Video Text Retouch: Retouching Text in Videos with Direct Manipulation

Laurent Denoue, Scott Carter, Matthew Cooper
FX Palo Alto Laboratory
3174 Porter Dr.
Palo Alto, CA, 94304 USA
{denoue, carter, cooper} @fxpal.com

ABSTRACT

Video Text Retouch is a technique for retouching textual content found in many online videos such as screencasts, recorded presentations and many online e-learning videos. Viewed through our special, HTML5-based player, users can edit in real-time the textual content of the video frames, such as correcting typos or inserting new words between existing characters. Edits are overlaid and tracked at the desired position for as long as the original video content remains similar. We describe the interaction techniques, image processing algorithms and give implementation details of the system.

Author Keywords
Video editing, video retouch, direct manipulation, text editing, web-based systems, HTML5 video processing

ACM Classification Keywords
H.5.2 Graphical user interfaces (GUI): Miscellaneous

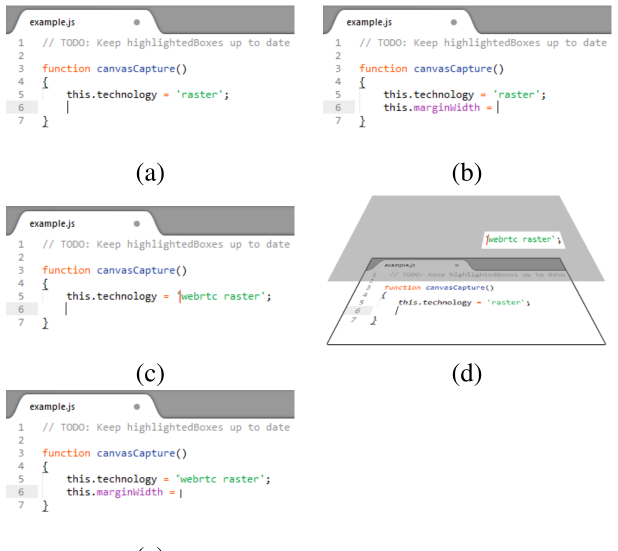
INTRODUCTION

Millions of training videos or recorded meetings and lectures are available online. Unfortunately, when an author or another user needs to correct the content found in these videos, the process can be very time consuming. The whole sequence can be recaptured, or the user needs to edit the video using a standard video editor and retouch it frame by frame, e.g. to delete a word or add new content.

Instead, taking inspiration from past work on retouching static images [1], our system uses direct video manipulation as an interaction technique to more easily retouch the text (and ink) content displayed in these videos.

To edit a line of text, the user can click over that line in the video canvas. The tool automatically pauses the video, identifies the text line using visual content analysis, and allows the user to edit this text line, as shown in Figure 1.

The retouched content is automatically rendered over the original, giving viewers the illusion that the original video has been edited. The system automatically adjusts the edited content over new video frames and removes overlays when new frames appear but the original line is no longer found.



(a)

```
example.js
// TODO: Keep highlightedBoxes up to date

function canvasCapture()
{
    this.technology = 'raster';
    |
}
```

(b)

```
example.js
// TODO: Keep highlightedBoxes up to date

function canvasCapture()
{
    this.technology = 'raster';
    this.marginWidth = |
}
```

(c)

```
example.js
// TODO: Keep highlightedBoxes up to date

function canvasCapture()
{
    this.technology = '|webrtc raster';
    |
}
```

(d)

'webrtc raster';
example.js
// TODO: Keep highlightedBoxes up to date
function canvasCapture()
this.technology = 'raster';

(e)

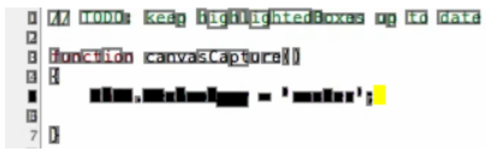
```
example.js
// TODO: Keep highlightedBoxes up to date

function canvasCapture()
{
    this.technology = 'webrtc raster';
    this.marginWidth = |
}
```

Figure 1. An example screencast video showing a code editor. In the original video the author types the first line of the function (a) and continues on to the second line (b). Using Video Text Retouch, the user can click directly on the first line to add "webrtc" before "raster" (c). Note in (c) that the cursor in the original video on the second line is black while the edit cursor on the first line is in red. As the user types, characters are automatically inserted, shifting the rest of the line to the right. Time-synchronized edits initially exist in a layer above the original video (d). Users can hide or show edits using our video player. When shown during playback the edits appear as if they are part of the video (e).

INTERACTION TECHNIQUE

To retouch a video, users simply drag and drop the video file onto the app's web page. The video plays inside an HTML5 video element. When the user clicks somewhere over the video canvas, the video is paused and a text insertion cursor is overlaid, prompting users to start editing the text line as if they were using a conventional text editor. New edits are appended to a time-synchronized layer above the video. Our video player collapses layers so that, to the user, it appears as if edits are part of the original video.



```
// TODO: keep highlightedBoxes up to date

function canvasCapture()
{
    [illegible] = '[illegible]';

}
```

Figure 2. Each video frame is binarized and the bounding boxes of connected components are used to identify text lines (gray boxes); when a user clicks over the video, the text line is detected (black boxes) and its bitmap saved to build the overlay stripe.

The user can use backspace to delete, arrow keys to position, and type any character letter. The system visually adjusts the text line to reflect the changes.

When the user clicks again over the video, the system resumes playback and the newly created overlay is automatically shown over subsequent video frames for as long as the original content underneath the edited line is similar. When the original content becomes different, the overlays are also no longer displayed.

An option in our video player allows viewers to toggle edits on and off so they can compare the original video with the edits. Another option instructs the system to collapse the layers into a single, standard video file for sharing.

BROWSER-BASED IMPLEMENTATION IN HTML5

The system processes the video frames in real-time using lightweight and efficient image processing in JavaScript. The original video is hidden and incoming frames are instead drawn in real-time over a visible CANVAS element, from which we can extract and manipulate pixel data, as well as capture the user's mouse and keyboard events.

When the user clicks over the CANVAS, the video is paused and the current frame is binarized using automatic thresholding; the bounding boxes of connected components are extracted from this binary image, as illustrated in Figure 2 following [2]. Next, the system finds the closest bounding box located to the left of the click position, and identifies the line of text to be edited by looking to the left and right of this first bounding box. A bitmap corresponding to this text line is extracted from the current video frame as a horizontal band spanning the whole frame width, where its height and position are given by the text line's height and vertical position.

The line height is the mean of the bounding box heights, and the text color is determined by the average color of the pixels found in the underlying connected components. These values are used to choose the font size and color of the characters that will be drawn in the bitmap to render any added text.

Each bounding box is assumed to represent a character. When a keyboard event is received and corresponds to a character insertion, the system shifts the adjacent boxes to the right and draws the character glyph over the blank position. Similarly, if the key code is navigational, i.e. backspace/delete/left/right, the system moves the appropriate bounding boxes and updates the position of the cursor. After each keyboard event, the edited version of the text line bitmap is then overlaid on the video canvas, giving users the perfect illusion that the edits are taking place.

PERSISTING EDITS IN REAL-TIME

When the user has finished editing and resumes playing the video, the system continues to analyze incoming frames, and applies the same technique of binarization and connected components extraction.

If boxes under the edited line are similar to any changes text lines, the system continues to show the corresponding overlay. However, two types of conflicts can occur: the original content may move somewhere else on the video canvas, or it has changed significantly.

When the underlying content changes, users may still wish for their edits to persist. Currently, we detect vertical shifts and reposition the overlays accordingly. For example, videos showing a text editor, web page, or word processor often scroll vertically. In those cases, the system automatically moves the overlay to correspond with the shift. However, when the content translates or new content appears, the system simply removes the overlay by default. Better tracking of content changes could be added to the system, e.g. tracking translations would allow the system to reposition the overlays when the video contains a desktop window moving on the screen. But our initial implementation is already very useful for many cases, including mostly static video recordings like lectures where the displayed slides do not change frequently, or even digital ink based tutorials such as Khan Academy videos where the content usually only scrolls vertically. Also, recreating edits later during the video playback is still much faster than using a regular video editing tool.

CONCLUSION AND FUTURE WORK

We presented a novel system that lets users quickly retouch text in videos such as lectures, and online screencasts. Users simply click over the video and start editing the text, reusing familiar metaphor for editing. The system monitors changes in subsequent frames and automatically adjusts the location of the overlays when the content scrolls vertically.

We are developing other tracking algorithms that will reposition the overlays when the content translates (e.g. a window is moved in the video); we are also applying the technique for ink-based content such as Khan Academy tutorials.

We believe that these editing systems will help people rapidly reuse and improve upon existing video content. Ultimately, our goal is to make it as easy to reuse and modify video content as it is to reuse and modify slide decks.

REFERENCES

1. Bagley, S. C. and Kopec, G. E. Editing images of text. *Communications of the ACM*. 37(12). 63-72. 1994.
2. Chang, F., Chen, C-J., Lu, C-J. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*. 93(2). 206-220. 2004.