, and anything else it could get its hands on as touches. Worse, the SDK could be overloaded; in that if some malicious user banged all ten fingers down on the space repeatedly for ten seconds, internal detection issues would occur and the "contact" event would interrupt the "contact removed" event and the program would end up telling the user they were making fifteen contacts on the space when in reality they were making none, because the "contact

removed" event had not been heard for reasons unknown.

The solution to this was the implementation of a background timer that continually counts to five over and over. When a "contact" occurs, the timer records it and resets itself to zero. If the timer reaches five seconds (i.e., it detects nothing for a full five seconds) the contact counter gets reset to zero regardless of it's previous state, making the assumption that the user's hands are not in the picture and that any leftover contacts are erroneous. This turned out to be a very user-friendly solution for ignoring the usual reading and writing gestures that the machine should not detect. If the user wants to make a specific gesture, they only need to remove their hands from the space, count to five, and make the gesture and the machine will read it.

PDF Editor

Since we designed the PDF to locate highlights based on text analysis as opposed to coordinate locations, if the PDF were to have two sentences that were exactly alike, a user highlight of one of those sentences would also highlight the other. This generates a more serious concern: the inability to highlight keywords. Keywords easily generate multiple pattern matches within a given PDF. With our simplistic document parsing, we would not be able to tell which of those matching keywords the user has highlighted. A solution to this problem is to pass in general coordinates of the highlight along with the keyword. However, if there are multiple keywords in general proximity of each other, without precise coordinates, we may still encounter the same problem.

Another issue we encountered was with page matching for documents with multiple pages. How would our PDFEditor know which page the user is working on? This is an important question because if the user highlights a keyword that happens to be in the same location on two pages, our editor would end up highlighting both. One method is to extract the page number located on the corners of the page and pass this information to the PDFEditor. However, since some documents do not have page numbers, this solution is not robust. Another method is to extract the last or first sentence of a given page and determine pages based on the string. These methods were not tested in our system.

Conclusion and Future Work

Read# is currently very sensitive to its environment due to the software used to develop it. A few changes to the system would make it more robust and more powerful than it currently is. We currently use the Interactive Spaces SDK which let us accomplish basic tasks, but for future implementations we are hoping to use 3Gear Systems' API[1] and setup of using two mounted Kinects. Because of the information flowing in from two Kinects, this setup is able to provide millimeter-level accuracy of the hands and fingers. Future implementations should implement straightening algorithms for running the document through the OCR software, since words on a diagonal are difficult to detect. Currently, the document must be perfectly aligned to be recognized by the OCR features. However a straightening algorithm will allow the users to not have to worry about the orientation of the paper and work more naturally. Our system currently only

[1] http://www.threegear.com/technology.html

supports the PDF Editor feature to PDFs that exist in the system library. However we hope to extend this feature so users will be able to scan their own documents into the system by using the cameras to scan them into the system, or find existing documents online. We would also like to support multiple paged documents in future implementations. A future direction for the dictionary component might allow more user interaction with the results. Perhaps a search over Oxforddictionaries.com is not sufficient and the user wishes to see the results for Wikipedia as well. This would be useful if the user is trying to look up idioms that would not be in the Oxford dictionary or words in a foreign language. This would be one way to support translation features in Read# and assist in learning a new language.

Through our work, we have presented Read#, a new system to introduce the advantages of digital documents to the realm of physical documents.

References

[1] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. HarperCollins, 2008.

[2] A. J. Sellen and R. H. R. Harper, *The Myth of the Paperless Office*. MIT Press, 2003.

[3] M. Hong, A. M. Piper, N. Weibel, S. Olberding, and J. Hollan, "Microanalysis of active reading behavior to inform design of interactive desktop workspaces," in *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, New York, NY, USA, 2012, pp. 215–224.

[4] D. Kirsh, "The intelligent use of space," *Artif. Intell.*, vol. 73, no. 1–2, pp. 31–68, Feb. 1995.

[5] N. Weibel, Y, Lui, R. Kanoknukulchai, A. M. Piper, and J. Hollan, "Revisiting the DigitalDesk: A Framework for Interaction On and Above the Desktop."