Enabling on-the-fly Video Shot Detection on YouTube

Thomas Steiner
Google Germany GmbH
ABC-Str. 19
20354 Hamburg, Germany
tomac@google.com

Ruben Verborgh
Ghent University – IBBT, ELIS
Multimedia Lab
9050 Ghent, Belgium
ruben.verborgh@ugent.be

Joaquim Gabarró Vallés
Universitat Politècnica
de Catalunya
08034 Barcelona, Spain
gabarro@lsi.upc.edu

Michael Hausenblas
DERI, NUI Galway
IDA Business Park
Lower Dangan Galway, Ireland
michael.hausenblas@deri.org

Raphaël Troncy
EURECOM
2229 route des crêtes, BP 193
Sophia Antipolis, France
raphael.troncy@eurecom.fr

Rik Van de Walle
Ghent University – IBBT, ELIS
Multimedia Lab
9050 Ghent, Belgium
rik.vandewalle@ugent.be

ABSTRACT

Video shot detection is the processor-intensive task of splitting a video into continuous shots, with hard or soft cuts as the boundaries. In this paper, we present a client-side on-the-fly approach to this challenge based on modern HTML5-enabled Web APIs. We show how video shot detection can be seamlessly embedded into video platforms like YouTube using browser extensions. Once a video has been split into shots, shot-based video navigation gets enabled and more fine-grained playing statistics can be created.

Categories and Subject Descriptors

I.2.10 [**Vision and Scene Understanding**]: Video analysis; H.5.1 [**Multimedia Information Systems**]: Video (e.g., tape, disk, DVI)

General Terms

Algorithms

Keywords

Shot detection, shot boundary detection, video processing

1. INTRODUCTION

Official press statistics [12] from YouTube, one of the biggest online video platforms, state that more than 13 million hours of video were uploaded during 2010, and that 48 hours of video are uploaded every single minute. Given this huge amount of video content, it becomes evident that advanced search techniques are necessary in order to retrieve the few needles from the giant haystack. Closed captions allow for keyword-based in-video search, a feature announced in 2008 [4]. Searching YouTube for a phrase like "that's a tremendous gift", a caption from Randy Pausch's famous last lecture *Achieving Your Childhood Dreams*[1], reveals the video of his lecture. If no closed captions are available, nor can be automatically generated, keyword-based search is still available over tags, video descriptions, and titles. Presented with a potentially long list of results, preview thumbnails based on video still frames help users decide on the most promising result. YouTube uses an unpublished computer vision-based algorithm for the generation of smart thumbnails on YouTube and lets video owners choose one out of three automatically suggested thumbnails.

In this paper, we introduce on-the-fly shot detection for YouTube videos as a third means besides keyword-based search and thumbnail preview for deciding on a video from the haystack. As a user starts watching a video, we detect shots in the video by visually analyzing its content. We do this with the help of a browser extension, i.e., the whole process runs dynamically on the client-side, using modern HTML5 JavaScript APIs of the `<video>` and `<canvas>` elements [8]. As soon as the shots have been detected, we offer the user the choice to quickly jump into a specific shot by clicking on a representative still frame. Figure 1 shows the seamless integration of the detected shots into the YouTube website enabled by the browser extension. The main contributions of this paper are the browser extension itself and improved video navigability by shot navigation. A screencast[2] and demo[3] of our approach are available.

2. RELATED WORK

Video fragments consist of shots, which are sequences of consecutive frames from a single viewpoint, representing a continuous action in time and space. The topic of shot boundary detection has already been described extensively in literature. While some specific issues still remain (notably gradual transitions and false positives due to large movement or illumination changes), the problem is considered resolved for many cases [6, 13]. The contribution of our approach is that it is entirely Web-based and on-the-fly, which introduces interesting new challenges that traditional approaches do not have to cope with. Highest shot detection

WWW2012 Developer Track, April 18–20, 2012, Lyon, France..

[1]Last Lecture: `http://bit.ly/pausch-last-lecture`
[2]Screencast: `http://bit.ly/filmstrip`
[3]Demo: `http://bit.ly/filmstrip-debug`

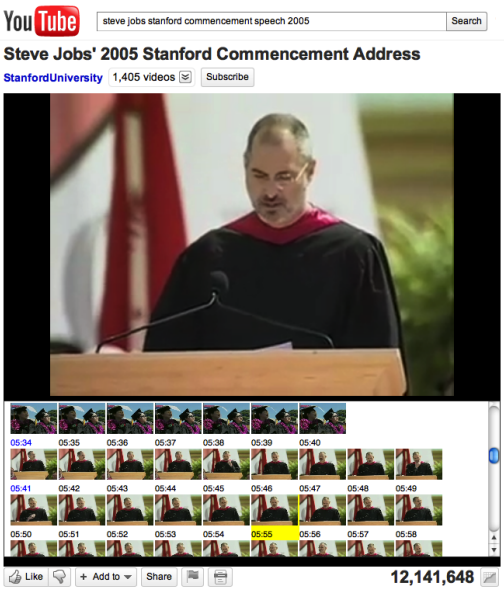

Figure 1: Screenshot of the browser extension running on YouTube, showing different detected shots.

accuracy can be reached with special command line tools, but therefore access to the hosting platform is needed, which in the general case is not given on the Web. Our Web-based approach abstracts away most of the low-level details like the video codec, in favor for the high-level `<video>` API, however, this also comes at a cost. A major issue is the uncertain streaming speed, where traditional approaches have immediate access to the video file on disk. An additional challenge is the unknown key frame distribution of the target videos, which – together with streaming speed issues – makes exact frame-wise video navigation impossible. Below, we present an overview of several well-known categories of shot detection techniques.

Pixel comparison methods [5, 15] construct a discontinuity metric based on differences in color or intensity values of corresponding pixels in successive frames. This dependency on spatial location makes this technique very sensitive to (even global) motion. Various improvements have been suggested, such as prefiltering frames [16], but pixel-by-pixel comparison methods proved inferior in the end and have steered research towards other directions.

A related method is *histogram analysis* [10], where changes in frame histograms are used to justify shot boundaries. Their insensitivity to spatial information within a frame makes histograms less prone to partial and global movements in a shot. We argue as a drawback that even visually very dissimilar frames can have similar overall histograms. For example, different shots in the same shot can be difficult to distinguish because of similar color information.

As a compromise, a third group of methods consists of a *trade-off between the above two techniques* [1]. Different histograms of several, non-overlapping blocks are calculated for each frame, thereby categorizing different regions of the image with their own color-based, space-invariant fingerprint. The results are promising, while computational complexity is kept to a minimum, which is why we have chosen a variation on this approach in this paper.

Other approaches to shot boundary detection include the *comparison of mean and standard deviations* of frame intensities [9]. Detection using other features such as edges [14] and motion [2] have also been proposed. However, Gargi *et al.* have shown that these more complex methods do not necessarily outperform histogram-based approaches [3]. A detailed comparison can be found in Yuan *et al.* [13]. At time of writing, YouTube is about to roll out a similar native feature, however, frame-based and not shot-based as our approach[4].

3. SHOT DETECTION ALGORITHM

In this Section, we discuss our shot detection algorithm, which falls in the category of histogram-based algorithms. Since visually dissimilar video frames can have similar overall histograms, we also take local histograms into account. We therefore split video frames in freely configurable rows and columns, i.e., lay a grid of tiles over the frames. The user interface (Figure 2) currently allows for anything from a 1×1 grid to a 20×20 grid. For each step we examine a frame f and its direct predecessor frame $f - 1$.

Apart from the per-tile histogram average distance, the frame distance function further considers a freely configurable number of *most different* and *most similar* tiles. This is driven by the observation that different parts of a video have different intensities of color changes, dependent on the movements from frame to frame. The idea is thus to increase the influence of movements in the frame distance function, and to decrease the influence of permanence. In the debug view of our approach (Figure 2), blue boxes indicate movements, while red boxes indicate permanence. In the concrete example, Steve Jobs' head and shoulders move as he talks, which can be clearly seen by the blue boxes in the particular tiles. Additional movements come from a swaying flag on the left, and a plant on the right. In contrast, the speaker desk, the white background, and the upper part of his body remain static, resulting in red boxes. For this example, we use a grid layout of 20×20 tiles ($nTiles = 400$), and a certain number *tileLimit* of most different or similar tiles, i.e., we treat one third of all tiles as most different tiles, one third as normal tiles, and one third as most similar tiles, and apply boosting and limiting factors to the most different and most similar tiles respectively. This distribution was empirically determined to reveal best results. We work with also empirically determined values of 1.1 for the *boostingFactor*, which slightly increases the impact of the most different tiles, and 0.9 for the *limitingFactor*, which slightly decreases the impact of the most similar tiles. The algorithm pseudo code can be seen in Listing 1.

We define the average histogram distance between two frames f and $f - 1$ as $avgHisto_f$. In a first step, we have examined the histogram distance data statistically, and experimentally found out that while the overall average frame distance $avgDist_f$, defined as:

$$avgDist_f = \frac{1}{nTiles} \sum_{t=1}^{nTiles} avgHisto_{f,t}$$

is very intuitive to human beings, far more value lies in the standard deviation $stdDev_f$, based on the definition of the

[4] `http://youtube-global.blogspot.com.au/2012/03/looking-ahead-in-youtube-player.html`

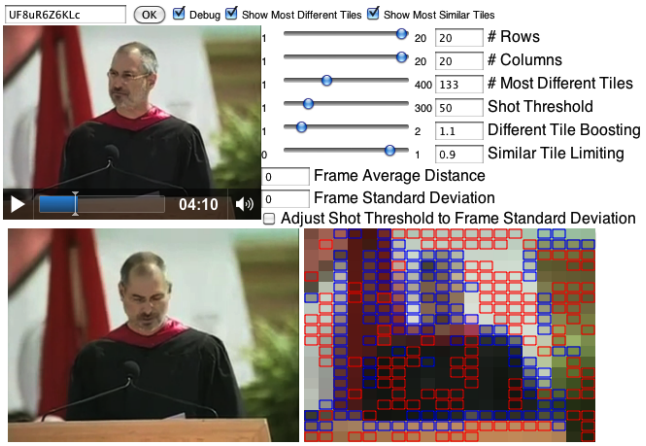

Figure 2: Debug view of the shot detection process. Blue boxes highlight tiles with the most differences to the previous frame, red boxes those with most similarities.

```
for frame in frames
  f = frame.index
  for tile in tiles of frame
    avgHisto[f][tile] = getTilewiseDiff()

  mostDiffTiles = getMostDiffTiles(avgHisto[f])
  mostSimTiles = getMostSimTiles(avgHisto[f])

  for tile in tiles of frame
    factor = 1
    if tile in mostDiffTiles
      factor = boostingFactor
    else if tile in mostSimTiles
      factor = limitingFactor
    avgHisto[f][tile] = avgHisto[f][tile] * factor
  avgDist[f] = avg(avgHisto[f])
```

Listing 1: Pseudocode of shot detection algorithm.

overall average frame distance $avgDist_f$:

$$stdDev_f = \sqrt{\frac{1}{nTiles} \sum_{t=1}^{nTiles} (avgHisto_{f,t} - avgDist_f)^2}$$

We use the value of the standard deviation as a value for the shot splitting threshold [9] to come to very accurate shot splitting results. We found the boosting and limiting factors to have overall a positive quality impact on more lively videos, and a negative quality impact on more monotone videos. Best results can be achieved if, after changing either the boosting or the limiting factors for the most similar or different tiles, the value of the shot splitting threshold is adapted to the new resulting standard deviation. The user interface optionally does this automatically.

4. IMPLEMENTATION DETAILS

Our shot detection algorithm is implemented in form of an extension for the Google Chrome browser. Chrome extensions are small software programs written in a combination of HTML, JavaScript, and CSS, which users can install to enrich their browsing experience. For this paper we focus on extensions based on so-called content scripts. Content scripts are JavaScript programs that run in the context of Web pages via dynamic code injection. By using the standard Document Object Model (DOM), they can read or modify details of the Web pages a user visits. The advantage of this browser extension approach is that it is very powerful and generalizable at the same time. Powerful in the sense that it allows for significantly changing ones user experience with a platform like YouTube and simply add new features, and generalizable in the sense that in theory it would be possible to simply add video shot boundary detection to any HTML5-enabled video website.

The complete video analysis process happens fully on the client side. We use HTML5 JavaScript APIs of the `<video>` and `<canvas>` elements. The extension is activated as soon as the user enters a YouTube video watch page. By default, YouTube uses Flash-encoded videos that are not programmatically accessible from a JavaScript context, however, via an API used by the YouTube `<iframe>` embed code, we can replace the Flash version with the HTML5 version of a video. In order to obtain a video still frame from the `<video>` element at the current video position, we use the `drawImage()` function of the 2D context of the `<canvas>` element, which as its first parameter accepts a `<video>` element. We then analyze the video frame's pixels tile-wise and calculate the histograms. In order to retrieve the tile-wise pixel data from the 2D context of the `<canvas>`, we use the `getImageData()` function. For processing speed reasons, we currently limit our approach to a resolution of one second, i.e., for each analysis step seek the video in *1s* steps. We then calculate the frame distances as outlined in Section 3. For each frame, we generate an `<img>` element with a base64-encoded data URI representation of the video frame's data that later gets injected into the DOM tree of YouTube, as can be seen in Figure 1. Each of the `<img>` elements has a registered JavaScript event handler that upon click triggers two actions: first, the video seeks to the corresponding time, and second, the shot is tracked as a hot spot in the video. Clicks on hot spots can be tracked using standard Web analytics services, which allows for the suggestion of more accurate entry points to videos in the longterm. In prior work [11] we have shown how hot spots can be used to detect different kinds of events in videos.

5. EVALUATION

Detecting shots on-the-fly in streaming video comes with its very own challenges. First, it is a question of streaming speed. We have to stream the same video twice in parallel: on the one hand the user-visible foreground video, and on the other hand the video used in the background for the analysis process. Especially with high-definition (HD) video this can be very demanding. We do not attach the background `<video>` element to the DOM tree to save some CPU cycles, however, the background video still needs to be seeked to each frame in second-steps and be processed, while the foreground video is playing normally. Even on a higher-end computer (our experiments ran on a MacBook Pro, Intel Core 2 Duo 2,66 GHz, 8 GB RAM), the process of analyzing and displaying in parallel a *1280 × 720* HD video of media type *video/mp4; codecs="avc1.64001F, mp4a.40.2"* causes an average CPU load of about 70%. The HTML5 specification states that *"[...] when the playback*

rate is not exactly 1.0, hardware, software, or format limitations can cause video frames to be dropped [...]" [7]. In practice, this causes the analysis environment to be far from optimal. In our experiments we differentiated between false positives, i.e., shot changes that were detected, but not existent, and misses, i.e., shot changes that were existent, but not detected. Compared to a set of videos with manually annotated shot changes, our algorithm detected fewer false positives than misses. The reasons were gradual transitions and shots shorter than one second (below our detection resolution) for misses, and large movements in several tiles for false positives. Overall, we reached an accuracy of about 86%, which is not optimal, but given the challenges sufficient for our use case of facilitating in-video navigation. Performance potential resides in the usage of Web Workers, JavaScript programs that run in the background, independently of other user interface scripts.

6. FUTURE WORK AND CONCLUSION

In a first step, future work will consist in improving the analysis speed by dynamically selecting lower quality analysis video files, given that videos are available in several resolutions (both LD and HD). We will check in how far analysis results differ for the various qualities. In a second step, we will work on more advanced heuristics for the various user-definable options in the analysis process (Figure 2). While there is no optimal configuration for all types of videos, there are some key indicators that can help categorize videos into classes and propose predefined known working settings based on the standard deviation $stdDev_f$ and the overall average frame distance $avgDist_f$. Both are dependent on the values of *boostingFactor*, *limitingFactor*, *rows*, and *columns*. Interpreting our results so far, there is evidence that low complexity settings are sufficient in most cases, i.e., a number of *rows* and *columns* higher than *2* does not necessarily lead to more accurate shot detection results. The same applies to the number of to-be-considered most different or similar tiles *tileLimit*. We even had cases where not treating those tiles differently at all, i.e., setting $boostingFactor = limitingFactor = 1$, led to better results. We plan to use an adapted version of the algorithm for event summarization based on user-generated video content.

Concluding, we have promising results, especially in combination with hot spot identification from prior work [11], which in future can provide for more accurate entry point suggestions in long videos, or enable skip marks in videos with intros. For video shot detection, some challenges remain, especially with the observed streaming speed-related issues and predefined parameter settings to improve accuracy. Nonetheless, when searching for a certain point in a video, the obtained results already allow for focused in-video navigation and also facilitate exploratory shot-wise video consumption.

7. ACKNOWLEDGMENTS

R. Verborgh is funded by Ghent University, the Interdisciplinary Institute for Broadband Technology, the Institute for the Promotion of Innovation by Science and Technology in Flanders, the Fund for Scientific Research Flanders, and the EU. T. Steiner is partially supported by the EC under Grant No. 248296 FP7 I-SEARCH project. J. Gabarró is partially supported by TIN-2007-66523 (FORMALISM), and SGR 2009-2015 (ALBCOM).

8. ADDITIONAL AUTHORS

Arnaud Brousseau (Google Germany intern, EURECOM student, email: `arnaud.brousseau@gmail.com`).

9. REFERENCES

[1] M. Ahmed, A. Karmouch, and S. Abu-Hakima. Key frame extraction and indexing for multimedia databases. In *Proc. of the Visual Interface Conf.*, pages 506–511, 1999.

[2] P. Bouthemy, M. Gelgon, and F. Ganansia. A unified approach to shot change detection and camera motion characterization. *IEEE Transactions on Circuits and Systems for Video Technology*, 9:1030–1044, 1997.

[3] U. Gargi, R. Kasturi, and S. H. Strayer. Performance Characterization of Video-Shot-Change Detection Methods. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(1):1–13, 2000.

[4] Google Video Blog. Closed Captioning Search Options, June 05, 2008. http://goo.gl/crmWZ.

[5] A. Hampapur, T. Weymouth, and R. Jain. Digital video segmentation. In *Proc. of the 2nd ACM Int. Conf. on Multimedia*, MULTIMEDIA '94, pages 357–364, New York, NY, USA, 1994. ACM.

[6] A. Hanjalic. Shot-Boundary Detection: Unraveled and Resolved? *IEEE Transactions on Circuits and Systems for Video Technology*, 12(2):90–105, Feb. 2002.

[7] I. Hickson. HTML Living Standard – The Video Element, July 29, 2011. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-video-element.html.

[8] HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft, November 2011. http://www.w3.org/TR/html5/.

[9] R. Lienhart. Comparison of automatic shot boundary detection algorithms. In *Storage and Retrieval for Image and Video Databases*, pages 290–301, Jan. 1999.

[10] A. Smeaton, N. Murphy, and S. Marlow. Evaluation of automatic shot boundary detection on a large video test suite. In *Institute for Image Data Research, Univ. of Northumbria at Newcastle*, pages 25–26, 1999.

[11] T. Steiner, R. Verborgh, R. Van de Walle, et al. Crowdsourcing Event Detection in YouTube Videos. Workshop DeRiVE at the 10th Int. Semantic Web Conf., Bonn, Germany, October 23-27, 2011.

[12] YouTube. Official Press Traffic Statistics, 2011. http://www.youtube.com/t/press_statistics.

[13] J. Yuan, H. Wang, L. Xiao, W. Zheng, J. Li, F. Lin, and B. Zhang. A formal study of shot boundary detection. In *IEEE Transaction on Circuit and Systems For Video Technology*, pages 168–186, 2007.

[14] R. Zabih, J. Miller, and K. Mai. A feature-based algorithm for detecting and classifying scene breaks. In *Proc. of the 3rd ACM Int. Conf. on Multimedia*, pages 189–200, New York, NY, USA, 1995. ACM.

[15] H. Zhang, A. Kankanhalli, and S. W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28, 1993.

[16] H. Zhang, C. Y. Low, and S. W. Smoliar. Video parsing and browsing using compressed data. *Multimedia Tools & Applications*, 1:89–111, March 1995.