Content-based Copy and Paste from Video Documents

Laurent Denoue
FX Palo Alto Laboratory
3174 Porter Dr.
Palo Alto, CA, 94304 USA
denoue@fxpal.com

Scott Carter
FX Palo Alto Laboratory
3174 Porter Dr.
Palo Alto, CA, 94304 USA
carter@fxpal.com

Matthew Cooper
FX Palo Alto Laboratory
3174 Porter Dr.
Palo Alto, CA, 94304 USA
cooper@fxpal.com

ABSTRACT

Unlike text, copying and pasting parts of video documents is challenging. Yet, the abundance of video documents now available including how-to tutorials requires simpler tools that allow users to easily copy and paste fragments of video materials into new documents. We describe new direct video manipulation techniques enabling users to quickly copy and paste content from video documents into a user's own multimedia document. While the video plays, users interact with the video canvas to select text regions, scrollable regions, slide sequences built up across many frames, or semantically meaningful regions such as dialog boxes. Instead of relying on the timeline to accurately select sub-parts of the video document, users navigate using familiar selection techniques such as mouse-wheel to scroll back and forward over a video shot in which the content scrolls, double-clicks over rectangular regions to select them, or clicks and drags over textual regions of the video canvas to select them. We describe the video processing techniques that run in real-time in modern web browsers using HTML5 and JavaScript; and show how they help users quickly copy and paste video fragments into new documents, allowing them to efficiently reuse video documents for authoring or note-taking.

Categories and Subject Descriptors

H.5.1 Multimedia Information Systems; I.7.5 [Document and Text Processing]: Document Capture - document analysis; H.5.2 [Information Interfaces and Presentation]: User Interfaces

General Terms

Algorithms, Human Factors.

Keywords

Video Document Structure and Analysis; User interaction for content reuse; Real-time Video Document Processing; Document Authoring Tools

1. INTRODUCTION

Every day, millions of users watch How-To video documents such as screencasts to educate themselves about programming, design, using new software or configuring computer systems, using a particular web site, listening to lectures, etc. Unlike text documents though, reusing content from video documents is hard. Few users would even take a simple screenshot of a video frame because it involves too much work: users need to pause the video, take a screenshot, crop it, and paste the image into their favorite document editor.

Inspired by work on direct video manipulation such as [1] and [2], we developed a set of techniques where users directly interact with video content using familiar techniques such as dragging a selection box over an area to highlight text, mouse-wheel to scroll up and down, or double-click to identify rectangular areas of importance, and show how they can be extended to accommodate the temporal nature of video documents. We show how to adapt existing document processing techniques and create novel ones to make it easy for users to copy and paste parts of video documents.

2. MOTIVATIONAL SCENARIOS

To motivate this vision, we describe three scenarios based on our experience observing users manipulate video documents.

2.1 Copying text from a video

Many tutorial screencast videos primarily show the presenter typing computer code into a text editor. Ideally, the viewer would like to pause the video when the line is completely typed or the function finished in order to review the code. However, using the timeline to accurately position the video time can be challenging. Our goal is instead to allow the user to click and drag over the video canvas where the line is being typed; quickly drag her selection toward the right to reveal the next video frames and finally lift when the line appears complete. She can then paste that part of the video canvas into her personal notes document as an image, request that the image be converted to text via OCR, or paste the section as a sequence of frames encapsulated in an animated GIF showing how the text was originally entered.

2.2 Copying a scrolled region

Video screencasts depicting web site creation or use often involve the presenter scrolling up or down within a browser or text editor to reveal new content. To document this particular interaction a user needs to 1) carefully position the video when the page starts scrolling; 2) run her favorite screen recorder (possibly identifying what region to capture); 3) play the video until the scroll action is complete; 4) quickly stop the screen capture recorder, and 5) finally paste the video file into her favorite document editor (which may be a complicated process in and of itself). Our goal is to make this process easier by allowing the user to move the mouse-wheel up and down over the video canvas to indicate in a single action the beginning, end and region of interest. Our system can then automatically generate an animated GIF of that video segment and region, ready to be pasted into a document editor.

2.3 Copying a region of interest

Consider a user watching a tutorial explaining how to setup a project in XCode who wants to copy and paste a fragment of the video frame showing how to configure a particular dialog box. Currently the user would need to stop the video, capture a picture of the screen, paste it into a photo editor, select the region of interest, and then copy that to a document editor. Our goal is to allow her to directly double-click inside the dialog box and have the system automatically record and crop the frames while this dialog box is visible in the video stream, while again making content available for pasting into a document editor.

Below, we describe a system we implemented that exemplifies our vision. We first present the architecture that lets us process users' actions over a video canvas using HTML5 and JavaScript. We then describe how the system maps traditional mouse or touch-based interactions over the video canvas to selection actions, and present the accompanying document processing techniques used to find text, and identify scrolled and rectangular areas in the video. Finally we describe our multimedia document editor.

3. DETECTION AND SELECTION

The core of the system runs in a web browser. Video is embedded as an HTML5 video and JavaScript paints incoming frames in a CANVAS element. Pixel data is manipulated in JavaScript to perform real-time video and document processing such as binarization, frame differencing, line-detection, connected-components analysis and scroll detection across frames.

In order to process video frames from a VIDEO tag inside our web page, the original (currently YouTube hosted) video is served through a proxy server written in NodeJS. The proxy reads video content from YouTube and streams it back to the client's browser, thus solving the same-domain policy constraint.

A timer is used to continuously draw incoming frames onto a CANVAS element that the user sees. JavaScript event listeners are attached to this element and trigger the different actions: double-click triggers region detection, mouse-wheel triggers scroll-analysis, and click and drag triggers text detection. We now describe useful direct video manipulation techniques and how they are used to quickly copy and paste parts of video documents.

3.1 Detecting regions in the video document

Incoming frames are first binarized using a simplified edge detector that finds vertical and horizontal edges in the video frame: it reports 1 if the gray scale value of adjacent (horizontally or vertically) pixels is greater than 32 and 0 otherwise (see Figure 1). We then find edges by counting lines and columns where 1 is found more than 40 times in a row or column.

When the user double clicks on the canvas showing the video frames, the system finds the smallest encompassing rectangle using the horizontal and vertical lines found above. If a rectangle is found, the system draws a blue rectangle to show users their current "selection". Because there might be several candidate rectangles, subsequent double-clicks prompt the system to show the next rectangle outside the current one. When the last rectangle is shown, the cycle repeats.

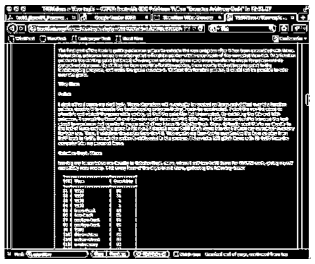


Figure 1. Video frames are binarized using a fast edge detector; vertical and horizontal lines are found by counting series of white pixels; when the user clicks, the system can find rectangular areas around the click position.

At this point, if the user initiates a Copy command using the keyboard or other means, the system automatically generates a PNG image out of the cropped area from the video canvas and copies it into the clipboard. If instead the user starts a left or right drag operation over the canvas, the system draws previous or next frames, depending on the drag direction. When the rectangular region is no longer detected at the same location, the system stops showing previous or next frames. At this point, if the user issues a Copy action, the system generates an animated GIF of all the frames found between the start and end time in the video sequence where that rectangular region was shown.

3.2 Detecting scrolling areas

In order to detect scrolled frame regions, we compute the difference between the binarized versions of the previous and current frames. Vertical and horizontal projection profiles are then computed to find the changed region. The best correlation is computed by finding the minimum difference of vertically shifted pairs of pixel columns, as shown on the bottom-right of Figure 2. If most columns agree on a similar vertical shift, we use the average of the votes as the scroll value, and store the frame in an array along with the current video playback time and scroll value. We currently do not compute horizontal scrolls because they are not common in screencasts, which typically depict computer content that scrolls vertically.

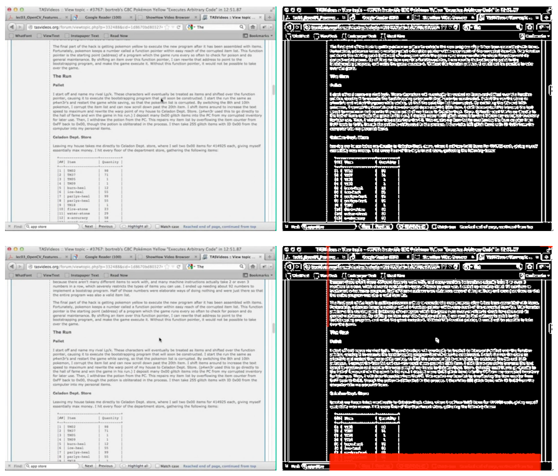


Figure 2. Left column shows 2 frames and their binary version on the right; Bottom-right: red overlay shows the amount of scroll found between the 2 binary images.

This computation is always performed, but frames and their scroll values are discarded after 5 seconds in order to minimize the memory footprint.

When the user initiates a mouse-wheel action over the video canvas when scrolled frames have been detected, the system overlays a scroll indicator to tell the user she is now driving the video using her mouse-wheel. At this point, mouse-wheel events indicate what frames to draw over the video canvas: mouse-wheel up shows previous frames and mouse-wheel down shows next frames. This method allows the user to quickly replay a passage of the video where content is scrolling vertically.

The system stops rewinding when the scroll value reaches 0 in either direction. Note that when the user started to mouse-wheel, the video kept playing in the background and frames were still being captured and processed to determine their scroll values. If the user's mouse-wheel event catches up with the real-time and frames are still being scrolled, the system keeps showing them to the user, and stops once no scrolling is found for at least 1 second.

When the user clicks over the canvas, the system records the time as either the beginning or end of the upcoming Copy action. Upon receipt of a Copy command, the system generates an animated GIF with the collection of frames between the start and end times. If the user has not clicked, the range is defined as the oldest and newest times reached by the user with mouse-wheel events.

3.3 Selecting text from video

When the user clicks over the video canvas, the system binarizes the current video frame, computes its connected components and their bounding boxes using [3] and defines the initial selection box rectangle (see Figure 3).

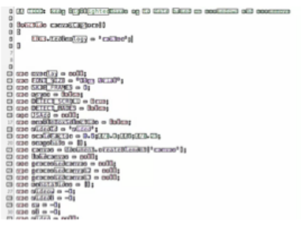
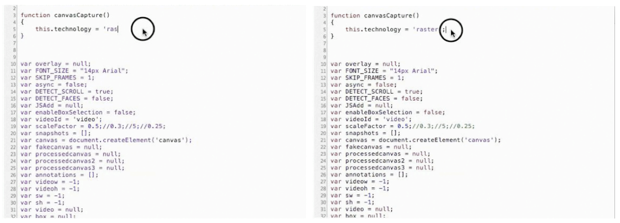


Figure 3. When the user starts a drag operation, the bounding boxes of connected components are computed

The system visually highlights the bounding boxes under this rectangle, mimicking what happens when a user selects text in a regular text-based widget. When the user initiates a drag to expand her selection box, the system updates the corresponding boxes underneath.

As before, the video is still playing in the background: the system accumulates incoming frames and performs connected components computations. Meanwhile, the user is shown the last frame corresponding to connected-components that match her selection box. This frame is updated whenever the selection changes and overlaps connected components in other frames. When found, the system draws that frame over the main canvas, giving the illusion that the video seeks to reveal more text (Figure 4). Similarly, if the user reduces the selection box to the left, the system finds previous frames that have connected components under the current selection but none to its right; again, this gives the illusion for the user that she navigates back into the video.



```

function canvasCapture()
{
    this.technology = 'ras
}



var overlay = null;
var FONT_SIZE = "14px Arial";
var SKIP_FRAMES = 1;
var async = false;
var DETECT_SCROLL = true;
var DETECT_FACES = false;
var JSAdd = null;
var enableBoxSelection = false;
var videoId = 'video';
var scaleFactor = 0.5;//0.3;//5;//0.25;
var snapshots = [];
var canvas = document.createElement('canvas');
var fakecanvas = null;
var processedcanvas = null;
var processedcanvas2 = null;
var processedcanvas3 = null;
var annotations = [];
var videow = -1;
var videoh = -1;
var sw = -1;
var sh = -1;
var video = null;
var box = null;
```

```

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



var overlay = null;
var FONT_SIZE = "14px Arial";
var SKIP_FRAMES = 1;
var async = false;
var DETECT_SCROLL = true;
var DETECT_FACES = false;
var JSAdd = null;
var enableBoxSelection = false;
var videoId = 'video';
var scaleFactor = 0.5;//0.3;//5;//0.25;
var snapshots = [];
var canvas = document.createElement('canvas');
var fakecanvas = null;
var processedcanvas = null;
var processedcanvas2 = null;
var processedcanvas3 = null;
var annotations = [];
var videow = -1;
var videoh = -1;
var sw = -1;
var sh = -1;
var video = null;
var box = null;
```

Figure 4. Text selection from video: as the user's selection extends to the right, the system displays future frames where boxes of connected-component are found under the user's selection box, allowing her to finely adjust the selection to copy into a new document.

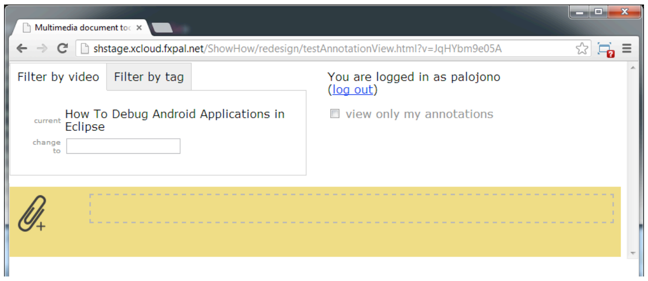
When the user lifts the mouse pointer, the system copies the currently shown frame as an image and also generates an animated GIF of the frames accumulated from the left-most to the right-most selection points. When the user pastes this data, she can choose what flavor to keep.

3.4 Selecting text across many frames

In many conference or lecture-type video recordings, the presenter shows a deck of slides, either shown intermittently (presenter/slide/presenter/slide) or always in the background of the room. Based on previous work [4], video slides are automatically detected and indexed on the server. When a user initiates a text selection over a slide and extends her selection box, the system is able to seek the video past the non-slide frames and get frames of the new slide. This allows users to select text from slides that are built-up across longer spans of time. For example, the presenter shows a slide with one header, talks for 30 seconds, then shows the same slide with a new sub-header. Our user can select the first line when the slide is first shown, then extend her selection box below and automatically have the system "jump" to the time when the second slide is shown with the second line underneath. When the user lifts the mouse pointer, the system only generates an animation of the slides found in between.

4. MULTIMEDIA DOCUMENT EDITOR

In our system, content extracted from video documents is placed on the clipboard in multiple data formats to support graceful degradation across applications with different levels of multimedia support. For basic document editors, such as text and photo editors, we paste content in 'text/plain', 'image/jpeg', 'audio/mp3', etc. format as appropriate. For editors with slightly more support we paste content as 'text/html' to support more advanced interaction, such as links from the pasted content back to the source video. Finally, we also paste content onto the clipboard using a manually defined format that our HTML5-based multimedia document editing tool (Figures 5, 6, 7) can parse.

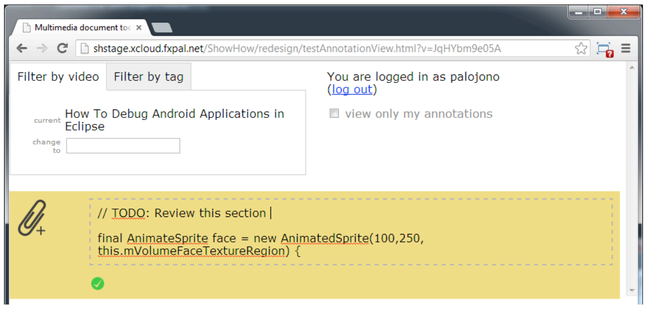


Multimedia document to
shstage.xcloud.fxpal.net/ShowHow/redesign/testAnnotationView.html?v=JqHYbm9e05A

You are logged in as palojono
(log out)

current How To Debug Android Applications in Eclipse
change to

Figure 5. Our multimedia document editor.



shstage.xcloud.fxpal.net/ShowHow/redesign/testAnnotationView.html?v=JqHYbm9e05A

You are logged in as palojono (log out)

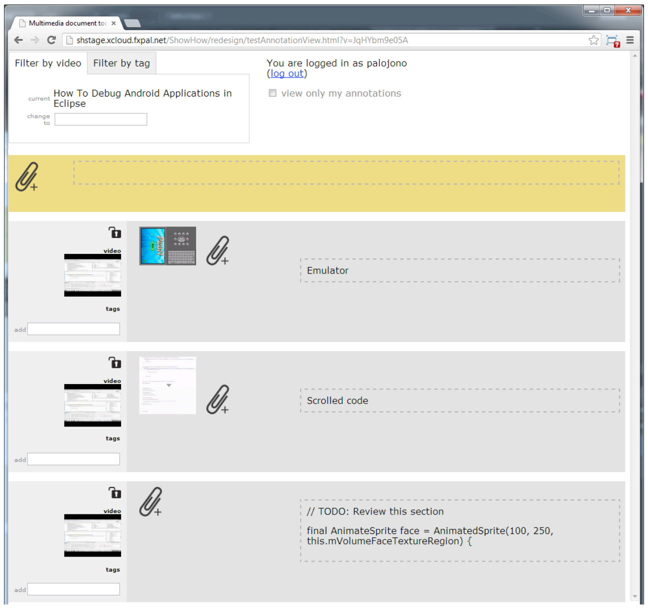
How To Debug Android Applications in Eclipse

```
// TODO: Review this section

final AnimateSprite face = new AnimatedSprite(100,250,
this.mVolumeFaceTextureRegion) {
```

Figure 6. Users type or paste content into the yellow textboxes to begin creating a new annotation.

Figure 6 shows how users can create a new annotation within our multimedia document editor by pasting selected text that was copied from a video document. OCR tools are then used to convert the image (video frame) text to the pasted content. If the text conversion does not rise above a pre-set confidence threshold the content is instead pasted in as an image.



shstage.xcloud.fxpal.net/ShowHow/redesign/testAnnotationView.html?v=JqHYbm9e05A

You are logged in as palojono (log out)

How To Debug Android Applications in Eclipse

Emulator

Scrolled code

```
// TODO: Review this section

final AnimateSprite face = AnimatedSprite(100, 250,
this.mVolumeFaceTextureRegion) {
```

Figure 7. A view of the multimedia document from Figure 6 after using our tools to copy-and-paste a region from a video (top), a compilation of frames from a scrolled section of a video (middle), and the previously selected text (bottom) .

Figure 7 shows annotations extracted using the selection techniques described in Section 3 including a video region, a compilation of frames from a scrolled section of a video, and text. Each gray area is a separate, "annotation" containing any number of media attachments and text. The lighter gray area in the left portion of each annotation contains meta-data including the video with which the annotation is associated as well as tags and privacy settings. All annotation media are linked back to their source videos (if available). Users can drag-and-drop media onto the paperclip icons or use the keyboard to paste content. Media can originate from other webpages, the desktop, or from the tools we developed. So that users can organize their notes flexibly, all elements are immediately editable, and users can drag media between annotations as well as reorder the annotations.

5. FUTURE WORK

One can imagine interesting extensions to this work, such as tracking windows being moved over the video (perhaps using techniques presented in [2]) and navigating the video timeline accordingly, or listening to keyboard events such as backspace or cursor arrows to rewind or fast-forward the video to corresponding text elements.

Desktop widgets are another common type of content seen in screencasts, for example a series of drop down menus in a PhotoShop tutorial. It is conceivable that the user watching the video would want to click on the menu over the video canvas and have the system automatically skip to frames where sub-menus are showing, allowing the user to quickly copy and paste the entire sequence of actions into the new document.

Also, many recorded lectures feature the presenter's slides warped in a non-rectangular shape; we could extend our current rectangular region detection to accommodate for these geometries and let users paste a rectified rectangular portion of the slides.

6. CONCLUSION

With millions of people trying to learn every day from video documents, it is important to develop tools that allow them to quickly copy and paste fragments of these documents into new documents, like it is possible today with text-based documents.

We presented initial steps in that direction for certain kinds of video documents and proposed techniques to directly manipulate video content. Users can 1) easily scroll up and down over a video to automatically rewind or fast-forward the video, 2) easily select text content from video frames, including automatic skipping of frames until the desired text segment is shown, and 3) easily identify rectangular areas of the video canvas that correspond to commonly seen widgets such as windows and dialog boxes. We described an implementation of these techniques that runs in real-time inside a modern web browser using HTML5 video, CANVAS element and JavaScript.

7. ACKNOWLEDGMENTS

Our thanks to FXPAL management, especially Larry Rowe and Lynn Wilcox for supporting this research.

8. REFERENCES

[1] Dragicevic, P., Ramos, G., Bibliowitcz, J., Nowrouzezahrai, D., Balakrishnan, R., and Singh, K. 2008. Video Browsing by Direct Manipulation. In Proceedings of ACM CHI 2008. 237-246.

[2] Goldman , D.B, Gonterman, C., Curless, B., Salesin, D., Seitz, S.M. 2008. Video object annotation, navigation, and composition. In Proceedings of ACM UIST 2008. 3-12.

[3] Chang, F., Chen, C-J., and Lu, C-J. A linear-time component-labeling algorithm using contour tracing technique. Computer Vision and Image Understanding, 93(2). 2004. 206-220.

[4] Adcock, J., Cooper, M., Denoue, L., Pirsiavash, H., and Rowe, L.A. TalkMiner: A Lecture Webcast Search Engine. In Proceedings of ACM Multimedia 2010. 241-250.