Title: Ability to watch media “edited” or “personalized” or “enhanced” for arbitrary (online and offline) movie players.

Abstract:

The invention described here can ascertain the “current playback time” from any arbitrary video player while it is playing on a device (ex: Netflix instant, hulu, hulu plus, VLC, amazon VOD, etc.,) and then use that information to perform any number of reactions, for instance “muting” profanity during a known timestamp, or skipping/obscuring a scene known to be offensive, etc. The invention also describes some playback/quality enhancements that can be used for any player’s arbitrary playback.

Details:

Basically this invention is a computer program that can track/monitor another arbitrary video player, during playback of its video. It can determine the player’s “current” playback time, then react if the playback time is related to timestamps of interest described somewhere and applicable to the given media. It can determine current playback time by using several methods. One is the API of a player in question (for instance, the youtube SDK in HTML/javascript, or the VLC movie player’s libvlc API which provides callbacks providing current video time). The other is that it can determine current playback time by taking a “screen snapshot” of a playing movie player, then using OCR (Optical Character Recognition) to determine the digits being currently displayed in the player’s on-screen playback timestamp, thus inferring the current playback position from the digits’ values (figure 1, in red—it takes a snapshot of those digits, then OCR’s them to discover current playback position, without having need of an API of the player available):

This is novel since (as far as I’m aware) there is no content edited playback for various online players like Netflix Instant/hulu (nor blu-ray), and it also introduces some new aspects to content edited playback that are not yet present in today’s marketplace.

It also includes some methods for “video enhancement” when playing back a video.

It is mostly based on the “sensible cinema” software project [8] which began in June of 2010 with some experimental features, including DVD -> edited movie file type conversion, then added some other features described here.



Figure

OCR techniques:

To achieve the screen snapshot and OCR, player “descriptors” are created which describe where to expect timestamps, given a certain player (alternatively, it could just OCR the entire monitor/screen). The descriptors can specify clues to detect which windows “contain” a currently playing video player, for instance specifying a window class ID, or a window title or regular expressions, which can be used within the Operation System’s window hierarchy to identify a window that contains a player. Or it can specify to extract from the “full screen” (i.e. based on full screen size). It can also specify absolute or relative position of coordinates for the known location of the on-screen timestamp, and/or each of its respective digits, or it can specify percentage coordinates (relative coordinates) to allow the monitor program to “find” where the timestamp digits occur, or are anticipated to occur. For instance “50 px from the left, plus 10%” could be an upper left most coordinate for the timestamps, and it could specify a height of 15 pixels per digit. It can then either screen snapshot the player, by reading from on-screen memory, then OCR either the entire timestamp (like all of “00:04/14:48” or of “00:04”) or snapshot and OCR the digits each individually. The descriptor can tell it it where each digit resides, then it OCR’s each, for instance “00:04” OCR would be [0,0,’:’, 0,4] which it combines to mean 0 minutes, 4 seconds. It could also snapshot an individual frame by requesting it from the video player’s API (doesn’t have to use screen capture).

The GOCR program is an option in practice, for performing OCR [1].

OCR:

To speed up OCR/make it more accurate, the program can “cache” the results of a previous screen capture and their given OCR value (or save and refer to the cached value from some kind of persistent cache). This allows the program to quickly identify the digits found in “previously OCR’ed images” and thus avoid recalculation of the OCR. In this way it saves CPU and can be edited in the case of failure. It could be manually pre-trained (i.e. “this image always means this digit”) to avoid having to ever do OCR at all, ever.

It can also take advantage of certain advanced features of the GOCR program, like being able to specify a “contrast limit” for detecting which pixels should be considered “part of the character” or by presenting it with an expected character list, ex: “this image should contain only the characters 0-9”. It can also use leverage other image manipulation, for instance sharpen the image, or convert from a black back ground to white background, before performing OCR, for improved accuracies.

After ascertaining the player’s current time signature/timestamp, it can get even more accuracy by tracking elapsed time since the last timestamp updated. For instance, it can remember the time that the on screen display changes (via polling for change within the section of the screen known to have the time code, or by OCR’ing constantly and looking for changes). When an image change is detected, it can start an internal timer that infers, from the time elapsed since the image was changed, sub-second accuracy since the time that it registers after the image last updated. For instance if it wants to know when second 4.5 occurs, it can wait 0.5 seconds of “wall time” after the change to 0:00:04 has occurred). It can also use some special tools to avoid confusion in parsing timestamps, like pausing briefly to allow for the “minute” digit to change after it detects that second digits have just changed from 59 to 00, etc.

Mouse Control during playback:

It can also control and programmatically move the system’s mouse periodically to make it so that the on screen timestamp is displayed. Some players “hide” their timestamp after a certain amount of mouse inactivity, so it programmatically can avoid this hiding. It could also perform this mouse movement less frequently, for instance only jittering it every 30 seconds for 3 seconds, to resynchronize, based on the next change of timestamp, thus avoiding having the on screen timestamp always visible. It could optionally perform this double check more frequently when it approaches “time stamp of known interest” (for instance, always move the mouse when it anticipates the player is near the start of an EDL section of interest, see below).

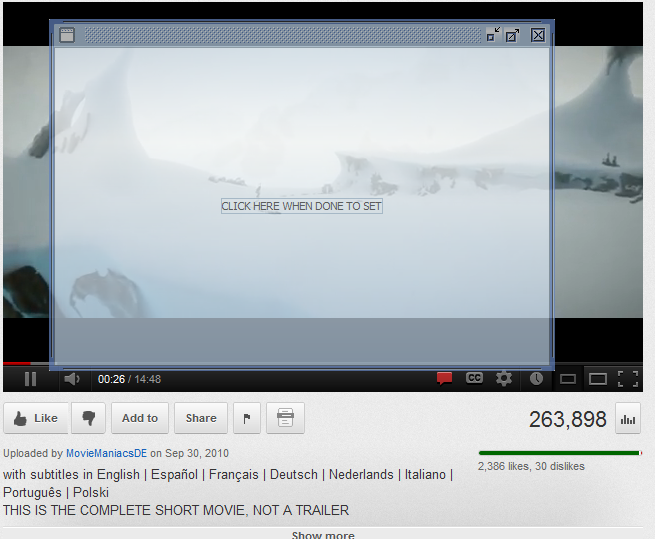
Reactions:

When the program detects that the client player in question has reached or is about to reach or is within a “timestamp of known interest” it can react with optionally several “reactions.”

It could mute playback by muting the player, via an API [ex: YouTube, or MS DVD Navigator mute or volume adjustment calls] or mute by using a mouse click to the location for the current player’s mute button (by moving the mouse there, then simulating a mouse click), or by muting the system volume. It could also partially mute, by modifying volume, by using a modification of the above (for instance, changing system volume to 10%, or using mouse clicks to simulate changing player volume to low, then back to high when it’s done). It could (itself) play some other “overlay” audio file.

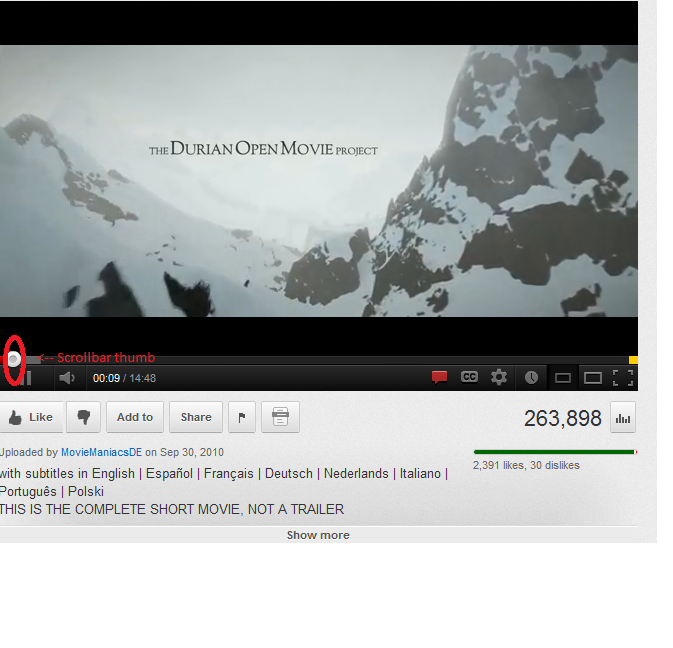
It could start playback of some video “over the top” of the video player (with or without alpha overlay display, which if used, would allow still showing of only certain parts of the video). For instance, it could display an entirely opaque obscuring “window” over the top of the player, to avoid viewing of nudity, until it infers that the time in question has passed [Figure 2] based on wall clock time, or it could overlay and still capture the player in certain operating system’s (Windows 7 DWM for instance). The overlay window could be made to either obscure the entire screen, or just certain sections of it. Or it could display several different overlay windows at the same time. It could also display a window that overlays “related text” that is to be read during the time it is on screen. The overlay window can thus simulate “skipping” a scene (when coupled or not with a mute), without having to control the player, nor needing an API.

Figure 2 example overlay window reaction, with text (“click here when done to set” could become any arbitrary text, and alpha level could be adjusted depending on what the action is specified to do):



Another action possible would be to seek to a new location, by using an API method call, or by using the mouse to drag the “playback bottom scrollbar thumb” (Figure 3, see also [2]) from its current location in some direction (for instance “right 4 pixels” to simulate moving forwarding X number of seconds). It basically could seek anywhere. It could also detect thumb location using computer vision type heuristics, similar to OCR.

Figure 3:



Another possible action would be to change the browser’s url (redirect it) to a different page, or open up a new browser window at a certain time. This might even include ads to be shown during a specified portion of a video playback. All of the above could be specified to be looping, as well (i.e. “every 10 minutes, display an ad in a different window”).

Another possible action would be to accelerate the video, during a certain time period, by using the API or simulating mouse clicks to select a speed setting. It could also make the video window “smaller” (resize or minimize it) during a certain time frame (for instance to dampen the effect of “scary” scenes) by using an API or generic window resize methods given by the Operating System). It could also add a “blur filter” over certain parts of the video, similar to an overlay video, discussed above, by super imposing a video or by capturing then redisplaying the video with a blurred portion.

It could also “mix and match” any of the above reactions, and/or have several going at once, etc.

It could also “mute during ads” by detecting changes and responding to them, etc.

Edit Decision List Format:

It stores reactions and metadata about the reactions (and about the movies themselves, and players) in some computer readable serialized format (like JSON or YAML or XML or custom format that is machine readable). It could store them in sections, with each section being descriptive (like “metadata about the media”) or containing a reaction description, or contain a list of those (nested bunch of the same). Reaction specifications are basically an EDL (Edit Decision List [3]) typically having a start time, end time, and an action to take during that time, as well as descriptive data. They could include a category, or sub category, a “rating” level within that category (for instance, “profanity”, “scatalogical”, “level: 5” [out of 10]). It could include descriptive text, for instance, full text (or euphemized text) of subtitles of the profanity in question, i.e. show what’s being muted, etc. It can have level/intensity information (“75% opacity”, or “decrease opacity to 50% by 5 seconds”). The EDL itself could also have new rating information (ex “with this edl, estimated age appropriate violence, age 12, drug use age 9”, or “PG”). It could have movie rating, as well, like “75% of users like this film,” “its IMDB url is X”, “its DVD unique ID is y”.

These reactions could serve the purpose of editing a film for content (ex: skipping violence/sex scenes, etc.), or skipping “boring parts” of films or providing alternate film endings, skipping plot holes, introducing new content, new voice overs, etc.

For an example demo EDL file created, see appendix A.

Including categories and levels and also individual details allows users to be able to select/reject items within the EDL by “specific item” or by “categories” or by “levels” (either programmatically “I dislike these specific profanities, or this category, or this rating” or manually, before using them for playback, by examining/editing the list). Optionally, users can select multiple EDL’s that apply to a given movie, for instance one EDL file identifies and mutes profanity, and another that identifies and skips violence can be combined. They can be prompted to and choose to combine the two or just use one or the other, or have a preference to automatically combine all. EDL’s themselves can include metadata, like which films/url’s/media they are associated with, metadata information about the film, actors, descriptive data (links to images, information about the film, subtitles, etc.). They could also be bundled with or distributed with useful other related files, like images concerning the media, or make them available for download.

Since some players’ on screen timestamps are not quite precise/accurate from computer to computer or even playback to playback on the same computer, it is also possible to add an extra “buffer time” to all timestamps, to ensure that this discrepancy is accommodated for.

It could also automatically infer from a browser’s current URL (or DVD’s unique ID), the EDL’s in its system that apply to that particular movie that is being displayed.

It can have community contribution for EDL creation (i.e. crowd sourcing creation/editing), such that end users can share EDL’s/timestamp information either through sharing files (offline) or through some web service that contains submissions (similar to a wiki, which could also include editable metadata on a film), and that can be queried/polled for EDL’s for a film, or downloaded every so often to resynchronize with the latest versions of each film/local changes, and referenced later to determine a specific film.

It can have “personalized” customized EDL’s as well, locally editable.

It can have an editor to aid in creation of the EDL that uses detected current playback time and allows users to specify start and end times, and allows them to select actions, and test their edits (with or without seeking first to find them). This can be in a standalone app on a computer, or within an HTML page. It can also allow for editing/creation of EDL items during playback (for instance, while playing, they can add to/delete from/edit the EDL the user is using, for instance, update a timestamp to be more accurate, add notes/commentary to it). The changes are automatically applied locally, and/or synchronized with a central server.

It could send an EDL-type file to a streaming player “source” (for instance, send it to Netflix’s servers for “Netflix Instant”, or send those servers them user preference information (or have them enter preference information on the receiving website), and then they apply it to EDL’s they already have/know of for a given media). The streaming source server could then do the editing for the end user, for instance skipping undesirable scenes “automatically” underneath, adjusting volumes on the fly, superimposing images, etc. The streaming server could also download the EDL’s itself, when desired.

It could “save” an instance of a media’s playback “while edited” (for instance, screen capture the playback), then allow the user re-watch the now edited media that at a later date, or on separate device. It could also take an input file and extract specific parts of it/adjust audio, etc. to create an output file that is an edited version of the original.

Subtitles:

It could take audio (or video subtitles/closed captioning) and programmatically “transcribe” or OCR it to determine subtitles and their timings. It could use a lookup database service to auto-download subtitles/other metadata for a known media. It could also detect non text cues (for instance, loud noise/volume, skin tones) to auto-detect timestamps useful in creating the EDL.

Once it has the subtitles, it can adjust them to synchronize them with current media (for instance, if the subtitles are off by 5 minutes, or a scale of 1.001, it can rescale the subtitles and/or EDL itself to match the current media, based on offset and scale) if they don’t match.

Once it has accurate timings for subtitles, it can use them, given "known profanities and/or euphemisms" automatically infer profanity timestamps, or other scenes that are questionable, that should be automatically skipped or obscured or both, and add them to an EDL. It can also allow the user to enter personalized word/phrase lists to be included in the scanning and automatically added to the EDL as well. It can also give the user a list of possible profanities, and let them choose which to add and which to leave out.

It could automatically detect when commercials are being played back and perform some action, for instance when it detects that a 30 second commercial is playing, it can mute until the commercial is over, or any other reaction.

It can also playback media “edited” by converting EDL’s into the format used by 3rd party players (like Mplayer [6], movie-content-editor [5], ZoomPlayer MAX [4] etc.), then having those players perform EDL actions themselves, internally. Given a known video file, it could create a “side by side” edl file, which is then picked up and used automatically by players, like XBMC and Smplayer [7] which support this feature.

In can also convert timestamps depending on the player, for instance DVD players sometimes have a different offset, and playback in 30 fps when other players (or files) are based on 29.97 fps. It converts from one timestamp to the other so that it works with both.

It can automatically detect a media’s applicable EDL’s based on md5 value, or title. It can programmatically detect, from the browser in question (that is displaying a player), the url, and use that information to lookup applicable EDL’s and present them or automatically use them during playback. It could be made into a browser plugin with this capability, as well. It can also use English text in the URL to lookup/display information/metadata about the movie in question.

It can automatically detect a DVD’s applicable EDL’s based on DVD Unique ID: <http://api.themoviedb.org/2.1/ids-hashes> as well as use this retrieve data/information about the media and display it.

It can translate an EDL into a “playlist” style playback form (for instance, a playlist that instructs a player to play from second 0 to second 5, then play muted from second 5 to second 10, then play at full volume from second 10 to 20, etc.). This allows players to natively playback EDL’s, even if they lack EDL support per se (for instance, a DOS batch file could instruct a player to do the same).

It can play back a video to a computer screen, edit the playback, then broadcast that (edited) screen and its audio to the user’s console device or TV or other device, for edited streaming/remote playback.

It can use video up-conversion techniques (for instance upscale resizing by using the lanczos re-scaling method, and other post processing techniques such as using denoise and post processing to remove blockiness, etc.) to display a different final video, by screen capturing the initial playback, then processing it, then re-displaying it or streaming it to another device or computer. It can also optionally process displayed video and use it to infer coloring for a “back lighting” system for when the video is being played (for instance, when there is lightning, it lights up the back wall behind the film/TV with white light, when the lightning ends, it goes back to being black, etc. For example having one monitor on your laptop pointed at the wall, which displays “all white” during the lightning, to provide lighting, or any other lighting device. It could also target multiple walls/ceiling, etc.). It could also stretch or redisplay the video to fit multiple monitors/projectors (for instance, one pointed at the ceiling to give a more immersive view or attempt to simulate an IMAX theater, etc.)

Audio:

It can also playback edited “pure audio” files/online streams, in a similar way to how it does video. It can also re-save audio files as ones that have edits applied to them. For instance stripping “chapter headings” and stripping/muting profanity, speeding up the audio (global directives, like “speed\_up\_factor” => 1.25), etc. based on user preferences.

It can also use automatic auto upscale/upconvert the audio or add to it during playback or streaming.

Related works:

Almost [any professional editing system and many others](http://en.wikipedia.org/wiki/Comparison_of_video_editing_software#Output_options) support some form of XML/EDL saving/processing, as listed on <http://en.wikipedia.org/wiki/Edit_decision_list>.

Comskip: <http://www.kaashoek.com/comskip> Analyzes video for certain aspects and auto-skips scenes based on computerized criteria, it also includes machine learning for that purpose.

Movie Content Editor [5] has capability to take a list of subtitles (captions) for a VLC compatible device (file, DVD) and control VLC to mute or skip over scenes with detected profanity.

ZoomPlayer Max [4] has the ability to apply EDL’s to media like DVD’s and blu-ray’s. Their EDL’s can mute, turn off/on enabling of subtitle display, skip forward, or skip to different section/point in the same film (skipping to another film would be an interesting option if provided). It also has an “inline” type editor that allows you to select sections based on the current timestamp, and review and modify sections inline.

Clearplay can do some related edited playback, though just for DVD’s, not for arbitrary [i.e. online] players.

TVGuardian can do some parsing of subtitles (or is it closed captions) to automatically (and in almost realtime) detect and mute profanity.

For a more extensive related work list, see <https://github.com/rdp/sensible-cinema> file: history\_and\_related\_works\_list.txt.

References:

[1] GOCR: <http://jocr.sourceforge.net/>

[2] Scrollbar thumb described here: <http://en.wikipedia.org/wiki/Scrollbar>

[3] <http://en.wikipedia.org/wiki/Edit_decision_list>

[4] <http://www.inmatrix.com/zplayer/scenecut.shtml>

[5] <http://code.google.com/p/movie-content-editor>

[6] <http://www.mplayerhq.hu/DOCS/HTML/en/edl.html>

[7] <http://smplayer.sourceforge.net/>

[8] <https://github.com/rdp/sensible-cinema>

Initial publication thoughts:

<http://betterlogic.com/roger/2009/02/user-contributed-movie-scene-selection-or-past-clearplay/>

See also <https://github.com/rdp/sensible-cinema> “change log” and “todo.inventionzy.txt” files for more published ideas/thoughts on the matter.

Appendix A:

Demo/example EDL file

# comments can go after a # on any line, for example this one.

"name" => "aladdin",

"mutes" => [

"0:04:15.95" , "0:04:20.44", "profanity", "al...", "By [al...]",

"1:21:44.66" , "1:21:48.41", "profanity", "moron", "Shut up you [\_\_\_] Don't tell me to shut up", “category”, “profanity”,

],

"blank\_outs" => [

"7:51", "7:55", "skimpy dancing", “lewdness”, “category 10”, “level75”,

"8:00", "8:07", "skimpy dancing",

],

"volume\_name" => "QL10NNW1",

"disk\_unique\_id" => "f0f84375|1dbbed43",

"dvd\_title\_track" => "20",

“new\_rating\_if\_watched\_with\_this\_edl” => {“violence” => “age 12”},

# "dvd\_title\_track\_length" => "9999", # length, on the DVD, of dvd\_title\_track (use the show DVD info button to get this number).

"subtitle\_url" => "http://www.opensubtitles.net/en/opensubtitles-player.sexabe/3289935",

"not edited out stuff" => "there maybe more in there",

"closing thoughts" => "only...",

"dvd\_start\_offset" => "0.28",

"beginning\_subtitle" => ["Oh I Come from a land", "30.5"],

"ending\_subtitle\_entry" => ["Made you look. ", "1:25:33.37"],

]

XML or JSON/YML would work just as well.