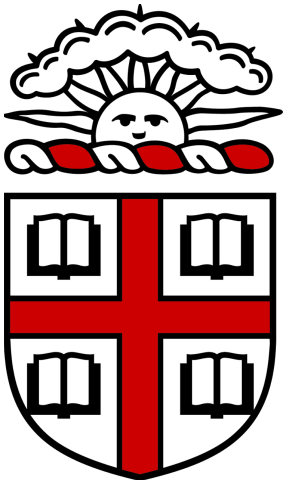

BROWN

A web application for splicing online videos with annotations

Tan “Charles” Zhang
Department of Computer Science
Brown University
Master’s Project Report

1. Introduction

As of the writing of this paper, approximately 90% of Internet traffic comprises of the delivery of online video content. Youtube is the largest video sharing site and every minute there are 72 hours of video uploaded, over 4 billion hours of videos are watched each month on Youtube. It is easy to see that there are tons of video content on the Internet and people are spending a lot of time online watching streamed videos. Traditional ways of grouping multiple videos together is by way of a playlist which is simply a list of videos that play one after another. It is good for grouping together multiple videos of interest and works fine for this simple purpose. But what if you want to do more than just putting those videos in a list? What if you are not interested in the whole video but only a section of the video? What if you want to show something interesting about this video and would like some visual display on top of the video? What if you want to create a new video that consists of existing video clips? The application presented in this paper solves just those problems.

The content of this paper consists of the following. I first will give a descriptive summary about the current video streaming technologies, the pros and cons of these technologies and where they are used, as well as their browser compatibility. Then I will move on to give a detailed illustrated description about the UI design of the video splicing application, the usage scenario. And finally I will conclude my work and talk about the future work.

2. Technologies

2.1 video streaming technology overview

Back in the early days when the Internet had just started, videos were delivered by first downloading the full video file and the playing it. There was no video streaming technology. What that means is that basically if you want to watch a video online, you have to use a file transfer software to download the video, wait for the transfer to be completed, and then use a video playing software to open the video, and you couldn't watch the video while it's been downloaded. And then came the video streaming technology. Real Time Streaming Protocol (RTSP) is used in conjunction with Real Time Transport Protocol (RTP) and Real Time Control Protocol (RTCP) to stream the video content and control the video playback. With this technology, video content is buffered in memory and no file is saved on disk. There are a few shortcomings for this type of video streaming technologies which makes the user experience a little bit unpleasant for the work in the paper. To further explain these shortcomings, allow me to take a short detour and talk a little bit about the videos in question itself.

A video is essentially a series of images displayed in succession with very small time intervals along with audio tracks. To store the video, you need to store all the images within the video. But that would take up a lot of storage space. Consider a video of quality 640 * 480 (the equivalent of the 480 quality video on youtube), say each image is worth 0.5 MB and the video frame rate is 20 FPS. For a 5 minutes video, that's 6000 frames, and takes approximately 3GBs to store the

video. So video compression technologies are used to compressed the video down to a much smaller size without losing much fidelity to the original video. How the video compression technologies work is that instead of storing each individual frames, they will only store the difference between frames. Most videos don't actually change that much from frame to frame, so this will achieve a much compression rate, resulting in a much much smaller file size. Now I will introduce the concept of a key frame. Whenever a drastic change to the image occurs, such as switching from one scene to another, a key frame is used to store the complete image of the new scene instead of only storing the delta from the previous image, because most of the pixels will be different in the new image. To put it in a more technical way, whenever the visual difference between two frames is so great that representing the new image incrementally from the previous frame would be more complex and would require even more bits than just storing the whole image. In action, the server will stream each frame to the client and the client will decode the images. So when you are streaming a video in this way and you do a seek to a part of the video that has not been buffered, the streaming client will issue a command to make the server stream video from that time point. Since the key frames are scattered so sparsely, chances are you won't seek to a key frame and the first frame is an incremental change, which will result in the streaming client to interpret the new frame using the new delta and the image before the seek. So what you see is as if the video stopped on one frame and starts to smudge until you receive the next key frame.

This problem could be solve by using another media streaming technology called progressive download, or pseudo-streaming. The major difference between this technology and the aforementioned video streaming technology is that the previous one has its own streaming, transmission and control protocol, whereas this technology relies on HTTP protocol. What it does is basically breaks down the original video into little video pieces each only a couple seconds long, and transfer those files to the client via HTTP. Each piece of small video files will certainly start with a key frame, and when the client does a seek to a part that has not been buffered, the server will transfer the entire chunk of video file to the client, and the client will be able to know which key frame to interpret from. One other benefit of this type of video streaming technology is that it facilitates adaptive bit rate. Client could run a simple program to monitor the network condition and let the server know of this information. Server can then choose to transfer video segments of higher or lower quality depending on the client's network condition. Of course, this will require the server to have multiple version of the same video segment ready, but it's a reasonable price to pay. Since the server is essentially transferring small static files, those video segments could be cached along the way and the content distributor could even use CDN (Content delivery network, a network of caches) so that the user can have a faster and more pleasant video watching experience. Youtube's desktop site is currently using this technology to stream videos.

2.2 The HTML5 video

The HTML standard version 5 has included the video element with the intention to allow videos to be shown on the web without using any plugins. It sounds very exciting, but the reality is, the support for the html5 video element is still evolving, which is a polite way of saying it doesn't work yet. The progression is mainly hampered by lack of agreement as to which video formats should be supported. Currently, the HTML5 video element only support the type of videos that can be downloaded via HTTP, that is to say it does not support any type of streaming protocols, so youtube videos or RTSP videos cannot be played. I experimented with the HTML5 video element a little bit and it has a nice API and you could tell the people behind it has good intentions. So I really hope that they could have an agreed upon standard soon to make HTML5 video a true reality, but in this work, HTML5 video element is not used.

2.3 The Plugins

In this work, I used two plugins. One other plugin has great potential but unfortunately is incompatible on mac computers so it cannot be used here. I will give more information later in this section.

Since youtube is the biggest online video sharing site, it is within reason that we have a youtube plugin to stream youtube videos. Youtube has a nice player API as well as data API for you to control the player and get related video information. Youtube used Adobe Flash video. All major browsers come installed with flash player so the user don't have to install it themselves. The video streaming is really fast and you could pretty much achieve 'seamless' transition between two videos since it only takes less than one second to switch from one video to the next.

A lot of other videos are served using streaming protocols such as RTSP, so I need to find another more general plugin to play this type of videos. There aren't so many freely available plugins, the ones that I ended up experimenting with are Apple's quicktime browser plugin and VLC browser plugin. Both of them have platform compatibility issues. Quicktime's API doesn't work so well on Linux, some functions simply does not work, and VLC browser plugin does not work on mac computers at all. I chose to go with quicktime plugin for the time being because in the targeted audience at the moment there are much more users using mac os than there are using linux. But in the future if one wish to target a wider range of audience, they could show the user different media player plugins for different operating systems.

3. the UI

3.1 Goal

The goal here is to provide users with basic video splicing/editing functionalities such as choosing the start and end position for video clips, changing the ordering of the video clips, adding/removing annotations, adding/removing video clips, and also the playback of the videos as if they were one single video with annotations.

One possible scenario where this could be useful is that students would be able to easily write multimedia essays using video clips and annotations. Also for instructional videos where the content of the video consists mainly of texts, one could exploit the virtual timeline to introduce paused time into the whole timeline of the video which would allow one to have only a very short video but turning it into a fairly long video leaving enough time on each instructional frame for people to reach through the content.

The following content of this section gives a detailed description of the UI from the perspective of a potential user.

3.2 Detailed description

3.2.1 At a glance

Figure 1 shows a screenshot of the application. The basic view is very simple and clean. There weren't a lot of visual design, all UI elements serve their basic functional purpose.

In the middle top of the screen you could see that there is the current video. Below the video player there are a couple of buttons. On the left are two playback buttons. The play button will change to a pause button once it is clicked and the video starts playing. On the right there is a button for adding new annotations. You can read more on annotations in section 3.2.2.

Below the player are two sliders. The one with two slider handles is for choosing the span of the current video to show up in the final composite video. And the longer slider with only one handle and other markings is the timeline of the composite video. Read more about them in the next section.

At the bottom there is a panel with video thumbnails of the videos. The one with orange border indicates that this video is the one currently being played. Each of the thumbnail is draggable so you could reorder the videos by drag and drop operation. By clicking on one of the thumbnails, the corresponding video will be loaded and seeked to the starting position. How long each video

lasts is not indicated in this panel.

3.2.2 The sliders and the timeline

The shorter slider is called the ranger selector. The total length of this slider corresponds to the length of the video and the length of the orange section in between the two sliders is in proportion to the length of the video clip. You can adjust the starting and ending position of the video clip by dragging these slider handles. When you do drag one of the handles, the player will seek to the position the handle is positioned at in order to show you the selection being made. All of the slider handles in this application can receive input focus by clicking on them. Once the handle is focused (you can tell by the dashed border line), you can use the left and right arrow key on the keyboard to fine tune the selection. Each keystroke correspond to a delta of 0.1 seconds, which provides a fine granularity.

The longer slider with only one handle and other markings is called the timeline slider. Figure 3 shows part of this slider in isolation. The length of this slider corresponds to the length of all the video clips combined. The slider handle indicates the progress of the playback. You could also drag the slider to seek to a different part of the video and also switch to a different video.

There are several markings on this timeline slider. The first kind of marking is the short vertical orange line which indicates the boundaries of a video. The horizontal orange line indicates the span of the current video. The markings with gray color are for annotations. They indicate the starting and ending position and the duration for each annotation and they serve as the input for changing the starting and ending positions of the annotation. Each annotation's timeline marking has three parts, the vertical gray bar indicating the starting position of the annotation, the round circle indicating the end position, and the horizontal span indicating the duration of the annotation. The two end marks can be interacted with. When you move the cursor over the start/end markings, they will scale up a little as a visual feedback. If you click on them, the video player will seek to that position. But if you want to change the starting/ending position of the annotation, you need to press the mouse button for one second on either one of the markings until it turns red, indicating that it is not draggable. Now you can drag the marking to change the annotation's start and end position. After you release the mouse button, the marking will turn to color orange, indicating that it has now received input focus, which means you could fine tune the position of that marking with the left and right arrow keys. Moving the start position of the annotation will automatically change the ending position of the annotation as long as the ending position is within the boundary of the initial video (not the video clip). If the annotation's end point goes beyond the end position of the video clip, the annotation will not continue to be displayed once the video switched to the next one, but the actual duration of the annotation remains unchanged. This will be manifested when you extend the end position of the video clip. It is provides a more pleasant editing experience when you are selecting the ending position of a video with existing annotation which the end position happens to sit upon. This saves the user the trouble of having to adjust the ending position of the annotation everytime he/she makes an adjustment to the ending position of the video clip. Moving the end position of the annotation will not change its starting position, thus its a way of changing the duration of the annotation.

3.2.3 Annotations

If you want to add annotations, you simply need to click on the annotate button. After you do that, you move your cursor onto the the player and the cursor pointer will turn to a cross, indicating that you need to select a rectangular region for the annotation to show up like shown in figure 2. And after the region has been selected, you can drag around the region to reposition the annotation area, change the size of the region with the adjustment corner at the lower right side. Double click on the region to input annotation content. Right now the annotation content only support text. Support for other types of content could be easily added since the display of the content is done by adding new HTML elements. It is not added at the moment for lack of ways to store the annotation content since it is not connected with a database yet.

At the bottom on the right side of the annotation region you could find two buttons for either confirming the addition of the annotation of the cancellation of the change which will remove the annotation. Once you are done with editing the annotation and click on the check mark at the bottom, the annotation will be added to the video document and the markings for this new annotation will show up in the timeline slider.

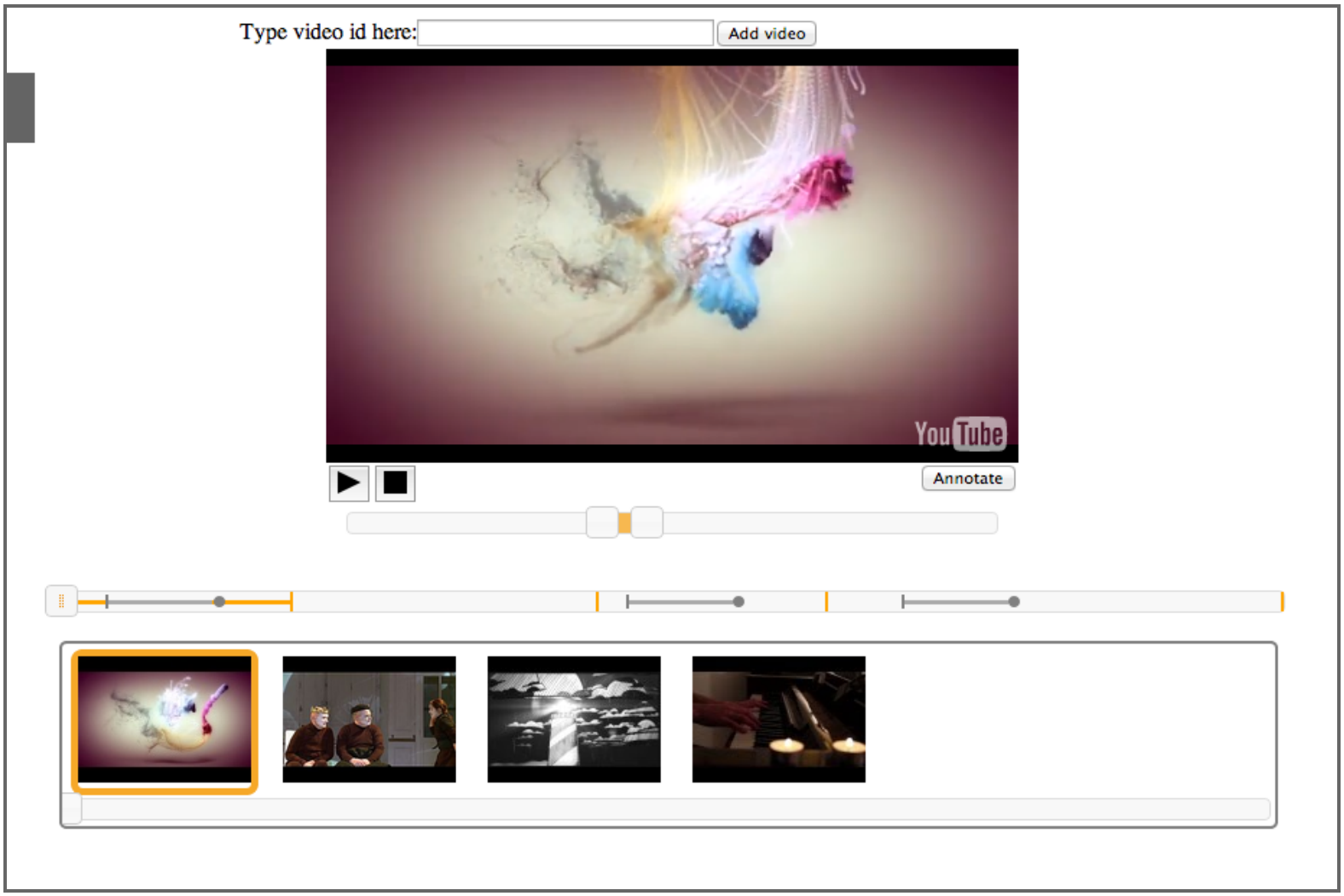

Figure 1 Application Screenshot

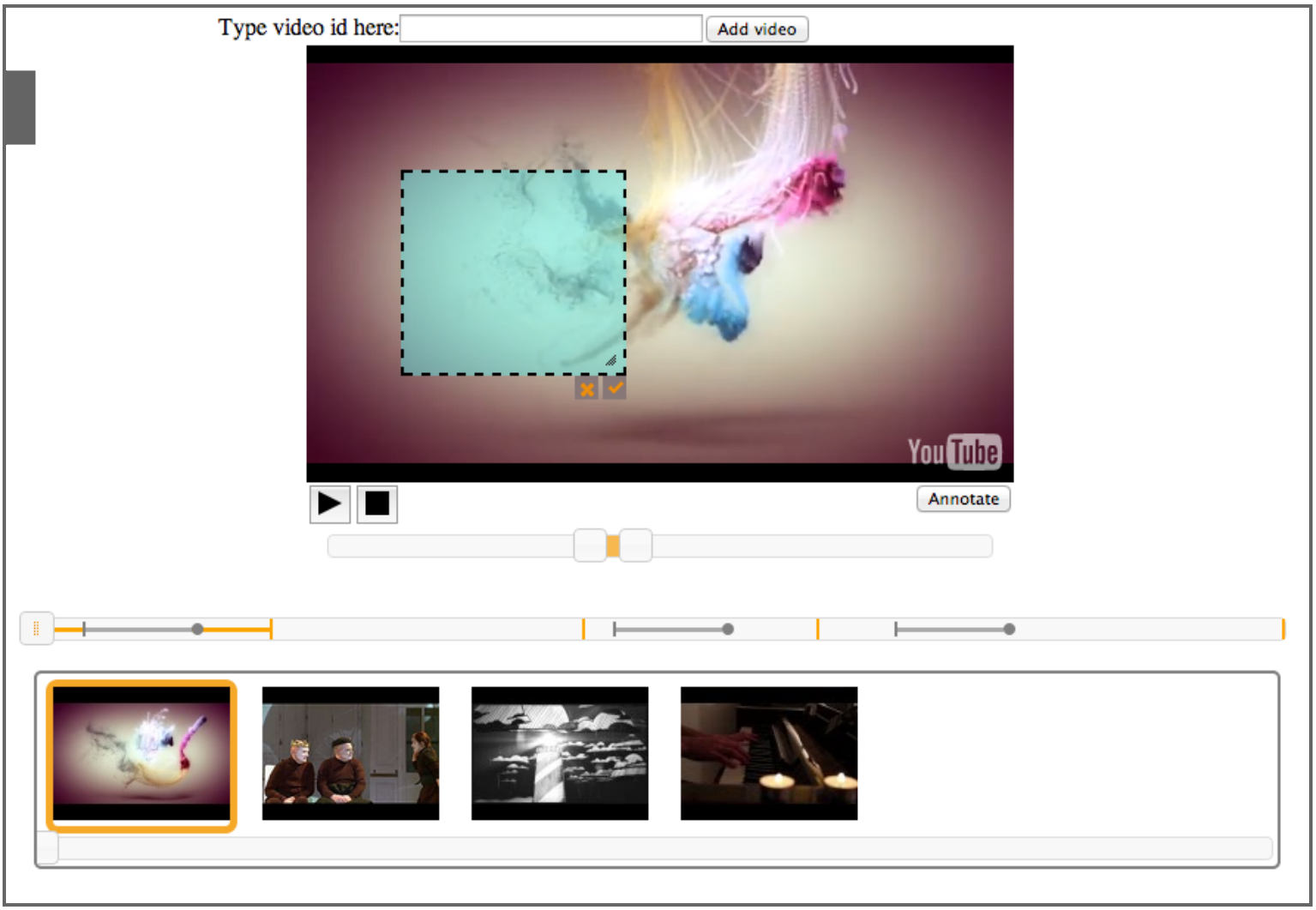

Figure 2. Selecting annotation region

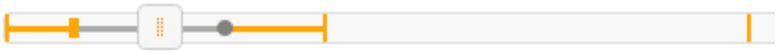

Figure 3. Timeline slider with a focused annotation marking

3.2.4 Adding new youtube videos

Youtube has a nice data API that exposes their search functionality for videos, so user could search for videos of interested within the app without having to leave the application and go to youtube to get the video's id or url. There is a gray bar at the left side of the application which if you click on it will slide out a search panel where you could search for videos like you do on youtube website. Figure 4 shows a screenshot of this panel. As you can see you can add video from the search result with a single click.

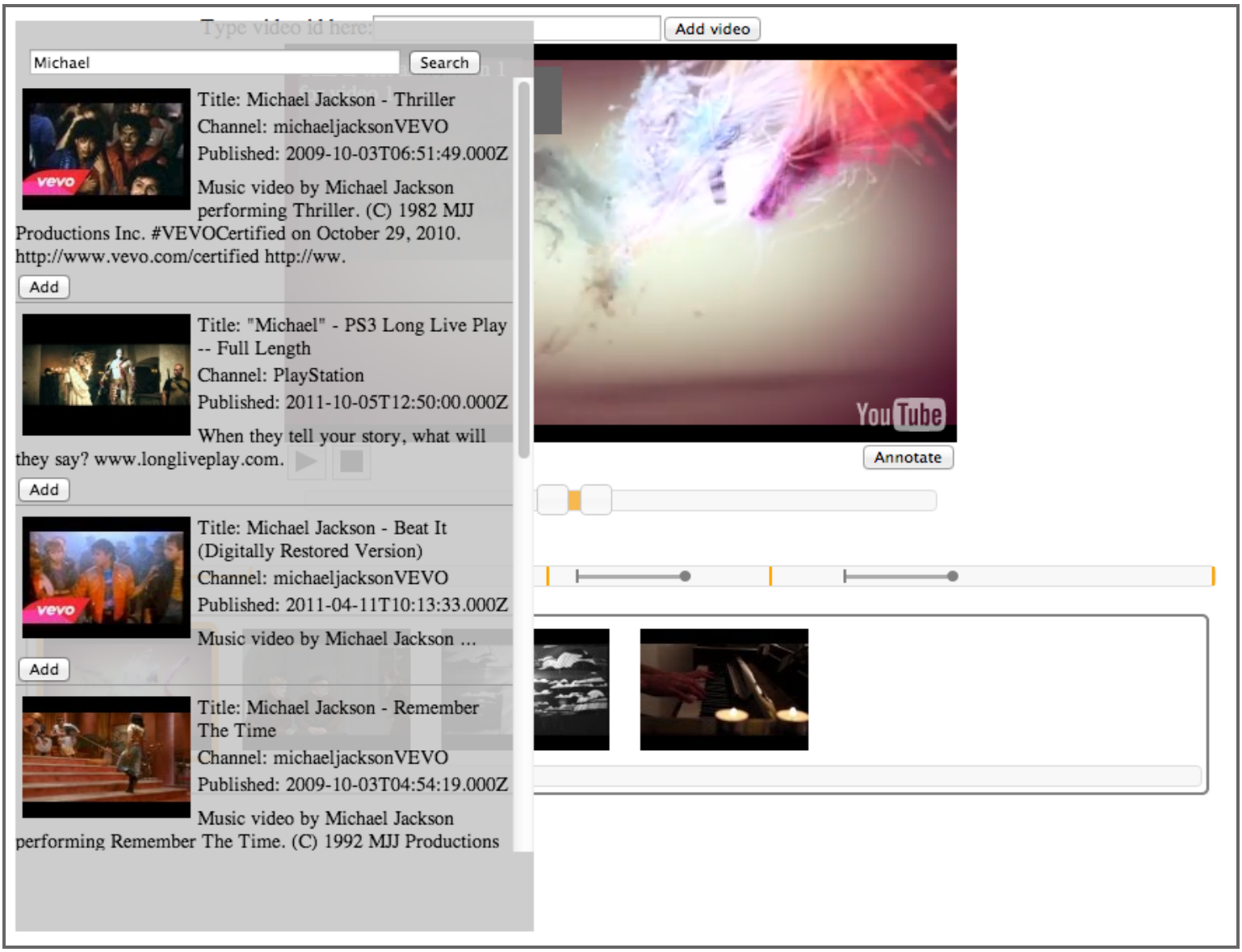

Figure 4. Search panel for youtube videos

After you hit add, the application for make calls to the youtube API for the video's length, url, thumbnail, etc and add all the necessary information to the video document.
The application also supports videos served from a RTSP server. There is no UI support for

adding such videos yet mainly because there is no support for querying existing videos of the kind. In the test case one such video is added in a hard coded way. In reality, if one wishes to provide a pool of such videos for other people to choose from, one way of doing it is to provide APIs for querying which will takes in search terms and returns relevant information, and also support the inspection of video parameters such as length of video, video thumbnail, etc.

4 Implementation

In this section I will give an overview of the implementation of the application. The application is written as a jQuery plugin which expose API functions such as load videos, play and pause video playback, etc. Basically if one wants to embed the application inside their webpage, all there is to do is include the latest jQuery library if you don't already have it on your webpage, include the youtube player's swfobject library, and quicktime player's wrapper javascript script, also the library for jQuery UI, then place a div element on the page where you want the application to be displayed and grab that element from jQuery and make the API call function on that element.

This application is also integrated as a special document type into Worktop, a digital desktop application for the humanities. Worktop is built upon microsoft's WPF(Windows Presentation Foundation) and it has support for rendering web pages and making console function calls on the web pages that it renders. So the application exports global functions as interface sto Worktop to allow for the integration.

As mentioned earlier in the video streaming technology section, the streaming videos does not provide a very pleasant playback experience when you seek ahead of time. In quicktime player browser plugin, when you do a seek when paused, the video player doesn't show you the new frame unless you start to play the video. To alleviate this problem, whenever the user does a seek, the application will seek a few hundred milliseconds ahead of the desired time and add an event listener to the player monitoring when the needed data is loaded, and then play the video for a very short time and stops the playing as soon as the position reaches the actual desired time in the hope of at least giving a visual feedback of the seek operation. Sometimes if you are lucky you will catch a keyframe during that short period of time and actually see a clear image, but most likely if you are seeking to a section of the video that has not been buffered, the image will be smudged until you reach the next keyframe. But it is better than seeing no visual feedback.

5. Conclusion

We introduced an application that groups together video resources in an organized way to create new video documents from existing videos. The users are not actually making new videos so there wouldn’t be any infringement of copyright even if the user is using a copyrighted video. Such video document is extremely easy to create and share. With annotations, one could add more content to the video and truly make it their own. With new emerging web technologies, the potential for this type of video splicing and editing is quite promising.