Story Tracking
in Video News Broadcasts

Jedrzej Zdzislaw Miadowicz

M.S., Poznan University of Technology, 1999

Submitted to the Department of Electrical Engineering and Computer Science
and the Faculty of the Graduate School of the University of Kansas
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

_____
Chair

_____

_____

_____

_____

Date Defended: June 4, 2004

Abstract

Since the invention of television, and later the Internet, the amount of video content available has been growing rapidly. The great mass of visual material is an invaluable source of information, but its usefulness is limited by the available means of accessing and tailoring it to the needs of an individual. Long experience with text as a medium of conveying information allowed us to develop relatively effective methods of dealing with textual data. Unfortunately, the currently available techniques of accessing and processing video data are largely inadequate to the needs of its potential users. Hence video material remains a valuable but grossly untapped resource.

In the domain of video news sources, this problem is especially severe. Television news stations broadcast continuous up-to-the-minute information from around the globe. For any individual viewer, only small portions of this news stream is of interest, yet currently no methods exist which would allow him to filter and monitor only the interesting news.

In this dissertation, we demonstrate a solution to this problem by exploiting the repetitive character of the video news broadcast to create a story tracking algorithm. We observe that news stations often reuse the same video footage when reporting on the development of a story. We use this information to detect repetitions of the video footage and utilize this information for story tracking. To achieve this purpose in live video news broadcasts, we develop a real-time video sequence matching technique capable of identifying very short and only partially repeated sequences. We also

introduce improvements in existing temporal video segmentation methods, which allow us to more accurately detect short video shots.

The story tracking technique presented in this dissertation is complementary to existing textual topic detection and tracking methods and could be used in conjunction with them to improve the overall performance.

Acknowledgements

I am deeply indebted to my advisor and mentor, Professor John Gauch, for his guidance and inspiration during my career at the University of Kansas. His invaluable advice and direction offered me during countless hours of discussion, especially during the last few months of this work, were absolutely indispensable in the successful completion of this research.

I owe gratitude to Professor Susan Gauch, who first suggested the problem of story tracking as an interesting area of investigation, and guided me through the process of writing and presenting the dissertation proposal. Her constructive criticism and insightful observations expressed during numerous consultations in later phases of my work have greatly enhanced the quality of this dissertation. I am truly grateful to the members of my Ph.D. committee, Professors Jerzy Grzymala-Busse, Jerry James, Allard Jongman, and Costas Tsatsoulis for their comments during my comprehensive examination, which were very helpful in establishing the direction of my research, as well as remarks and corrections to this manuscript.

I would also like to thank my wife, Heather, for her assistance in creating the manual data for many of the experiments presented in here, as well as proof reading and editing the earlier revisions of this manuscript. More importantly, I am eternally grateful for her enduring love and unceasing words of encouragement which have carried me through the most difficult days of this work, when in moments of despair I was ready to give up. She alone knows how many times this dissertation was only a hair's width away from never being completed.

I will never be able to fully repay my parents, Lidia and Zdzislaw, whose love, devotion and often personal sacrifice created a family in which I was blessed to spend a happy and carefree childhood, form my character during adolescence, and later acquire the knowledge and skills needed for this work. Their encouragement and support ultimately allowed me to begin the graduate studies at the University of Kansas. I owe thanks to numerous other friends and family members, whose wisdom has guided me over many years, and whose valuable advice, as well as countless prayers undoubtedly contributed to the successful completion of this dissertation.

Finally – last but certainly not least – I would like to offer my thanks to the Lord, God. His eternal will – the origin of all things – allows all good things to come to fruition.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As the advent of print in the fifteenth century offered a widespread access to information and caused a proliferation of printed material, the invention of the television set, and later the Internet, initiated an explosion in the amount of video content available. And if the proverb “a picture is worth a thousand words” is right, the mass of information contained in this visual material should by far surpass that available through only textual sources.

In practice, however, the useful amount of information from any source depends on how easily that information can be accessed and trimmed to the needs of an individual. Many centuries of experience with text as a medium of conveying information allowed us to develop relatively effective methods of accessing and processing textual data. Unfortunately, the currently available techniques of accessing and processing video data are largely inadequate to the needs of its potential users. Hence, video material remains a valuable but grossly untapped resource.

This state of the matters is especially pronounced in the realm of news. Multiple television news channels, as well as various Internet news sources, broadcast continuous up-to-the-minute information from around the globe. Theoretically then, virtually anyone could have instantaneous access to the latest news at almost any time. In practice, this would require constant monitoring of all available news sources, which is humanly impossible. Moreover, continuous viewing of news sources would prove very time-inefficient, as a vast majority of information they contain is of little or no interest to the user and the ratio of truly new information in the news is fairly small.

Therefore, it is needed to develop methods of effective access to video news. These techniques should allow the human user to specify the scope of interest, and report to him only the news related to that interest. They should also eliminate all redundant material, and present only the actual new information. A system with such capability could find application not only in personal use, but also in areas such as national security, or automatic archive and library creation.

In this dissertation, we focus on the domain of television news broadcasts. As stated above, television news stories have a very repetitive character. Breaking news occurs only so often, yet news channels broadcast continuously. The rest of the regular programming has to be filled with information that has been reported earlier in previously recorded video footage. Furthermore, even when reporting on new story developments, the old footage is re-broadcast to provide context.

Hence, the continuous stream of video news broadcast contains a substantial amount of repeated material. Frequently however, repeated portions are not duplicated in their entirety. Rather, an event that was described at length in the morning news may be only briefly mentioned in the afternoon edition. Thus, the accompanying video material must be truncated. It is also conceivable that a video sequence will be extended in subsequent programs, for instance, an event described in headline news may be examined in detail later. Therefore, when searching for repeated video

material, one must account for such changes in length and composition of the relevant portions.

It is also important to note that, although the video footage may be repeated, it is often accompanied by new audio and closed captions. This happens when the same event is reported live by a different anchor person in a different news program, or even by the same anchor person using different words. Consequently, while examining news broadcast for repetitions, one must consider video signal separately from audio and closed captions.

Whatever the nature and composition of the repeated material, the fact that it is duplicated indicates that the different portions of the story containing it are most likely related. In our dissertation, we propose to use repeated video footage as an indication of the relationship between events reported in the news. We design, implement, and evaluate an algorithm for story tracking in video news broadcasts based on detection of repetitions in the video stream.

1.2 Problem Definition and Proposed Solution

In this dissertation, we aim to provide a solution for the problem of effective access to news video broadcast addressing the following scenario. A person watches the morning news edition, and with assistance of a computer identifies interesting stories. The story tracking system independently tracks the development of these stories throughout the day. In the evening, the user is provided with a summary of the relevant news.

To accomplish this goal, we must tackle a number of problems and provide solutions which will become the major components of the story tracking system. First, we must monitor a live video broadcast and divide it into shots, the basic units of video production. Second, we need to identify video footage that is reused by the news station to provide visual clues to the viewers. Third, we must devise methods of

building a story around the repeated footage. And finally, we need to create techniques of meaningful presentation of the story to the viewer.

In the process of dealing with these challenges, we touch on several areas of the broad domain of video retrieval. Therefore, before embarking on a detailed description of our story tracking techniques, we first give a brief overview of the research in the related fields. More detailed discussion is presented in the corresponding chapters.

1.3 Related work

Temporal Segmentation

The video stream which reaches viewers in the form of a television broadcast is a composition of video material coming from multiple sources and arranged by the producer or editor. The most basic components of the broadcast are sequences of video frames captured from a single camera, called shots. In the production process, video shots from different cameras recorded at different times, are combined into a single video sequence by means of shot transitions. Thus, the final product is a continuous video stream whose original structure is not directly available. The recovery of this structure by dividing the continuous stream of video frames into the original shots, is the fundamental step in any kind of processing video content, and is the subject of temporal video segmentation.

Temporal video segmentation methods aim to detect shot transitions inserted by the producer, and so detect the boundaries of the original shots comprising the video. Today, thanks to the advances in computer and video production technology, a number of editing effects, and hence, shot transitions are available. The majority of editing effects used can be classified into three basic categories: cuts, fades, and dissolves, but recently television stations employ an increasing number of more sophisticated computer-generated effects.

The field of video segmentation is relatively mature and has been researched for well over ten years. The resulting methods concentrate primarily on detection of the three basic types of transitions, and generally ignore the computer-generated effects. The fundamental idea of transition detection results from the assumption that certain characteristics of video are different during shot transitions than within the shots. Therefore, transition detection techniques select certain features of video frames and analyze their properties over time. If a characteristic pattern is recognized, shot transition is reported.

A variety of image and video sequence features, such as color histogram, edges, or motion have been used in a number of transition detection methods. Although the overall performance for simple transitions, such as cuts is excellent, the detection of gradual transitions, especially dissolves, certainly leaves room for improvement.

Considering that television news stations tend to use more sophisticated transition effects, we seek to improve on the existing segmentation methods. In Chapter 2 we present a real-time temporal segmentation algorithm based on color moments as compact video frame feature. The evaluation experiments demonstrate that our approach achieves cut and fade detection performance equivalent to other methods presented in literature, and is superior in detection of dissolves.

Video Retrieval

As indicated earlier, we are interested in detecting repetitions in the live video news broadcast in order to use this information in our story tracking technique. Though our goal is to detect exact copies of the same video footage, this task is closely related to assessing video sequence similarity. Much of the research in the field has focused on search for conceptually similar material: given, say, an image or video clip of a sailing boat, any clips of sailing might be regarded as a match. In this work, we are primarily interested in a different type of similarity, which may be described as matching of co-derivatives. Thus, two distinct types of video similarity may be recognized:

1. Semantic similarity. Two video sequences are semantically similar if they represent or describe the same or similar concept.
2. Co-derivative similarity. Two video clips are co-derivatives if they have been derived from the same original video sequence.

Semantic similarity has been the focus of considerable research efforts generally classified as content-based video retrieval. Matching and retrieval of co-derivatives has been explored in other domains, such as text, but the problem has received little attention in the realm of multimedia.

The semantic video retrieval is generally shaped by the cognitive gap separating humans from machines. Conceptual content of video, intuitively recognized by the humans, is inherently difficult to obtain by computers. As a result, existing semantic video retrieval systems resort to defining human information need in terms of lower level machine-accessible features, such as colors, textures, etc. Alternatively, manual annotations of video may be used, but they do not scale well with the rapid expansion of available video material.

In certain domains of video retrieval, such as news broadcasts, additional information regarding the semantic meaning is contained in metadata, such as accompanying audio or closed captions. With the help of automated speech recognition methods, the audio signal may be transcribed into a textual form, and as such may be analyzed using textual information retrieval methods. Since these methods have been researched for many years, they often provide an attractive alternative to relatively new techniques of visual information retrieval.

In the last few years, initial efforts have begun to derive semantic concepts from video by applying certain machine learning techniques. The methods developed have been shown to work reasonably well in limited contexts, but they remain in their very early stages, and do not offer viable solutions to more general problems of semantic video retrieval.

While semantic video retrieval has been researched for a number of years, matching and retrieval of co-derivative video material is a new field, and so far has received little attention in the research community. Recently, some work has been done to identify commercials in television broadcasts. Introductory research into recognizing co-derivative video sequences from different sources has also been reported. Detection of co-derivative similarity could find a number of applications, such as copyright management, video compression, or – as shown in this dissertation – news story tracking.

In this dissertation, we are interested in detecting repeated video sequences in live video news broadcasts. This task offers a number of challenges, such as recognition of similarity in the presence of on-screen captions, detection of repetitions between very short sequences, as well as identification of only partially matching video clips. In Chapter 3, we describe our method of repeated video sequence detection, which addresses these problems. We demonstrate experimentally that our technique successfully identifies repetitions in a typical television news broadcast.

Story Tracking

The main goal of this dissertation is to design and develop methods of story tracking in live video news broadcasts. The problem of story tracking is relatively new, and was first posed in the domain of textual information retrieval as part of the Topic Detection and Tracking (TDT) initiative in 1997. According to the original authors:

> *"TDT is a research program investigating methods for automatically organizing news stories by the events that they discuss. TDT includes several evaluation tasks, each of which explores one aspect of that organization – i.e., splitting a continuous stream of news into stories that are about a single topic ("segmentation"), gathering stories into groups that each discuss a single topic ("detection"), identifying the onset of a new topic in the news ("first story detection"), and*

exploiting user feedback to monitor a stream of news for additional stories on a specified topic ("tracking")."

As in any new field of study, the general taxonomy is not fully developed and certain concepts are not precisely defined. In addition, the different tasks of TDT are very closely intertwined. As a result, the problem of story tracking addressed in this dissertation, as defined earlier in this chapter, spans two TDT tasks, i.e. detection and tracking. We also believe that the phrase "story tracking" is more suitable for our work than "topic tracking", and we provide detailed definitions of our understanding of the basic concepts of event, story, and topic in the introduction to Chapter 5.

In the years since the inception of TDT, a number of topic detection and tracking techniques have been proposed. Their performance on textual news sources has been found to be satisfactory, but leaves room for additional improvement. Topic detection and tracking does not restrict its attention to only textual news sources, and its methods can be applied to the transcripts of video news broadcasts obtained from closed captions or through automated speech recognition. However, these methods rely entirely on text, and completely ignore the rich layer of visual information present in the video stream.

In recent years, initial research has been done on incorporating elements of visual information into organization of video news. Visual features have been used to improve alignment of textual transcripts of news broadcasts by means of shot classification. Basic image features of video background have also been used to identify scene location in a limited database. Overall, however, the visual content of news broadcasts is vastly underutilized, and has never been exploited for the purposes of story tracking.

In this dissertation, we demonstrate that it is possible to use only visual content of news broadcasts in order to effectively track stories over time. In Chapter 4, we describe the design, implementation and evaluation of a video-based story tracking system.

1.4 Organization

This dissertation is organized as follows. Chapter 2 presents our temporal video segmentation method. In this chapter, we first review shot detection techniques available in literature. Then we discuss improvements introduced in this work, and evaluate the performance of our method. Finally, we compare our results with those reported by other researchers, and demonstrate the superiority of our technique.

Chapter 3 is devoted to the problem of repeated sequence detection. This chapter starts with a brief discussion of the specific challenges involved in detecting repetitions for purposes of story tracking in video news. Subsequently, we present a number of video sequence similarity metrics and assess their suitability for detection of very short and only partially repeated video clips. We then introduce two basic repeated sequence detection algorithms, and show that they could not be applied to real-time detection. We follow by presenting a heuristic shot filtering technique, which greatly reduces the average execution time of repetition detection. At the end of the chapter we experimentally verify the effectiveness of our approach.

Chapter 4 introduces our story tracking method based on repetitions of video material. In this chapter, we first define the fundamental concepts of story tracking in video. We then present our algorithm of story tracking using repeated shot detection, and demonstrate its effectiveness experimentally. Manual and automatic story tracking are performed on a typical video news broadcast and their results are compared. This chapter also discusses potential improvements to story tracking based on shot classification and introduces two approaches to story presentation.

Finally, Chapter 5 presents the summary of the research contributions discussed in this dissertation and enumerates suggestions for future research in this area.

Chapter 2

Temporal Segmentation

2.1 Introduction

In the production process of any video news broadcast, the producer or editor combines and arranges video material coming from multiple sources. The original video sequences captured by different cameras at different times – some live, some prerecorded – are blended into a single long sequence of video frames which is broadcast by a television station. Thus, when the news broadcast reaches its viewers, its original structure is not obscured by a variety of editing effects. The recovery of this structure, i.e. the task of dividing the continuous stream of video frames into the original components, is the fundamental step in any kind of video content processing, and is the subject of temporal video segmentation, also known as video shot detection or transition detection. Thus, temporal video segmentation is an important element of story tracking in video news broadcasts.

In this chapter, we present the problem of temporal video segmentation, and discuss specific challenges pertaining to story tracking in the domain of video news. We review certain existing solutions, and propose an effective algorithm of shot detection

in video news broadcasts. First, however, we provide an overview of terminology used in the field.

The task of temporal video segmentation is to recover the original structure of a video stream as a composition of shots. A ***shot*** is defined as a sequence of successive video frames taken from one camera. In the production process, individual shots are combined to form a video sequence using a variety of editing techniques. Any sequence of frames resulting from video editing is called a *shot transition*.

Today, thanks to the advances in computer and video production technology, a number of editing effects and, hence, shot transitions are available. All of them can be organized into a few basic classes [Lie01a, Ham95] based on 2D image transformations applied. The following classes may be distinguished:

i. ***Identity Class***: Neither of the two shots involved is modified, and no additional edit frames are added. Only hard cuts qualify for this class.

ii. ***Spatial Class***: Some spatial transformations are applied to the two shots involved. Wipes, page turns, slides, and iris effects fall into this category.

iii. ***Chromatic Class***: Some color space transformations are applied to the two shots involved. These include fade and dissolve effects.

iv. ***Spatio-Chromatic Class***: Some spatial as well as some color space transformations are applied to the two shots involved. All morphing effects fall into this category. Note that in practice often all effects in the spatial class in principle fall into the spatio-chromatic class since some chromatic transformations are always applied at the boundary between the pixels of the first and second shot such as anti-aliasing, smoothing or shading operations.

An alternative shot transition classification scheme presented in [Lie01b] is based on the spatial and temporal separation of the two shots involved (see Table 1). For instance, for hard cuts and fades the two sequences involved are temporally and

spatially well-separated. Their detection comes down to identifying that the video signal is abruptly governed by a new statistical process, as in the case of hard cuts, or that the video signal has been scaled by some mathematically simple and well-defined function, as in the case of fades. For wipes, the two video sequences involved in the transition are spatially well-separated at all times. This is not the case for dissolves. At any time, the two video sequences are temporally as well as spatially intermingled, requiring dissolve detection algorithms to deal with a two source problem.

Type of transition	The two involved sequences are	
	spatially separated	temporally separated
Hard cut	Yes	Yes
Fade	Yes	Yes
Wipe, Door, Slide	Yes	No
Dissolve	No	No

Table 1 Transition classification scheme for transition detection

Although several kinds of shot transitions are available, only three basic types-cuts, fades, and dissolves-constitute a large majority of all transitions seen in real world video streams. Each of those transition types corresponds to a production process which can be modeled mathematically. We can consider a video sequence as a three dimensional intensity function $I(x,y,t)$, which assigns an intensity value to every pixel in every video frame. During a transition, the intensity function can be viewed as a superposition of the functions corresponding to the two shots involved: $I_1(x,y,t)$, $I_2(x,y,t)$. If we assume that the function arguments are continuous, we can describe each transition type by the mathematical model of the intensity function during the transition sequence:

i. ***Cut*** is a direct concatenation of two shots not involving any transitional frames, and so the transition sequence is empty.

ii. ***Fade*** involves only one shot and is a sequence of frames $I(t,x,y)$ of duration T resulting from scaling pixel intensities of the sequence $I_1(t,x,y)$ by a

temporally monotone function *f(t)*:

$$I(t,x,y) = f(t) \cdot I_1(t,x,y), \quad t \in [0,T]$$

iii. ***Dissolve*** is a sequence *I(t,x,y)* of duration T resulting from combining two video sequences $I_1(t,x,y)$ and $I_2(t,x,y)$, where the first sequence is fading out while the second is fading in:

$$I(t,x,y) = f_1(t) \cdot I_1(t,x,y) + f_2(t) \cdot I_2(t,x,y), \quad t \in [0,T]$$

These models allow us to predict certain characteristics of the intensity function during shot transitions, and became the foundation of the research in temporal video segmentation.

An overview of the related work in this area is presented in section 2.2. The remainder of this chapter is organized as follows. Section 2.3 discusses the challenges of shot detection in the domain of video news. In section 2.4, we present the evaluation methodology used to assess the performance of the transition detection techniques. We propose and evaluate two different transition detection algorithms in sections 2.5 and 2.6. Section 2.7 contains a summary of our findings and our conclusions.

2.2 Related Work

The field of video segmentation is relatively mature and has been researched for over ten years. Numerous algorithms have been created and their performance is acceptable for general purpose video segmentation. However, no single technique exists that would provide 100% segmentation accuracy for all types of transitions in all kinds of video sequences. Presented below is a brief summary of the most effective segmentation methods with the emphasis on those especially suitable for the task of story tracking in video.

The fundamental idea of transition detection stems from the assumption that certain characteristics of video are different during shot transitions than within the shots. In

the previous section, we showed mathematical models of the effects of the three basic transitions on the pixels in the video frames. These effects are reflected in the global features of video frames, such as color composition, number of edges, or motion continuity. Researchers have exploited this fact to create detection methods for the basic types of transitions. The existing detection techniques use one or more video frame features and analyze their properties over time. If a pattern typical of one of the transition types is recognized, then the transition is reported. Since the temporal feature patterns are quite different for different types of transitions, most of the detection techniques are tailored to a specific type of transition. The following sections present an overview of the detection methods.

2.2.1 Cut detection

A hard cut produces a temporal discontinuity in the video stream, which manifests itself as a radical change in the time series of a video frame feature. Thus cuts may be discovered by looking for large difference in the frame feature between consecutive frames. Existing algorithms use this fact to detect hard cuts by identifying isolated peaks in the feature difference time series.

The most effective algorithms use color histograms [Nag92, Lie99, Gar00, Tru00a], edge pixels [Zab95, Zab99] and motion [Aku92, Dai95, Lup98, Sha95] as video frame features. Once the feature is selected, cut detection consists in identifying feature differences large enough to be considered cuts. This is accomplished by selecting a discontinuity threshold. A cut is then declared each time the feature difference exceeds the given threshold.

A common problem with having only one global threshold is that it is impossible in practice to find one value that fits all kinds of video material [Lie99]. Therefore different techniques have been used to provide adaptive threshold [Tru00a] that adjusts to the type of video material currently being processed.

In general, the histogram-based methods prove to be simple, yet very effective. The more sophisticated edge- or motion-based techniques do not offer significant performance gains [Lie01a]. The accuracy and performance of various hard cut detection algorithms has been thoroughly studied and can be found in [Bor96, Dai95, Gar00].

2.2.2 Fade detection

Fades correspond to a gradual transition from a given shot to a monochrome (usually black) screen or vice versa. Two effective approaches to detecting this type of transition are described by Lienhart [Lie01a]. One is based on the observation that fades show as a gradual and steady decrease/increase in the color variance of video frames. The other uses the observation that during a fade edges of objects in the frame tend to become weaker (fade-out) or stronger (fade-in). The former was introduced independently by Lienhart [Lie99] and Alattar [Ala97], and later combined and extended by Truong at al. [Tru00a]. The latter is presented in [Zab95, Zab99]. Their comparison has been performed and its results reported in [Lie99, Lup98]. Both of these studies show that the edge-based method did not perform as well as the color variance approach. It has also been demonstrated that fade detection using the color variance method yields very high accuracy.

2.2.3 Dissolve detection

The problem of dissolve detection is by far the most challenging in the domain of video segmentation due to the lack of either spatial or temporal separation of the two shots involved. The issue is further complicated by motion in the shots being combined. The effects of motion and dissolve on several characteristics of video frames are virtually indistinguishable.

Lienhart [Lie01a] presents a number of dissolve detection techniques. The first group of techniques attempts to recognize dissolves by examining temporal change of pixel colors. These methods rely on the observation that during a dissolve with little or no

motion most of the pixels change approximately linearly over time [Ham95]. Unfortunately, this approach is very sensitive to motion, and in the presence of it behaves poorly.

The second group uses changes in the time series of video frame color variance. Alattar [Ala93] noticed that the color variance curve during an ideal dissolve has a parabolic shape. This observation has been further exploited in [Fer99, Tru00a, Tru00b] and proved to be reasonably effective.

Lienhart also reports two edge-based methods introduced and examined in [Zab95, Zab99, Lie99]. He states, however, that these techniques produce an unacceptably high false alarm rate.

Finally, a multi-resolution pattern recognition approach was introduced in [Lie01b]. Lienhart uses a neural network method, which is reported to be very effective on a wide range of dissolves. It is shown to be very effective on even difficult and unusual dissolve sets. However, this method is computationally intensive and requires extensive training which is prohibitive in the processing of continuous, live video broadcasts.

2.2.4 Compressed domain methods

With recent popularization of digital video sources (DVD, digital cable, etc.), some researchers proposed methods of shot transition detection in the compressed (MPEG) domain. In comparison to detection in uncompressed domain, segmentation of MPEG video streams has the advantage of motion estimation already encoded in the stream. Performing such estimation for purposes of segmentation of a live video stream is currently too computationally expensive.

An evaluation of currently available MPEG segmentation techniques can be found in [Kop98, Kas98]. In this work we elect not to consider compressed domain segmentation methods. This choice offers freedom from the intricacies of various

video compression formats, as well as independence of the encoding quality offered by different encoders.

2.3 Temporal Segmentation for Video News Broadcasts

Video news broadcasts differ considerably from other types of video. They typically contain a limited number of shot types, such as anchor person, studio, or report from the field. Often the core of the news content is communicated verbally. Therefore, the stations strive to make the broadcast more visually appealing, by introducing sophisticated effects, such as moving or morphing borders, computerized transitions, picture-in-picture effects with icons corresponding to currently discussed events, as well as multiple camera shots in one frame. In addition, they frequently display a caption bar at the bottom of the screen to provide textual cues as to what is being discussed, as well as brief summaries. These specialized effects make transition detection a difficult task. Although transitions between live coverage (studio, anchors, etc.) are usually relatively simple (i.e. cuts), this is not the case for prerecorded material. Such footage is combined with the rest of the broadcast primarily by means of dissolves. Since detection of repeated video footage is the primary focus of this work as the foundation for story tracking, accurate dissolve detection is very important to us. The importance of accurate transition detection is emphasized by the fact that repeated news clips are often short and constitute only a small portion of the overall video. In addition, the clips of interest tend to contain considerable motion, as they are often filmed by hand at an off-site location.

Transition detection is further complicated by on-screen captions and the live broadcast indicator. Since these elements tend to be displayed in bright colors and sharp contrast in order to stand out from the rest of the screen content, they alter significantly the color composition of video frames. Additionally, the content of the bottom caption changes frequently during shots, causing abrupt changes in color

composition. Similarly, the live footage indicator often appears or disappears in the middle of gradual transitions, such as when the video transitions from an anchor person to some prerecorded footage. These effects tend to obscure changes in video frames when an actual transition occurs, and therefore make the temporal segmentation task more difficult.

Modern news video broadcasts also abound in computer generated transition effects, aimed at making the visual content more appealing. Such effects come in a large variety of flavors, and there is no good general method of consistently detecting them all. Fortunately this does not affect our research significantly, as these transitions occur mainly in promos and commercials, not in the actual news footage. This allows us to largely ignore them in this work.

All of the above discussed aspects of video news make the task of temporal segmentation in this domain challenging. Therefore, in our research on story tracking we needed to create transition detection methods capable of working effectively in video news. Since transition detection is only the first step in the story tracking process, which needs to be performed on a live news broadcast, it is important that our techniques be very fast.

As described earlier (see section 2.1), temporal segmentation relies primarily on choosing a feature or set of features of video frames and analyzing their behavior over time in order to detect patterns characteristic of the three main types of transitions. In this work, we chose the three primary color moments (mean, standard deviation and skew) as the frame feature. The mean, standard deviation, and skew of a color image are calculated as follows.

$$M(t,c) = \frac{1}{N}\sum_{xy} I(t,x,y,c)\,, \tag{1}$$

$$S(t,c)^2 = \frac{1}{N}\sum_{xy} \left[I(t,x,y,c) - M(t,c)\right]^2\,, \tag{2}$$

and

$$K(t,c)^3 = \frac{1}{N}\sum_{xy}\left[I(t,x,y,c) - M(t,c)\right]^3 . \quad (3)$$

Color moment frame representation is the basis of two different temporal segmentation algorithms developed in this work. The first method, the cross-difference algorithm, relies on moment differences between video frames to detect all types of transitions. The second technique uses mathematical models of different transition types. The cross-difference algorithm was developed first and proved to be effective in cut detection. However, its performance for gradual transitions was unsatisfying. Therefore, to improve the accuracy of gradual transition detection, we developed the transition-model approach extending previous work in the field. Both methods are discussed in detail in sections 2.5 and 2.6, but first we describe the evaluation methodology.

2.4 Evaluation Methodology

Experimental Data

In order to evaluate performance of our temporal segmentation algorithms we recorded a one-hour block of typical news broadcast obtained from CNN News. The video clip was compressed using Windows Media Encoder 9.0 and saved in the Windows Media Format with video size of 160 x 120 pixels.

The video clip was then manually inspected and annotated with shot transitions. Four different types of transitions were distinguished: cuts, fades, dissolves, other transitions. The first three types follow their respective definitions presented earlier, and the last category is comprised of all transitions that do not fit any of the previous types. For every transition the start and end frames were recorded, where the start and end frames are defined as the first and last video frame for which the effects of the

transition are discernable by the human eye. In the case of cuts, the frame before the cut was taken as the start frame, and the frame after, as the end frame.

The video clip consisted of several shots of news content, such as studio shots, outdoors shots, anchor person shots, as well as several commercials and promotional clips. The clip contained a total of 971 transitions, of which 618 were cuts, 84 were fades, 189 dissolves, and 70 were other transitions. Clearly the sharp cut is the dominant type of transition used, but there is a surprisingly high number of dissolves.

In a few cases of commercials and promos the video material was entirely computer generated, and therefore could not be objectively divided into shots and transitions. We decided to exclude this part altogether, because segmentation results from this material would be highly subjective.

Finally, in order to reduce the impact of on-screen captions and the live footage indicator, we analyzed the video source to determine their location and size across the entire clip. We determined that the bottom caption – if present – occupies, at most, the bottom 25% of the screen. We also found that the live indicator always appears in the top left corner of the screen, and is limited to at most 10% of the screen size. Given this information, we chose to exclude these areas of the screen from frame feature calculation entirely, regardless of whether a given frame contained an on-screen caption or live indicator.

Evaluation Methodology

Automated transition detection was performed on the same one-hour segment of video broadcast. Results of the automated detection were compared to those of manual segmentation in order to find matching pairs. A pair of transitions could be considered matching if they are of the same type and their start and end frames are the same. Such a definition, however, is often too strict, as automatically detected transitions, especially gradual (i.e. fades, dissolves, and other) may differ in length from the manually annotated. Such a difference remains without significant impact on the quality of temporal segmentation. Therefore, we adopted a modified definition of

matching transitions. We say that two transitions match if they are of the same type and if they overlap.

Matching transitions are found by searching for a manual transition (*mt*) of matching type for every automatically detected transition (*at*). If *at* overlaps more than one *mt*, then we choose the manual transition with the same type, and report a match. If the transition types do not match, we pick the first manual transition which overlaps, and declare a mismatch. Finally, if no *mt* overlaps the current *at*, we declare a false alarm. Once this process is completed, all remaining manual transitions that have not been matched are reported as missed.

We then evaluate the transition detection performance in terms of the standard recall and precision measures defined as follows:

$$recall = R^x = \frac{N^x_{correct}}{N^x_{correct} + N^x_{miss}} \cdot 100\% \tag{4}$$

$$precision = P^x = \frac{N^x_{correct}}{N^x_{correct} + N^x_{false}} \cdot 100\%, \tag{5}$$

where

$$N^x_{correct} = |\Theta|,\ where\ \Theta = \left\{S^x_i, i \in \{1,...,k^x_a\} \mid \exists j \in \{1,...,k^x_m\}\ and\ S^x_i \cap S^x_j \neq \varnothing\right\} \tag{6}$$

$$N^x_{miss} = |\Theta|,\ where\ \Theta = \left\{S^x_i, i \in \{1,...,k^x_a\} \mid \forall j \in \{1,...,k^x_m\}\ S^x_i \cap S^x_j = \varnothing\right\} \tag{7}$$

$$N^x_{false} = |\Theta|, where\ \Theta = \left\{\begin{matrix} S^x_i, i \in \{1,...,k^x_a\} \mid S^x_i \cap \Phi = \varnothing \\ and\ \forall j \in \{1,...,k^x_m\}\ S^x_i \cap S^x_j = \varnothing \end{matrix}\right\} \tag{8}$$

In addition to these standard performance measures, we introduce a *utility function*, which aggregates recall and precision into a single performance estimator. The utility function is a weighted sum of recall and precision, and controls the desired tradeoff between their values, as presented in (9).

$$utility = \alpha \cdot recall + (1 - \alpha) \cdot precision \tag{9}$$

This aggregated value, which is often referred to as *f-measure* in the information retrieval literature, allows us to objectively determine what set of parameters gives the best performance.

2.5 Moment cross-difference algorithm

Overview

In view of our initial assumptions about video transitions in news broadcasts, we first developed a transition detection algorithm for all types of transitions based on the following observations. Generally, two frames within one clip, especially in close proximity to each other, are very similar in appearance. Two frames from two different adjacent clips are considerably different.

We can also assume that different frames have different sets of moments, and similar frames have similar sets of moments. Although this assumption does not hold true in all cases, it is true for an overwhelming majority of practical situations. Therefore, in a simple approach we compare color moments of every two consecutive frames and declare a transition if they differ sufficiently. We could select a difference threshold, compute the difference between the corresponding sets of moments, and declare a transition if the difference exceeds the threshold. Depending on the two clips in question, the difference may be very large or fairly small, which makes it difficult to choose a single difference threshold for an entire video broadcast.

This method could work for cuts, but would be inadequate for gradual transition detection. By the very definition of gradual transitions, consecutive frames during the transition change gradually. Hence, the differences between them are slight, and are only apparent for the frames before and after the transition. In order to identify such transitions, we could compare frames a certain distance away from each other.

However, if the shots themselves contain motion or other rapid changes, then frames sufficiently distant from each other will exhibit substantial differences in moments, and therefore will trigger transition detection. This would lead to a high rate of false alarms.

To account for this, but still recognize gradual transitions, we created a detection algorithm which attempts to compensate for the changes in color moments due to motion in shots. The following section describes the algorithm.

Algorithm

We rely again on the fundamental assumption that the frames of the shot before a transition are similar between themselves, as are the frames of the shot after the transition. In the presence of motion in the shots, the similarities are somewhat diminished, and the frames taken from the same shot may be different. Considering this, we attempt to find pairs of consecutive frames which differ substantially from each other. To achieve this, we define the concept of cross-difference as the average moment difference of every pair of frames within a certain window spanning the current frame less the average moment difference of every pair of frames to the left of the current frame and to the right of the current frame.

$$
\begin{aligned}
&cross\,difference = \sum_{i=f-w,\, i\neq f}^{f+w} \sum_{j=i}^{f+w} m_{ij} d_{ij} \\
&where \\
&m_{ij} = \begin{cases} 1 & if\ i < f\ and\ j > f \\ -1 & otherwise \end{cases} \\
&d_{ij} = absolute\,color\,moment\,difference\,between\ \ frames\ i\ and\ \ j
\end{aligned}
\tag{10}
$$

Thus, if little motion is present, and the frames to the left (right) of the current frame are very similar, but the frame pairs from the left and right are considerably different, the value of cross-difference is high. However, if motion causes frames to differ

significantly on either side of the current frame, the value of cross-difference will be lowered accordingly.

In order to detect transitions, we compute a cross-difference for every frame in video, using a window of certain size *2w + 1*. If so computed cross-difference is maximal within the window and exceeds a predefined threshold (*t*), then we declare a transition. This algorithm is presented in pseudo code in Figure 1.

Generic Transition Detection Algorithm

```
Function CrossDifference(frame)
    crossDiff = 0.0;
    ForEach pair of frames f1 and f2 in [frame – wndwSize, frame + wndwSize]
        diff = AbsMomentDiff(f1, f2);
        If (f1 <= frame And f2 <= frame) Or (f1 >= frame And f2 >= frame)
            crossDiff = crossDiff - diff;
        Else
            crossDiff = crossDiff + diff;
        EndIf
    EndFor
    return crossDiff / numberOfPairs;
EndFunction

Function CalculateCrossDifferences
    ForEach frame in VideoClip
        frameDiff[frame] = CrossDifference(frame);
    EndFor
EndFunction

Function DetectGenericTransitions
    ForEach frame in VideoClip
        If frameDiff[frame] is maximum in [frame – srchWndwSize, frame + srchWndwSize]
            If frameDiff[frame] > MinCrossDiffThreshold
                genericTransitions[frame] = True;
            Else
                genericTransitions[frame] = False;
            EndIf
        EndIf
    EndFor
EndFunction
```

Figure 1 Generic Transition Detection Algorithm

Experimental Results

We first evaluate the overall performance of the algorithm irrespective of the types of transitions. In this case, any automatically detected generic transition can match any

type of the manually detected transitions. Recall is therefore defined as the ratio of the number all matching generic transitions to the number of all manually detected transitions, and precision – as the ratio of the number of matching generic transitions to the number of all automatically reported transitions.

$$recall = R^{generic} = \frac{N_{correct}^{generic}}{N_{correct}^{generic} + N_{miss}^{generic}} \cdot 100\% \tag{11}$$

$$precision = P^{generic} = \frac{N_{correct}^{generic}}{N_{correct}^{generic} + N_{false}^{generic}} \cdot 100\% \tag{12}$$

Although the algorithm is controlled by two parameters, i.e. window size (w) and cross-difference threshold (t), its performance is practically independent of the first parameter. This is illustrated in Figure 2, which shows the recall and precision as a function of the window size. Clearly, the utility function remains essentially flat with increasing window size. In view of this fact, we chose $w = 5$ for further experiments.

Consequently, performance of the algorithm depends only on the cross-difference threshold. The value of this parameter is used globally for the entire video clip. A graph of recall and precision as a function of the cross-difference threshold is presented in Figure 3. A typical recall vs. precision tradeoff is clearly visible. As the threshold increases, so does precision at the expense of recall. For the extreme value of the threshold (10.0) the algorithm achieves over 95% precision, but recall drops to a mere 35%. Conversely, for a threshold equal to 0.0, recall approaches 95%, but precision decreases to almost 10%.

The algorithm never achieves 100% recall due to the requirement that the cross-difference at a given frame was maximal within a certain search window. Thus, if multiple transitions occur in quick succession, such that the distance between them is less than the size of the search window, only one of them – the one with maximal cross-difference – will be reported.

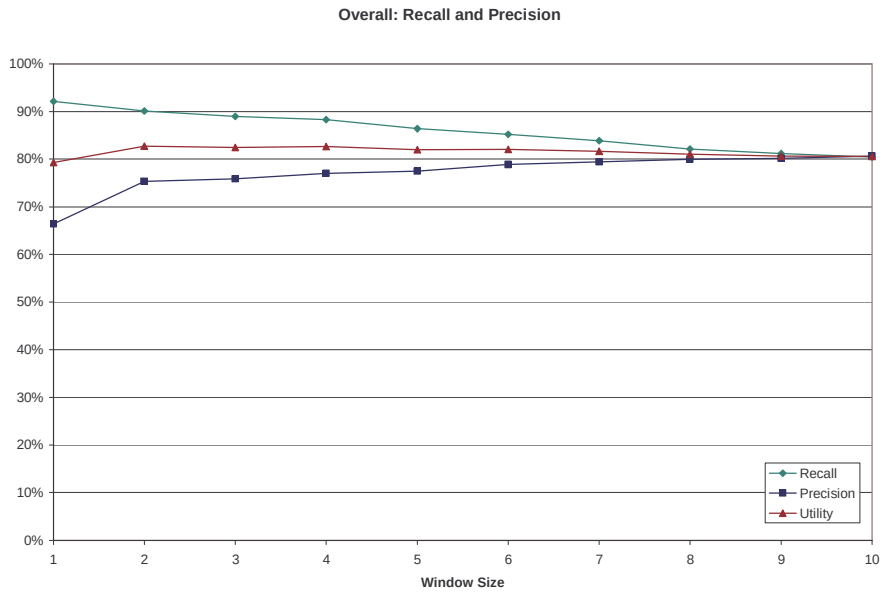

Figure 2 Recall and precision of generic transition detection as a function of the window size

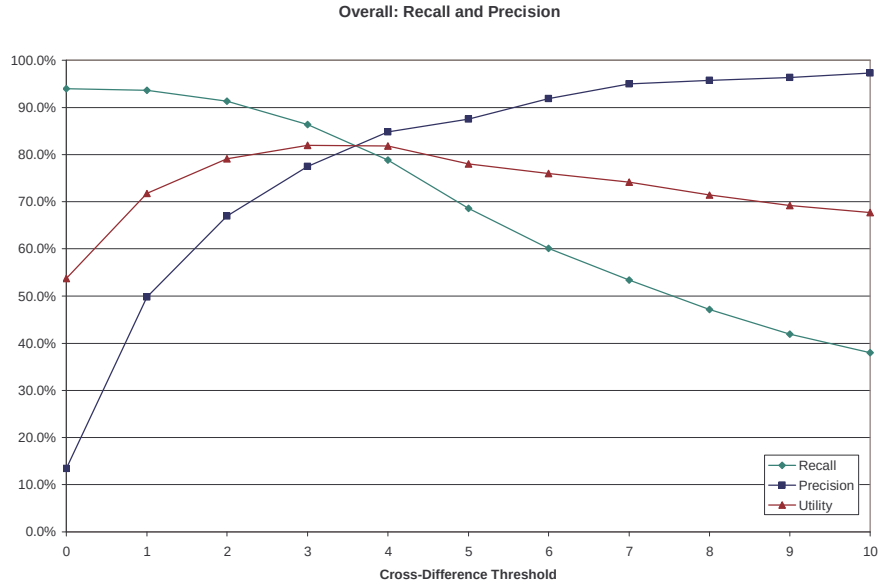

Figure 3 Recall and precision of generic transition detection as a function of the cross-difference threshold

In order to learn more about the performance of the algorithm for different types of transitions we conducted another experiment in which we reported detection results

for cuts, fades, and dissolves separately. These results will serve as a baseline performance for evaluation of other detection methods.

Since the manually detected transitions have been annotated as cuts, fades, and dissolves, we can determine cut (or fade, or dissolve) recall as the ratio of the number all automatic transitions matched to manually annotated cuts (or fades, or dissolves) to the number of all manually annotated cuts (or fades, or dissolves).

$$recall_{cut} = R^{cut} = \frac{number\ of\ detected\ cuts}{number\ of\ all\ cuts} \tag{13}$$

$$recall_{fade} = R^{fade} = \frac{number\ of\ detected\ fades}{number\ of\ all\ fades} \tag{14}$$

$$recall_{dissolve} = R^{dissolve} = \frac{number\ of\ detected\ dissolves}{number\ of\ all\ dissolves} \tag{15}$$

Precision, however, cannot be directly evaluated, as the algorithm does not differentiate between types of transitions, and hence it is impossible to determine how many cuts (or fades, or dissolves) have been falsely reported. In order to present some measure of precision we resorted to estimation. We took the total number of false alarms reported by the algorithm and divided it among the different transition types in proportion to their share in the total number of transitions. We therefore assume that the distribution of the different types of false alarms is the same as the distribution of the types of transitions in the clip.

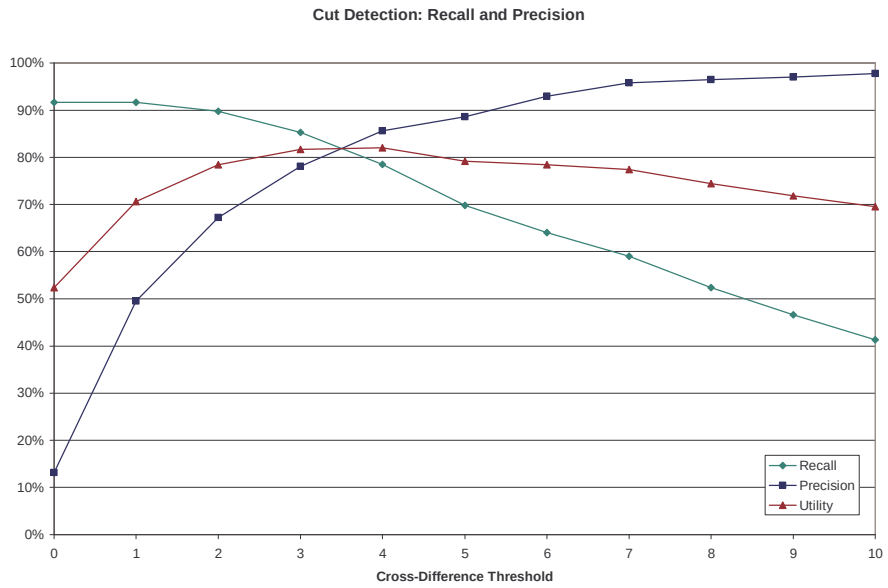

Figure 4 Recall and precision of the generic transition detection as a function of cross-difference threshold for cuts

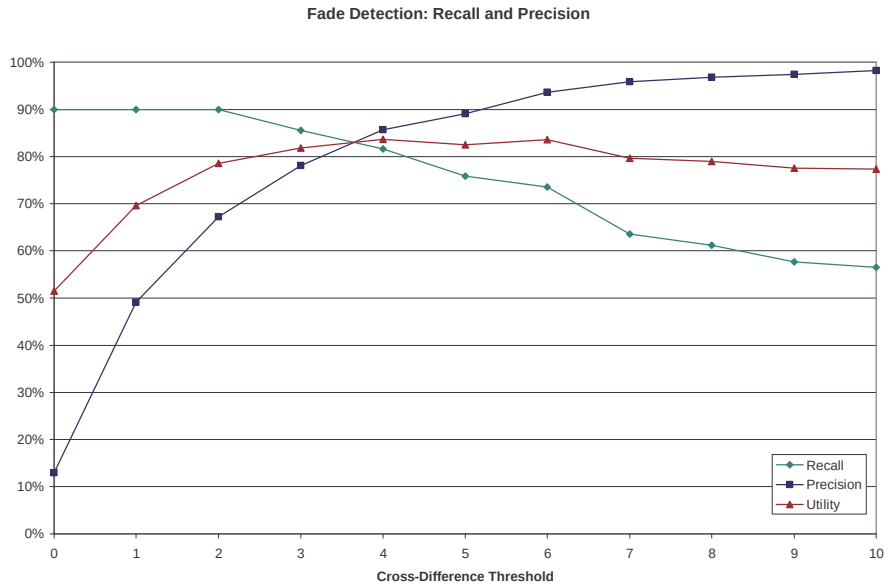

Figure 5 Recall and precision of the generic transition detection as a function of cross-difference threshold for fades

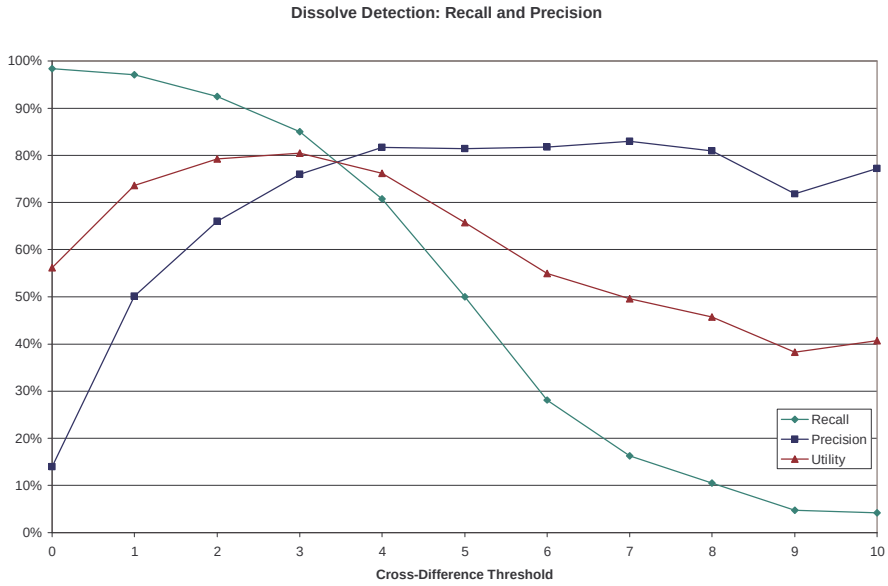

Figure 6 Recall and precision of the generic transition detection as a function of cross-difference threshold for dissolves

The graphs above show the algorithm's recall and precision as functions of the cross-difference threshold. Again, the typical trade-off between these two performance measures is apparent.

Since cuts are by far the most frequent type of transitions in the video material used, the detection performance for this type follows the overall trend. As shown on the graphs above (Figure 4, Figure 5, and Figure 6), a reasonably high recall and precision of around 80% can be achieved with cross-difference threshold between 3.0 and 4.0. As the threshold increases past this point, so does precision at the expense of the drop in recall. The utility function indicates that any threshold value between 3.0 and 4.0 yields the same overall performance.

The algorithm's performance for fades and dissolves follows the same pattern. The highest values of the combined utility are reached for threshold between 3.0 and 4.0. The algorithm detects fades with over 80% recall and precision, but performs worse in dissolve detection, achieving below 80% recall and precision. In addition, the precision curve of dissolve detection levels off just above 80%, and further increasing

the cross-difference threshold does not improve precision. This demonstrates that the cross-difference is not a sufficient dissolve indicator.

Analysis

In this section, we analyze the results of the experiments performed, to obtain additional insight into the strengths and weaknesses of the cross-difference algorithm.

The algorithm performs well detecting cuts in the absence of significant motion. This situation is demonstrated in Figure 7. As shown on the graph of color mean (Figure 8), a clear discontinuity is present when the cut occurs. The same discontinuity manifests itself as a narrow peak in cross-difference around the cut frame (Figure 9).

Figure 7 Simple cut with little motion

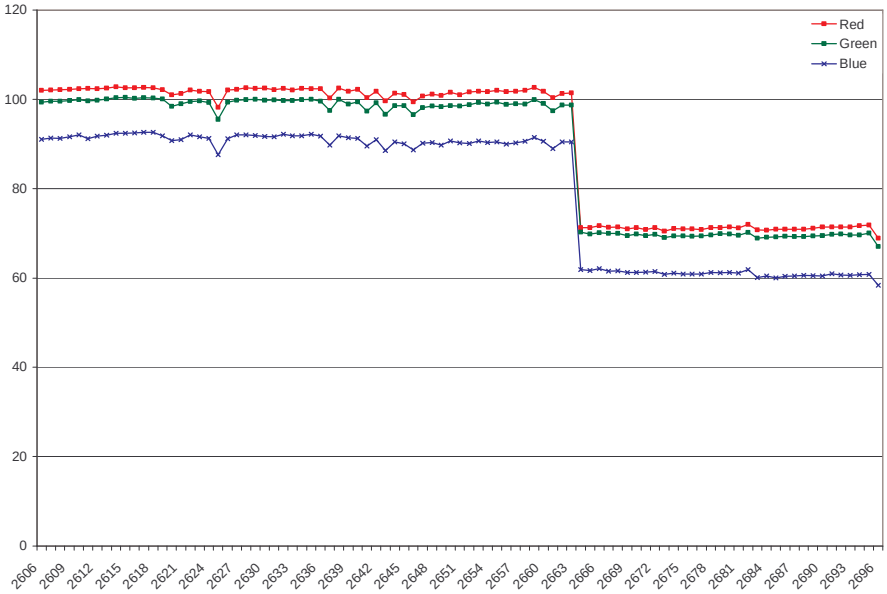

Figure 8 Mean curve for a simple cut with little motion

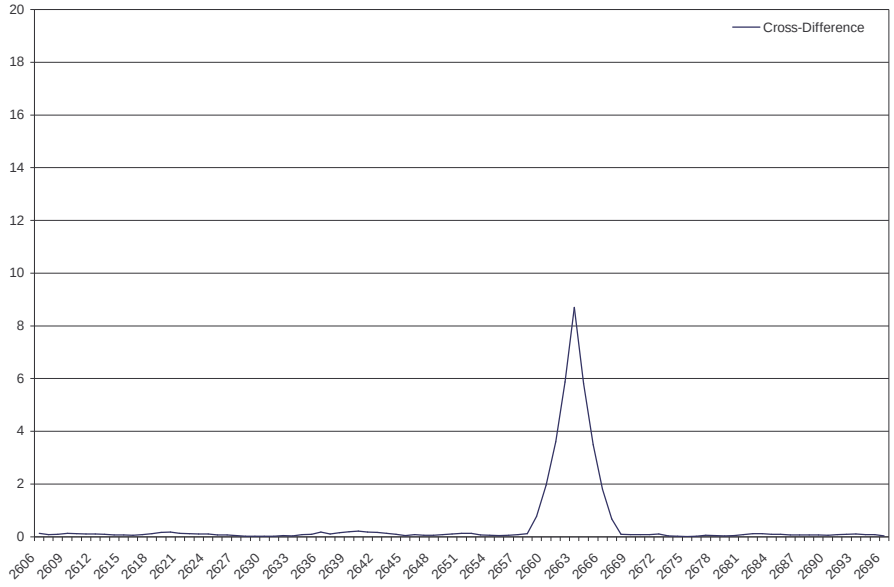

Figure 9 Cross-difference curve for a simple cut with little motion

Gradual transitions, especially dissolves prove more challenging. As shown in Figure 10 through Figure 15, the peak in cross-difference is not as clearly defined for such transitions, and generally tends to be smoother and more spread out. This is the result of the gradual nature of the transitions, which means that for any frame during the transition the differences computed between pairs of frames to the left (right) of the current frame are not negligible, and may be close in magnitude to the differences computed across the current frame. Hence, the cross-difference even at the highest point of the peak may not reach the predetermined threshold, which leads to a missed transition.

It will be shown later, that the same transition becomes very apparent when color moments are analyzed with respect to the mathematical model of gradual transitions.

Figure 10 Example of a correctly detected dissolve

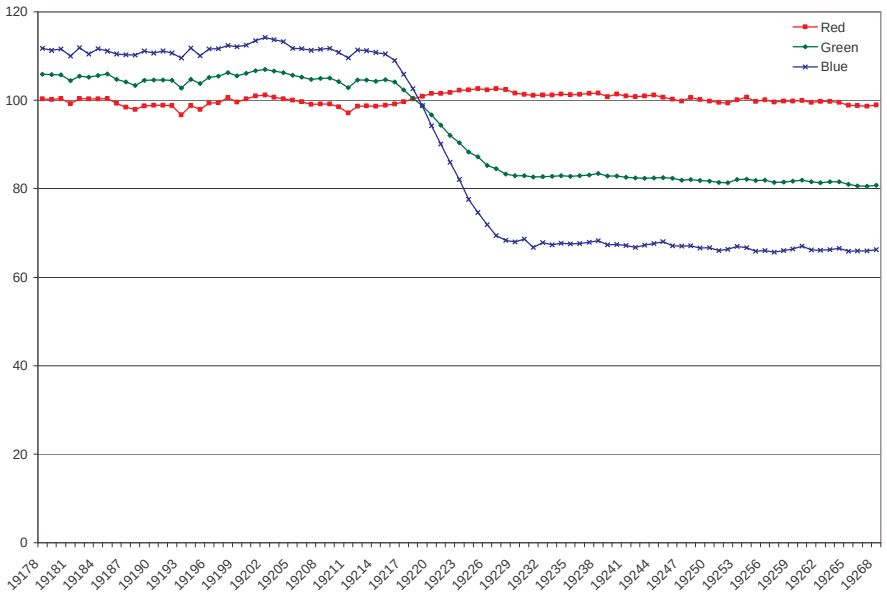

Figure 11 Mean curves for a correctly detected dissolve

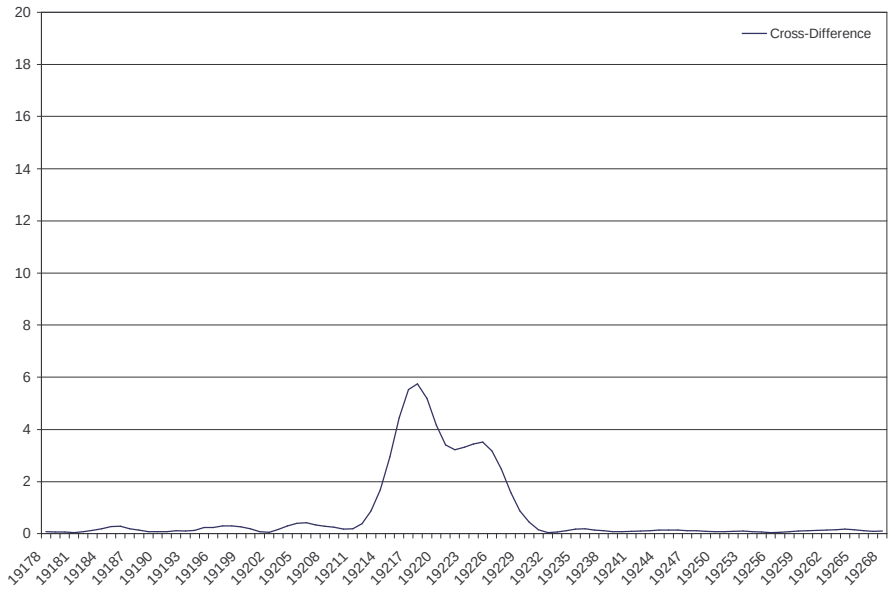

Figure 12 Cross-difference curve for a correctly detected dissolve

Figure 13 Example of a missed dissolve

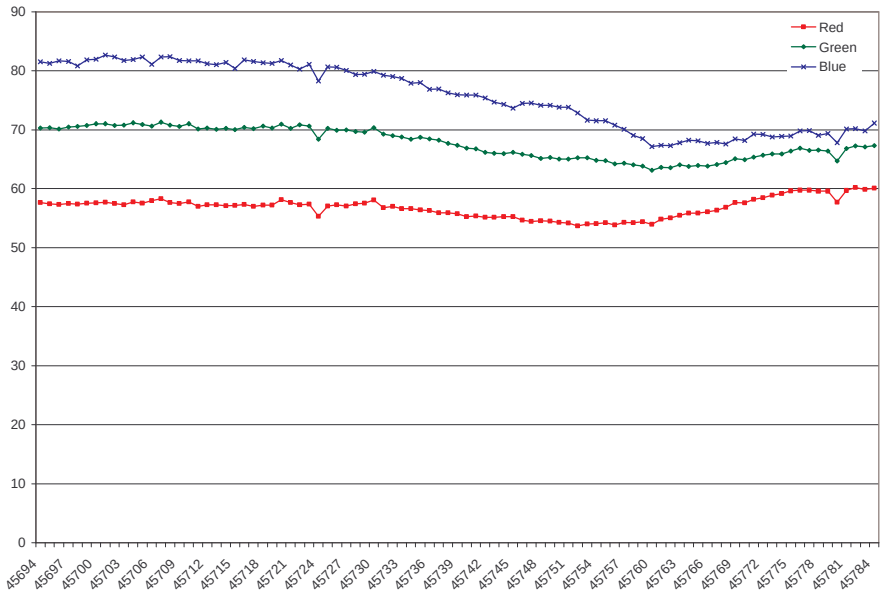

Figure 14 Mean curves for a sample missed dissolve

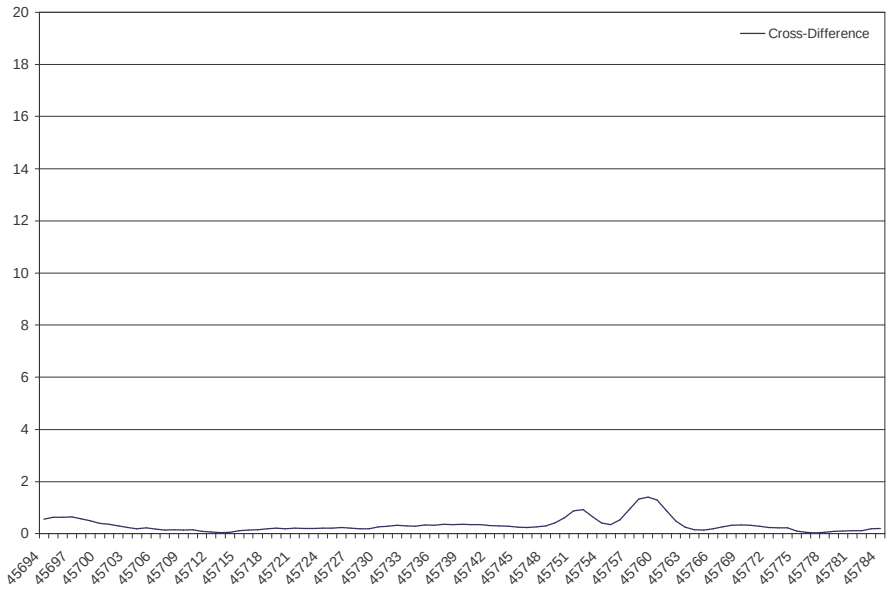

Figure 15 Small peak in cross-difference for a sample missed dissolve

Another problem the algorithm faces is manifested in shots containing significant motion, caused either by large moving objects, or by the motion of the camera, such as when the camera operator follows a walking person, as shown in Figure 16. During such sequences, color moment differences between consecutive frames may

be considerable, and the difference between frames a certain distance apart tends to be quite large. This produces peaks in the cross-difference curve similar to those resulting from gradual transitions (see Figure 18). If the peak exceeds the predetermined threshold, a false transition will be reported.

Figure 16 Example of motion sequence which triggers a false transition report

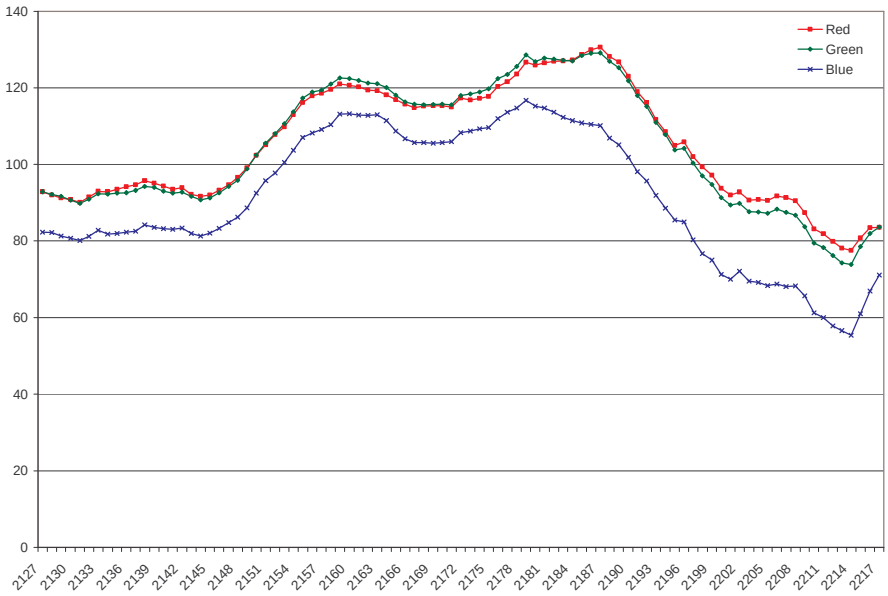

Figure 17 Mean curves for a sample motion sequence

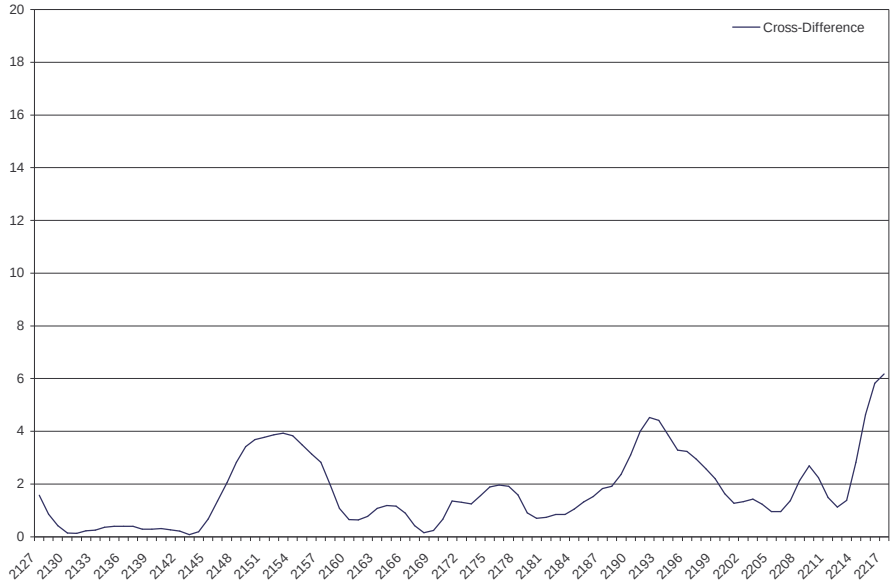

Figure 18 Cross-difference curve of a sample motion sequence

Finally, the algorithm requires that the cross-difference be maximal within the window in order to detect a transition. It follows then that if two or more transitions occur within the size of the window, at most one of them will be detected. Figure 21 presents an example, in which a fade directly following a cut is dominated by the latter in term of cross-difference, and therefore is not detected by the algorithm.

Figure 19 Example of a fade missed due to the proximity of a cut

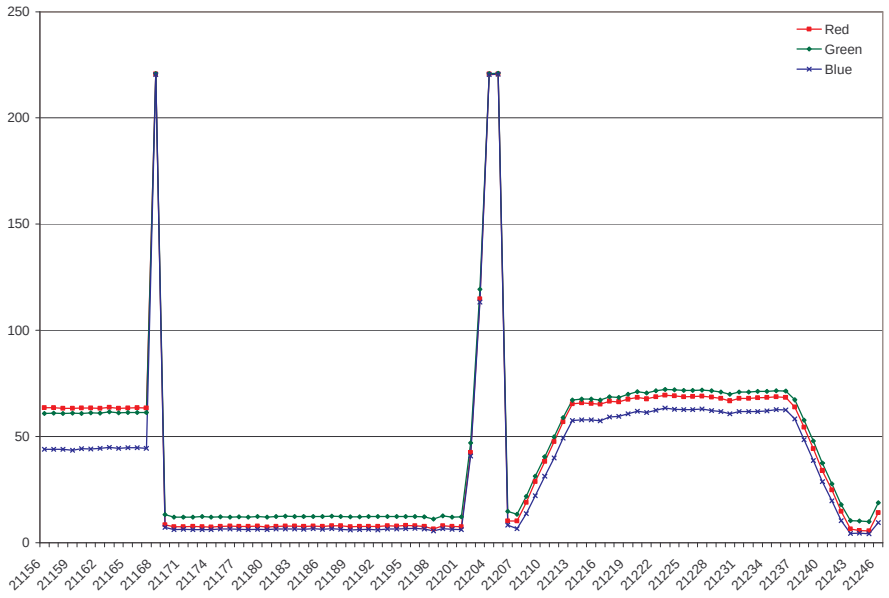

Figure 20 Mean curve for a sample missed fade

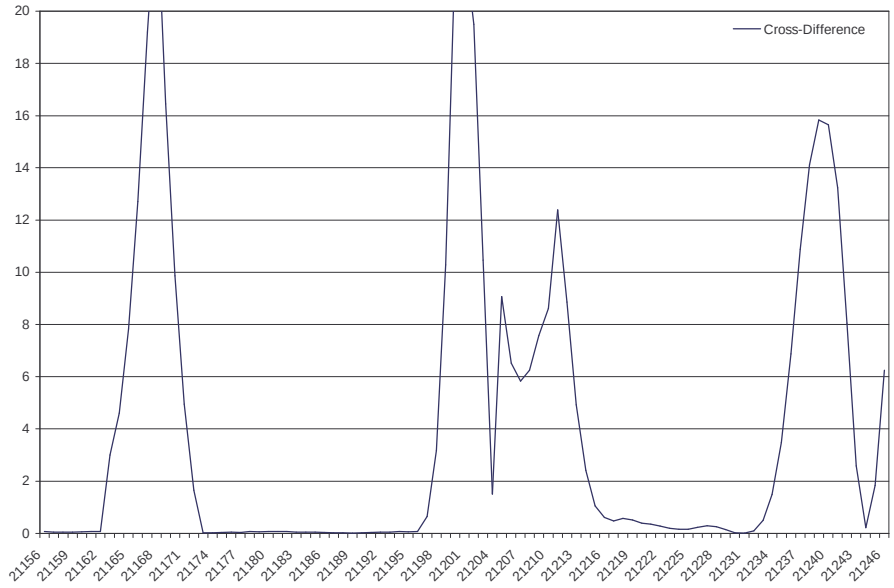

Figure 21 Cross-difference curve for a sample missed fade

Conclusions

Despite its simplicity the cross-difference algorithm performs reasonably well for all of the basic transition types, achieving around 80% recall and precision. On the other

hand, it suffers from two major shortcomings. First, its requirement that the cross-difference of the transition frame be maximal within the window of *2w+1* frames forces it to reject transitions which occur less than w frames apart. Although this is not a very frequent occurrence, it does deteriorate performance.

The second problem involves the method's inability to distinguish well between gradual transitions and effects of motion. No setting of the cross-difference threshold provides a good separation between those two cases. Consequently, for any setting of this single parameter the algorithm either reports a large number of false positives triggered by motion, or misses a large number of gradual transitions.

Finally, the cross-difference method does not provide any way to determine precise boundaries of gradual transitions. Rather, it determines the transition on a single frame with maximal value of cross-difference. Since this frame is not guaranteed to fall in the center of the gradual transition most of the transition frames may end up included in the shot directly preceding or succeeding the transition. Such inclusion may have an adverse effect on repeated footage detection (see Chapter 3).

As discussed in section 2.3, precise temporal segmentation is important for purposes of story tracking in video. Considering that our method of story tracking relies on detecting repeated video material, which is often surrounded and separated by dissolves, we should try to create better methods of temporal segmentation, especially focusing on improving dissolve detection. To this end, we decided to explore transition detection methods based on mathematical models of transitions in video (see section 2.1).

2.6 Transition model algorithm

This temporal segmentation method aims to detect different types of transitions separately. For each transition type a mathematical model representing the transition is chosen (see section 2.1) and some of its properties are determined. The detection

method for a given transition type consists in identifying characteristic patterns in the time-series of video frame features corresponding to the model of the transition type. Results of the individual transition type detection are then combined to form the overall temporal segmentation of the video stream.

Three main types of transitions are detected: cuts, fades and dissolves. All other transitions, which result from applying some computer generated effects to the video stream, are typically ignored. This is due to a wide variety of such effects used in modern video broadcasts, which make it virtually impossible for any single method to identify them consistently. The following sections describe individual transition type detection methods we developed, and discuss their performance on the experimental data set.

2.6.1 Cut detection

Overview

Cuts are the simplest type of shot transition and occur when the last frame of one shot is followed immediately by the first frame of the next shot. Thus, a cut does not consist of any frames, but rather occurs between frames. Assuming (as we have before) that frames belonging to the same shot tend to be similar to one another, and frames taken from different shots are generally dissimilar, we can infer that the frame feature (here color moments) of the last frame of the shot before a cut, and the frame feature of the first frame of the following shot will also be dissimilar. Therefore, in general, cuts manifest themselves as discontinuities in the feature representation of video frames, and can be detected by computing the difference in frame feature(s) (here: color moments) of every pair of consecutive frames, and declaring a cut when this value exceeds a certain threshold. This is well illustrated in Figure 22 through Figure 24.

Figure 22 Example of a simple cut with little motion

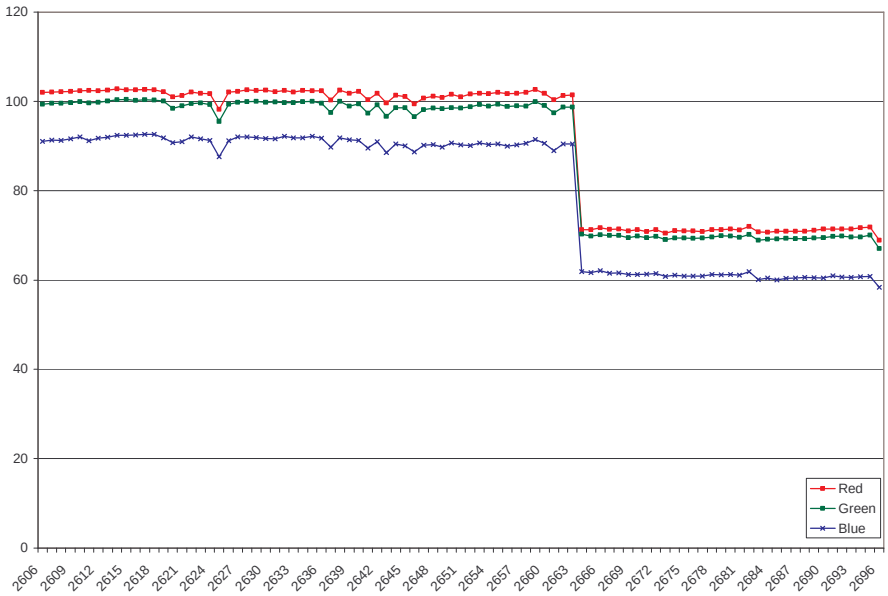

Figure 23 Mean curves for a sample cut with little motion

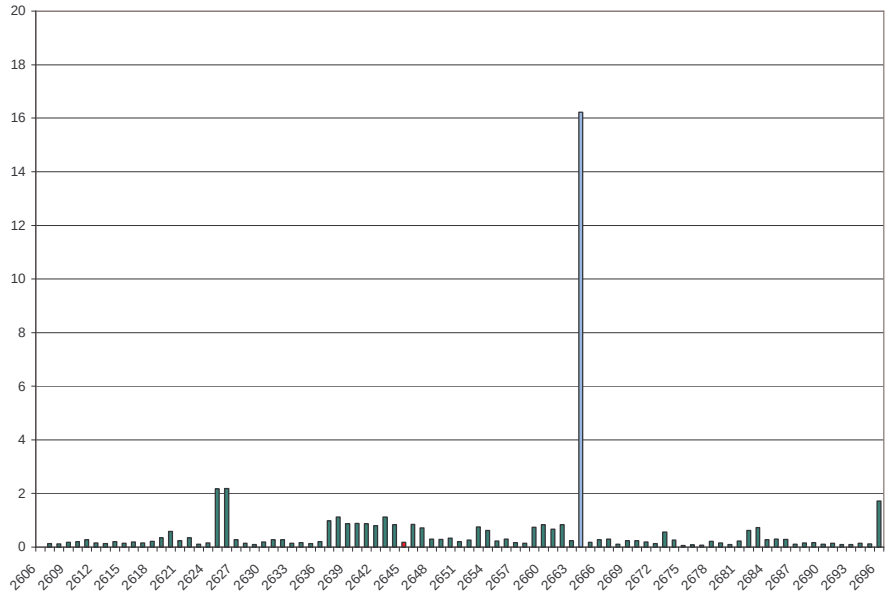

Figure 24 Frame moment difference for a simple cut with little motion

Clearly the moment difference at frame 2665 stands out from the surrounding frames. In this case, a simple threshold of 5.0 would isolate the cut frame from all other frames. Such a static threshold would work well across the entire video broadcast, if all shots in the broadcast were of similar nature. In practice, news videos contain a variety of shots ranging from often fast changing and intense-motion commercials to relatively static studio and anchor person shots. In some of them cuts are marked by only a very small change in color moments, such as in Figure 25 through Figure 27 below at frame 46034. In others, motion contained in the video footage causes color moments to change significantly even within a single shot. Consequently, one global threshold is not flexible enough to account for all different types of shots, and a method based on an adaptive threshold is needed.

Figure 25 Example of a cut with a small change in color moments

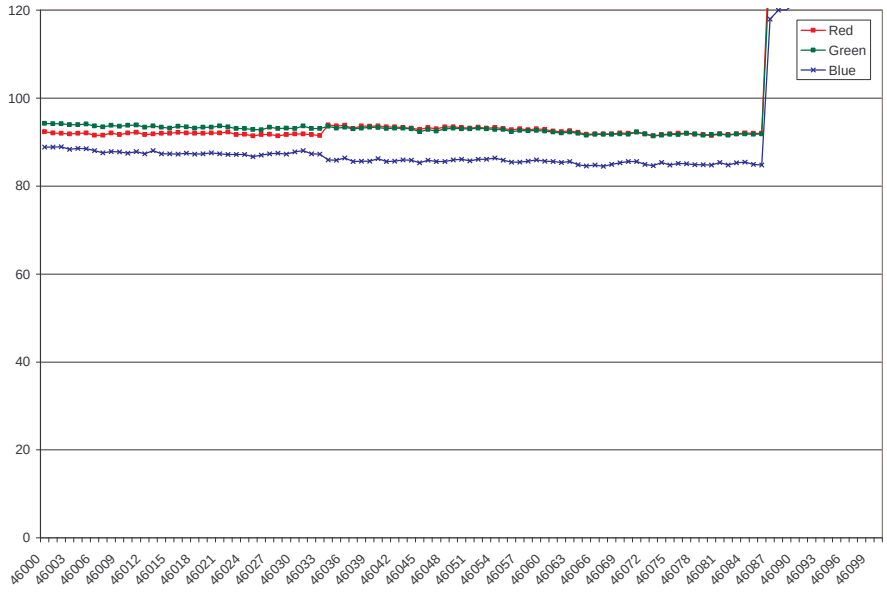

Figure 26 Mean curves for a sample cut with a small change in color moments

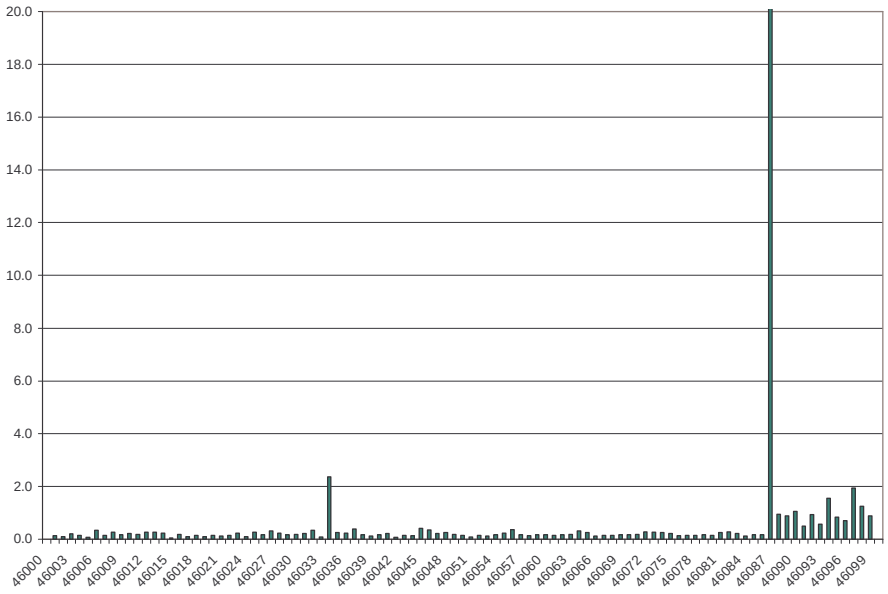

Figure 27 Frame moment difference for a sample cut with a small change in color moments

Truong et al [Tru00a] propose to detect spikes in frame feature difference using an adaptive threshold based on an average amount of change within a certain number of frames from the current frame. In their method, the authors utilize luminance histogram as the frame feature and calculate histogram differences of every pair of consecutive frames. They impose a window of *2w + 1* frames around every frame in video, and for every such window compute the mean histogram difference, excluding the center frame. A cut is reported if the following two conditions on histogram difference at the center frame hold:

1. The difference is maximal in the window.
2. The difference exceeds the dynamically adapted threshold, which is obtained by multiplying the mean histogram difference by a certain coefficient determined experimentally.

In addition, Truong observes that in the complete absence of motion the mean histogram difference is very close to zero, thus leading to the second condition being satisfied by center frames of even very small histogram difference, which do not represent cuts. In the domain of news, this is a fairly common occurrence during

sequences showing anchor persons. To guard against reporting such frames as cut frames, Truong *et al.* introduce a residual histogram difference by adding a small value to the histogram difference at every frame.

Since the authors report high performance of their method on a variety of video sources, including news, we decided to start the development of our cut detection algorithm with their method. Truong's method is independent of the frame feature selected, which allowed us to implement and test it using color moments instead of histograms for frame feature.

The advantage of using an adaptive threshold is apparent in the following graphs. Figure 28 shows the values of frame feature difference directly, while Figure 29 presents the ratio of the feature difference and the adaptive threshold. In the graphs, the difference values for cut frames (true positives) are shown as positive bars, while the values for all other frames (false positives) are depicted as negative bars. Introducing a threshold of some value into the first figure, we would report a cut whenever this threshold is exceeded. In the second figure, a cut is declared if the ratio at the given frame exceeds 1.0. Clearly, the adaptive threshold provides better separation of cut and non-cut frames. Choosing any constant threshold in the left-hand side graph would lead to either missing several cuts or reporting a large number of false positives. On the contrary, only very few false alarms have a cut ratio which exceeds 1.0. Thus reporting cuts at frames whose ratio exceeds 1.0 should yield high precision and recall.

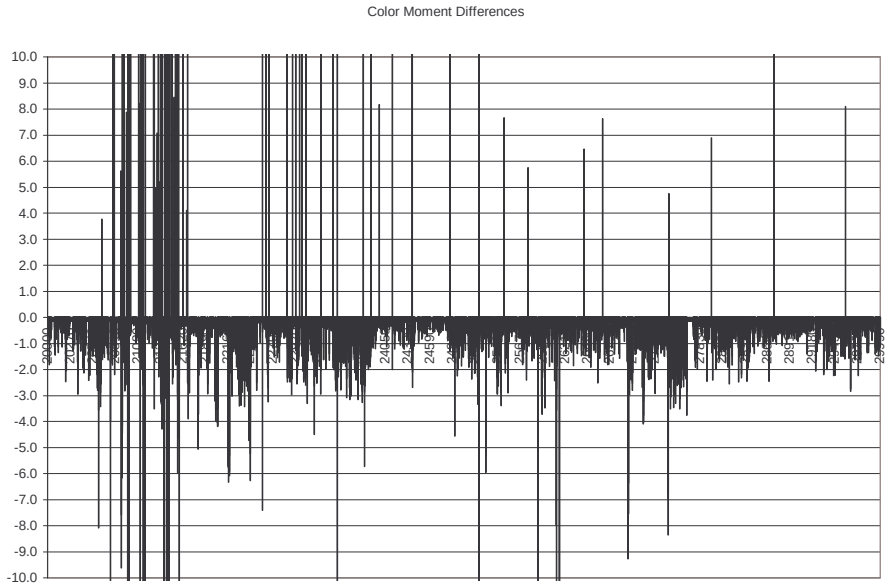

Figure 28 Frame moment difference values for true cuts and false positives

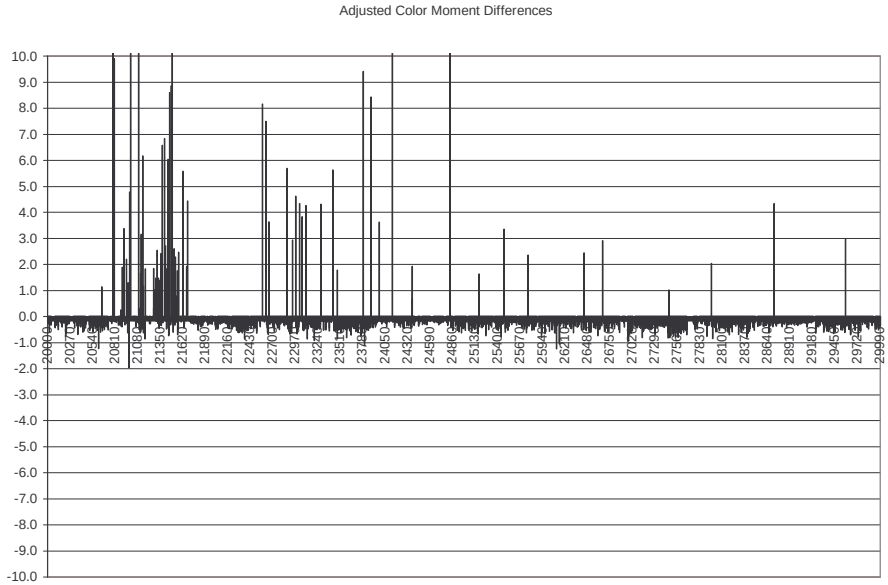

Figure 29 Frame moment difference values with adaptive threshold for true cuts and false positives

Truong *et al.* report very high performance (recall 98.5%, precision 98.5%) of their algorithm on news video streams [Tru00a]. We ran their algorithm on our test data, but were unable to reproduce their results. The performance we measured was recall

91.5% and precision 90%. This discrepancy is most likely caused by the difference in the news source used for experiments. Therefore, in order to provide performance comparison between Truong's algorithm and the method developed in our work, we evaluated both using the data set and methodology presented in section 2.4.

In working with Truong's algorithm we discovered two shortcomings. First, the algorithm requires that the moment difference of the current frame were a maximum within a window of *2w+1* frames. Consequently, the method systematically misses some cuts, if they occur less than w frames apart. Shots so short occur rarely, except in commercials. However, sometimes one shot cuts to black for just a couple of frames and then another cuts back from black. Out of two such cuts, at most one will be detected by the algorithm.

Second, Truong's method does not account for changes in feature difference across the window. We observed that even if color moment differences between consecutive frames in the window are significant, but do not vary much, a single frame with somewhat higher difference can be easily identified, and often represents a cut.

Therefore, we suggest a more statistically grounded approach. We propose that analyzing statistical properties of color moment differences may yield additional insight into distinguishing between motion induced changes and cuts. Specifically, we introduce standard deviation of color moment difference as the measure of consistency of color moment differences across the window. If the value of standard deviation is small, we can report cuts on frames with moment difference exceeding the mean by a smaller amount.

Algorithm

This section describes in detail the cut detection algorithm. The algorithm consists of three fundamental steps.

1. First, we compute color moment differences between every pair of consecutive frames in the video clip. We adjust the differences by adding a small (residual) difference to avoid false alarms in still frame sequences.

2. In the second step, we select a window of *2w + 1* frames around every frame in the video clip, and compute statistics of the moment differences within the window. Statistics are represented by the mean and standard deviation calculated over the frames in the window excluding the center frame.

3. Finally, in the last step we calculate the value of the adaptive threshold for this window and compute the ratio of the moment difference of the center frame to the threshold. If the ratio exceeds 1.0 we declare a cut. The adaptive threshold is calculated as a weighted sum of mean and standard deviation of moment differences. The weights applied are obtained experimentally.

Figure 30 presents the cut detection algorithm in pseudo code.

Cut Detection Algorithm

```
Function CalculateDifferences
    ForEach frame in VideoClip
        frameDiff[frame] = AbsMomentDiff(f1, f2) + residualDiff;
    EndFor
EndFunction

Function CalculateDifferenceStatistics
    ForEach frame in VideoClip
        mean[frame] = Mean(frameDiff[frame–w]:frameDiff[frame–1],
                           frameDiff[frame+1]:frameDiff[frame+w]);
        stdDev[frame] = StdDev(frameDiff[frame–w]:frameDiff[frame–1],
                           frameDiff[frame+1]:frameDiff[frame+w]);
    EndFor
EndFunction

Function DetectCuts
    CalculateDifferences();
    CalculateDifferenceStatistics();
    ForEach frame in VideoClip
        threshold = mw * mean[frame] + sw * stdDev[frame];
        cuts[frame] = frameDiff[frame] / threshold > 1.0;
    EndFor
EndFunction
```

Figure 30 Cut detection algorithm

The algorithm is controlled by four parameters, of which the first two must be determined *a priori*, and the remaining ones can be obtained experimentally:

1) window size (*w*)
2) residual moment difference (*d*)
3) moment difference mean coefficient (*mw*)
4) moment difference standard deviation coefficient (*sw*)

The main advantage of this algorithm is that no requirement is placed on the current frame to have the maximal moment difference within the window.

Experimental Results

In order to evaluate the cut detection algorithm, we examine its performance over the experimental data set described in section 2.4 using the same methodology. In addition, for comparison purposes, we evaluate performance of the Truong's algorithm on our experimental data.

Truong's algorithm is controlled by three parameters: window size (*w*), residual difference adjustment (*d*), and difference ratio threshold (*t*). The first two should be determined *a priori*, and we chose them to be the same as for our cut detection algorithm in order to make the results comparable. The values used were $w = 5$ and $d = 1.0$.

We measured recall and precision of the algorithm for the value of difference ratio threshold varying between 0.0 and 5.0. Results of this experiment are presented in Table 2 and Figure 31. It is apparent from the graph of recall and precision that the optimal combination of the two according to the utility function chosen is achieved for threshold 3.0, where both measures are close to 91%.

Threshold	Match	Mismatch	False Alarm	Missed	Recall	Precision	Utility
0.0	557	130	8335	29	95.1%	6.3%	50.66%
0.5	557	130	8335	29	95.1%	6.3%	50.66%
1.0	557	130	7448	29	95.1%	7.0%	51.00%
1.5	556	92	1199	30	94.9%	31.7%	63.28%
2.0	552	70	217	34	94.2%	71.8%	82.99%
2.5	545	46	89	41	93.0%	86.0%	89.48%
3.0	536	31	58	50	91.5%	90.2%	90.85%
3.5	516	23	38	70	88.1%	93.1%	90.60%
4.0	486	17	22	100	82.9%	95.7%	89.30%
4.5	461	14	12	125	78.7%	97.5%	88.07%
5.0	434	13	9	152	74.1%	98.0%	86.01%

Table 2 Truong's cut detection performance as a function of the difference ratio threshold

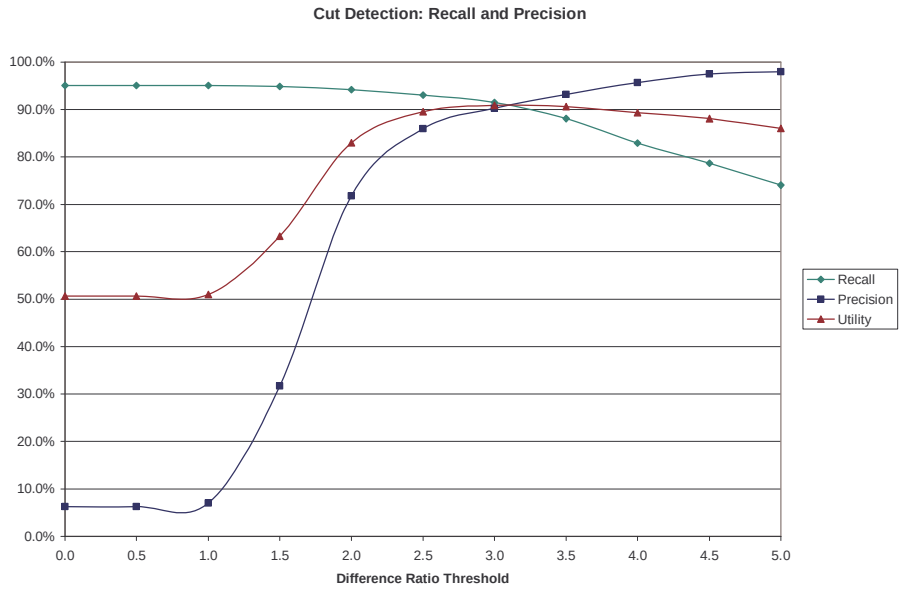

Figure 31 Recall and precision of Truong's cut detection algorithm as a function of the difference ratio threshold

It is important to note that even for threshold of 0.0, the algorithm does not yield 100% recall. This is due to the requirement that the moment difference at a cut frame be the maximum difference within the window of *2w + 1* frames. Therefore, out of every two (or more) cuts occurring less than w apart, at most one will be detected.

Since the value of recall at threshold 0.0 is 95.1%, we conclude that 5% of all cuts in our experimental video clip must occur within $w = 5$ frames of other cuts. This number is not negligible, and certainly leaves room for improvement.

The same experiment was performed using our cut detection method, and the results obtained will be discussed below.

%	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
0.5	50.39	49.84	49.39	49.26	48.97	47.76	46.26	2.91	0.00	0.00
1.0	51.05	51.99	53.86	59.98	76.12	**90.58**	84.29	0.00	0.00	0.00
1.5	62.62	71.51	81.91	90.12	**92.09**	87.80	58.87	0.00	0.00	0.00
2.0	81.18	87.19	90.98	**92.20**	88.90	78.98	51.45	0.00	0.00	0.00
2.5	88.74	90.99	**91.37**	89.56	83.97	71.42	0.00	0.00	0.00	0.00
3.0	90.94	**91.24**	89.88	85.80	78.29	62.97	0.00	0.00	0.00	0.00
3.5	**91.01**	89.73	86.87	81.90	73.37	58.45	0.00	0.00	0.00	0.00
4.0	89.63	88.01	83.53	78.11	68.52	55.12	0.00	0.00	0.00	0.00
4.5	88.47	85.51	80.48	74.57	63.65	53.07	0.00	0.00	0.00	0.00
5.0	86.42	82.39	78.35	71.84	60.32	51.88	0.00	0.00	0.00	0.00

Table 3 Cut detection performance as a function of mean and standard deviation coefficients

Table 3 shows the values of the utility function for different combinations of the weighting coefficients of mean and standard deviation. The values shown in bold face are maximal in their respective columns. It can be seen that the overall best performance achieved with $mw = 2.0$ and $sw = 1.5$ is better than the best performance attainable with $sw = 0.0$ (with $mw = 3.5$). This demonstrates that introducing standard deviation into the base of difference ratio improves the algorithm's performance. In addition, the comparison of the best value of utility of our algorithm (92.20%) with that of Truong's method (90.85%) shows that our algorithm performs better by a factor of 1.35%.

Figure 32 shows the plot of recall and precision for mean coefficient of 1.5 and standard deviation coefficient varying between 0.0 and 4.5. Overall with the optimal choice of all parameters, we were able to achieve recall and precision of 90% and above.

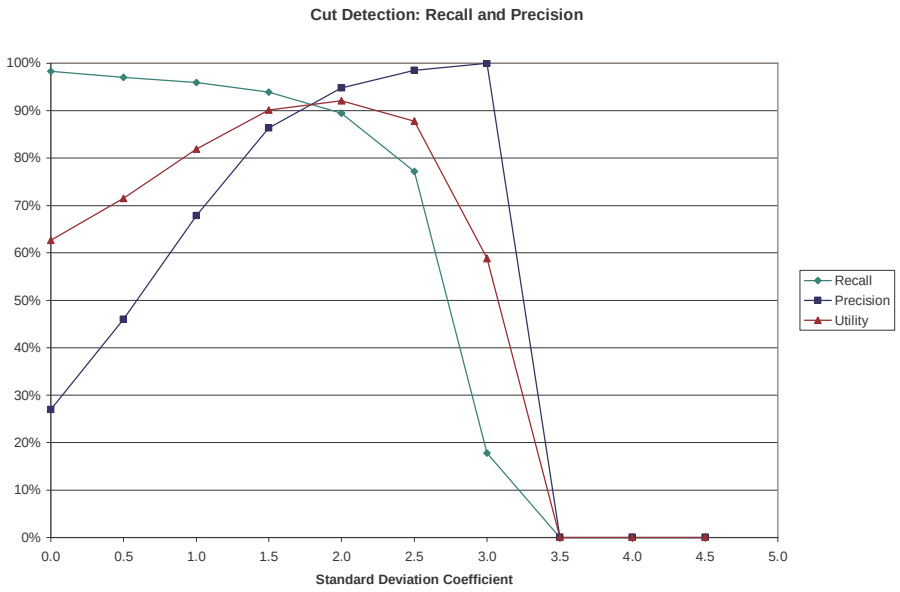

Figure 32 Cut detection performance as a function of the standard deviation coefficient with mean coefficient equal 1.5

Analysis of results

As described above, the cut detection algorithm performs very well across the whole video clip used in the experiments. Below, we will present examples of different contexts in which cuts appeared in the experimental video, and we will analyze the behavior of the algorithm.

A large portion of news video broadcasts consists of studio and anchor person shots, which are usually several seconds in length, and tend to contain little motion or otherwise change. Therefore, the color moments for these shots remain very stable and exhibit little variation. When a cut occurs, it manifests itself as a significant jump in the moment values (see Figure 34). Given very small differences between frames within the shots around it, the moment difference for the cut frame clearly dominates, and can be easily detected, as shown in Figure 35.

Figure 33 Example of a correctly detected cut with little motion and large change in moments

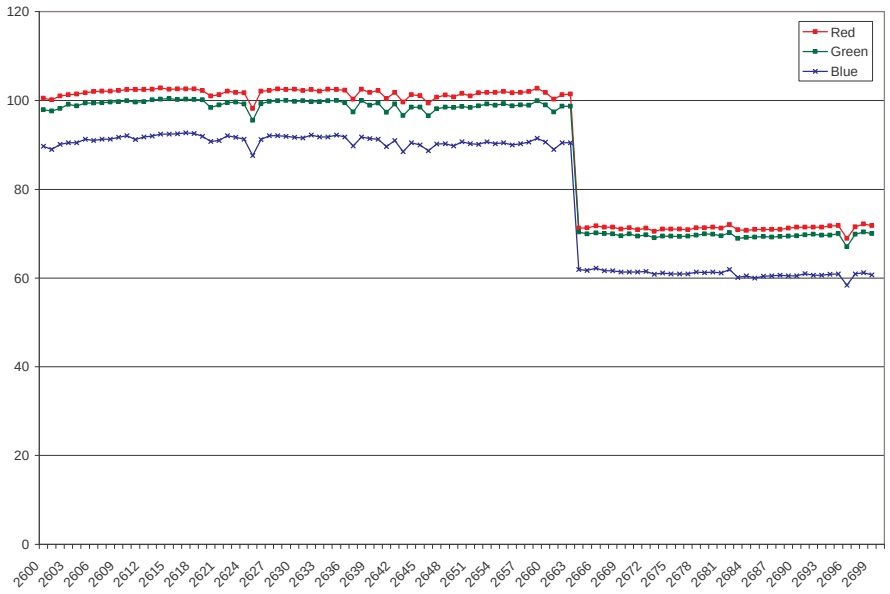

Figure 34 Mean curves for the sample cut

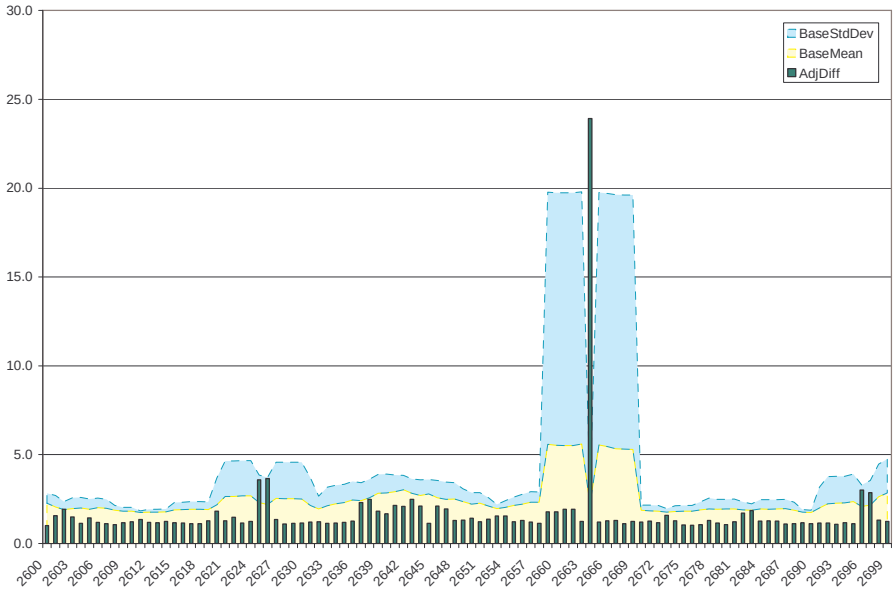

Figure 35 Adapted moment difference values for the sample cut

Such easy to spot changes in color moments can be accurately detected even by the trivial global threshold algorithm. The advantage of using adaptive threshold instead is demonstrated in Figure 40. Here the color moment change accompanying the cut at frame 46034 is hardly detectible even to the human eye (see Figure 37). Only after

zooming in on details (Figure 38) do we notice the change primarily in the red channel. But a quick glance at the adjusted moment difference plot (Figure 39), as well as the difference ratio graph (Figure 40), makes the transition very apparent.

Figure 36 Example of a correctly detected cut with a small change in moments

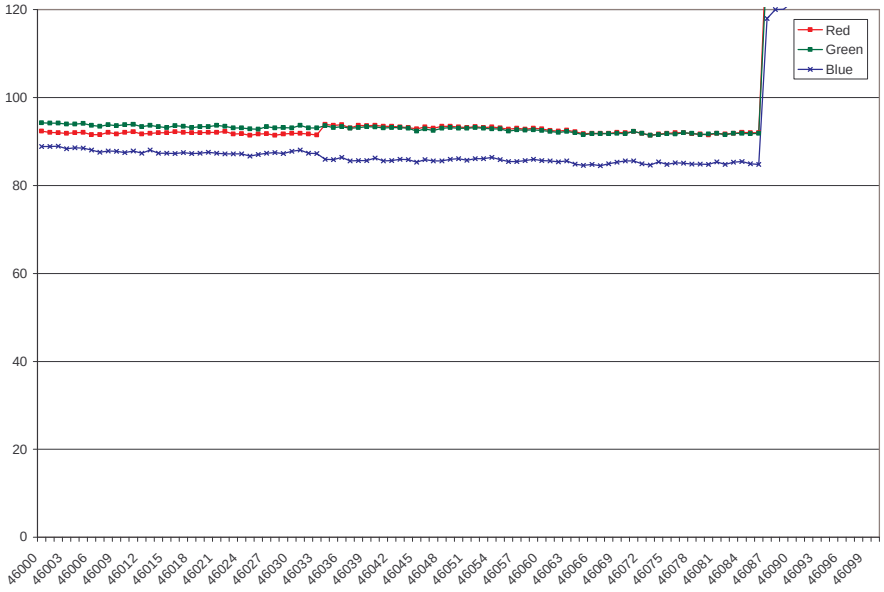

Figure 37 Mean curves for the sample cut

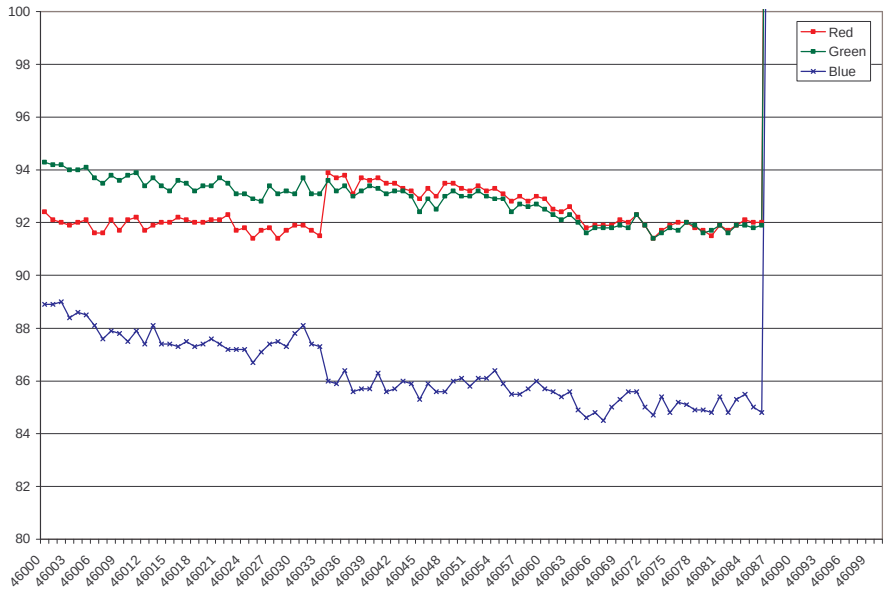

Figure 38 Enlarged mean curves for the sample cut

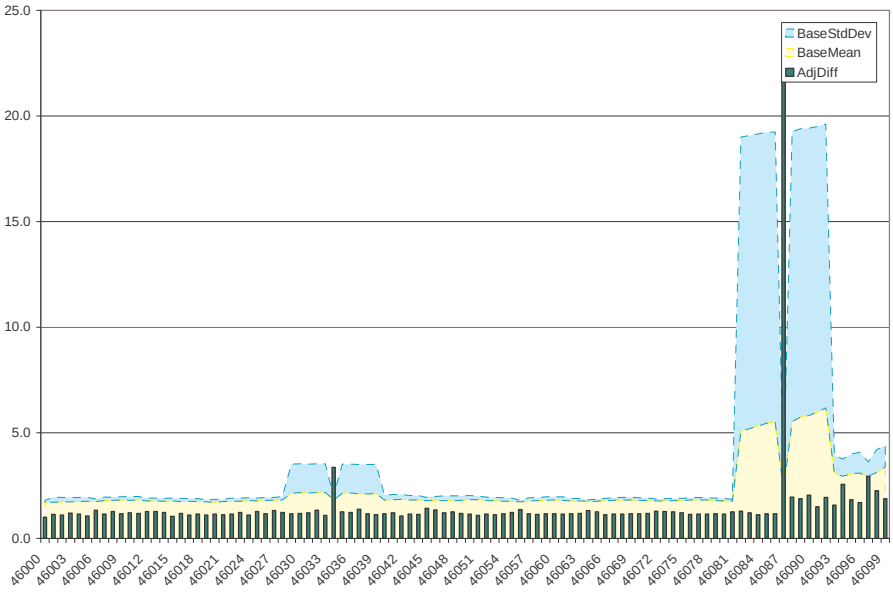

Figure 39 Adapted moment difference values for the sample cut

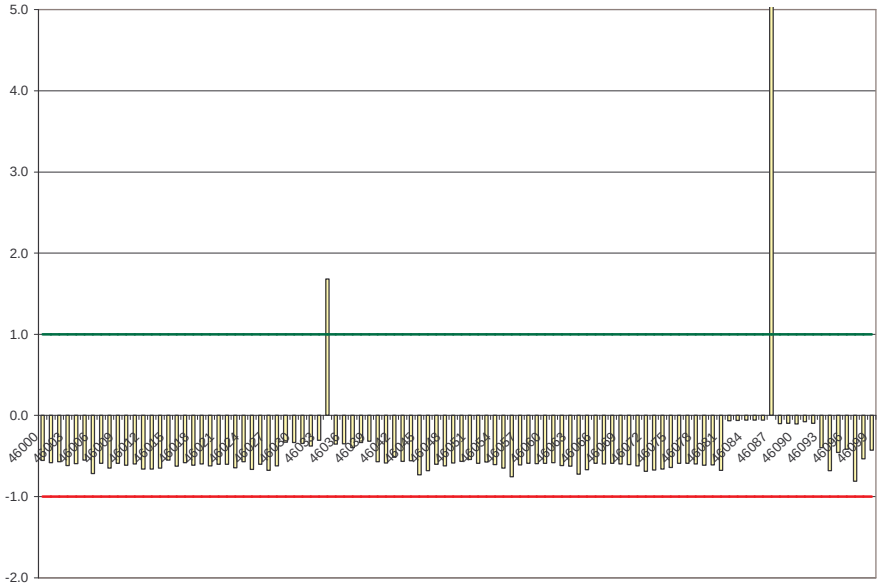

Figure 40 Moment difference ratio values for the sample cut

The cut detection task becomes more complicated in the presence of very short shots, especially if they contain rapid motion. The following example presents such a situation (see Figure 41 through Figure 44). The algorithm still performs reasonably well, detecting cuts at frames 21111, 21120, and 21139. The cut at frame 21106, however, goes unreported. The change in color moments due to this cut is visibly dominated by the nearby cut at frame 21111, which induces large mean and standard deviation.

Such missed cuts do not hinder us in our task of story tracking, as such short shots occur very rarely in actual news footage, and are present mostly in commercials and promotional clips.

Figure 41 Example of a sequence of very short shots

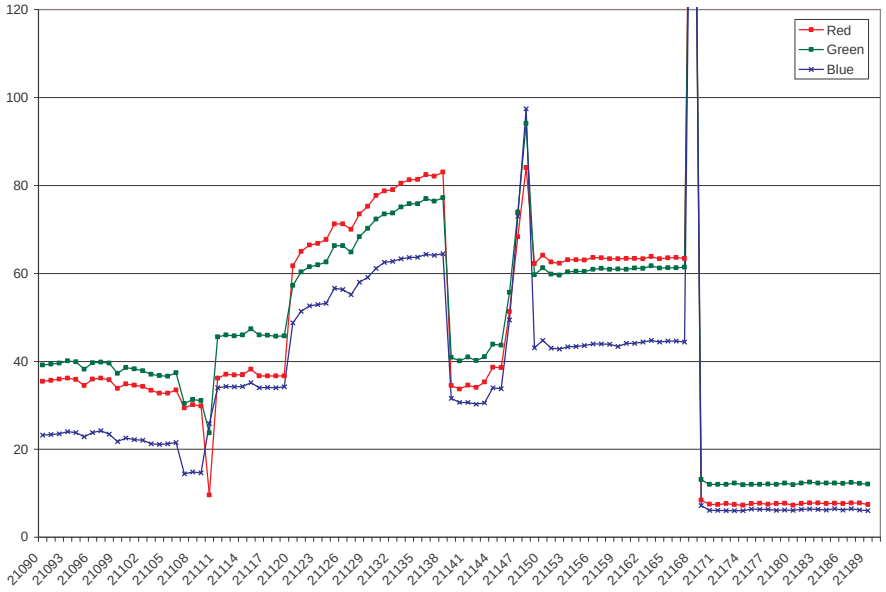

Figure 42 Mean curves for the sample sequence with short shots

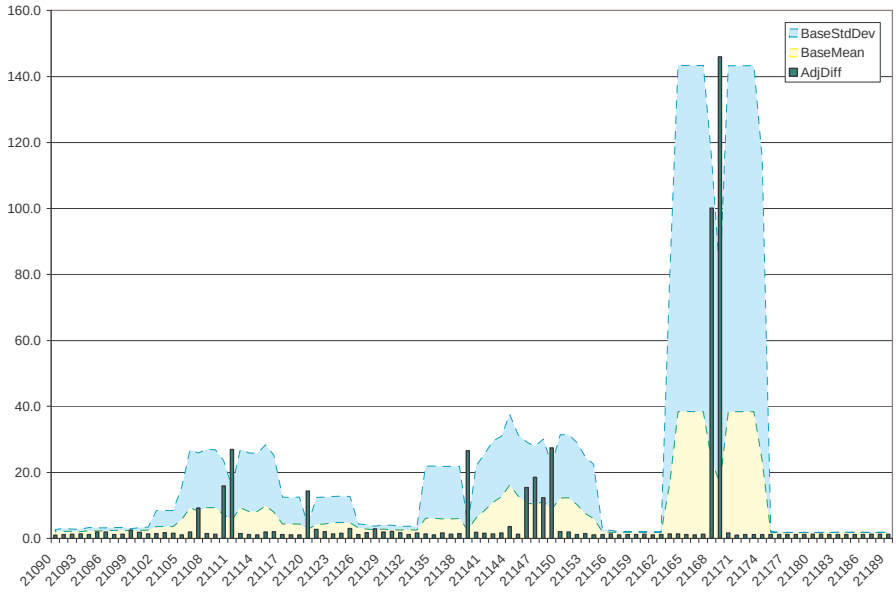

Figure 43 Adapted moment difference for the sample sequence with short shots

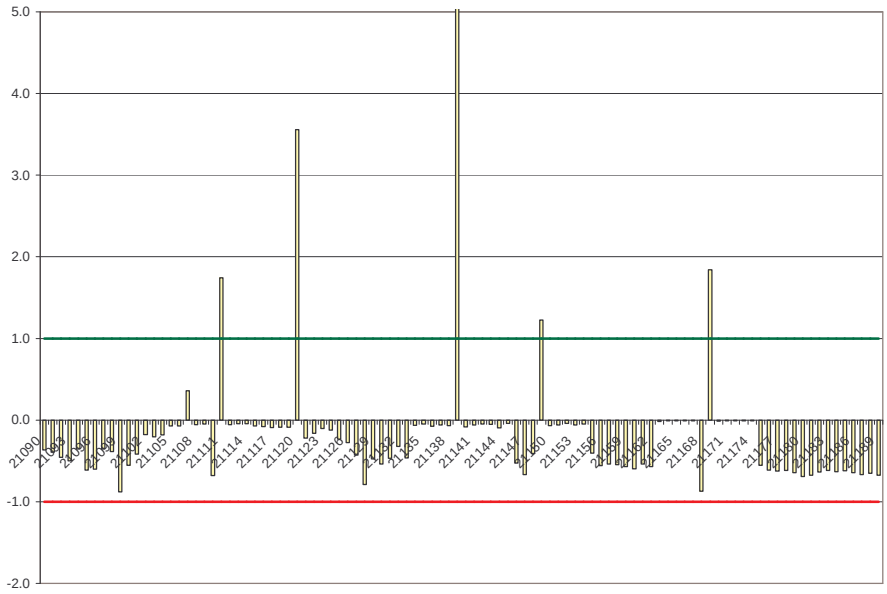

Figure 44 Moment difference ratio values for the sample sequence with short shots

The issue we were more interested in addressing with our approach was that of a shot ending with a cut to black followed by a shot starting very shortly afterwards with a cut from black, as presented in Figure 48. Here, both cuts are manifested by large moment differences, often of similar value (see Figure 46). Truong's algorithm systematically misses at least one of the cuts because of the requirement that the cut frame have the maximum difference value within the window. Our algorithm performs better in this case, allowing both cuts to be detected (see Figure 48). Whether both cuts will be detected depends on the values of mean and standard deviation of moment differences around them. Generally, if these differences are of similar magnitude, they are both correctly detected.

Figure 45 Example of a sequence with two cuts in close proximity

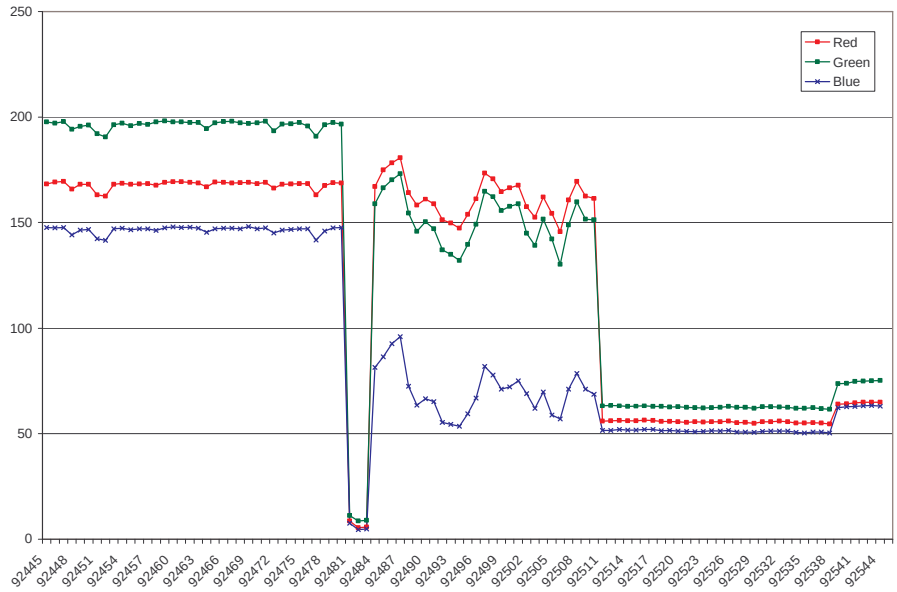

Figure 46 Mean curves for the sequence with two cuts in close proximity

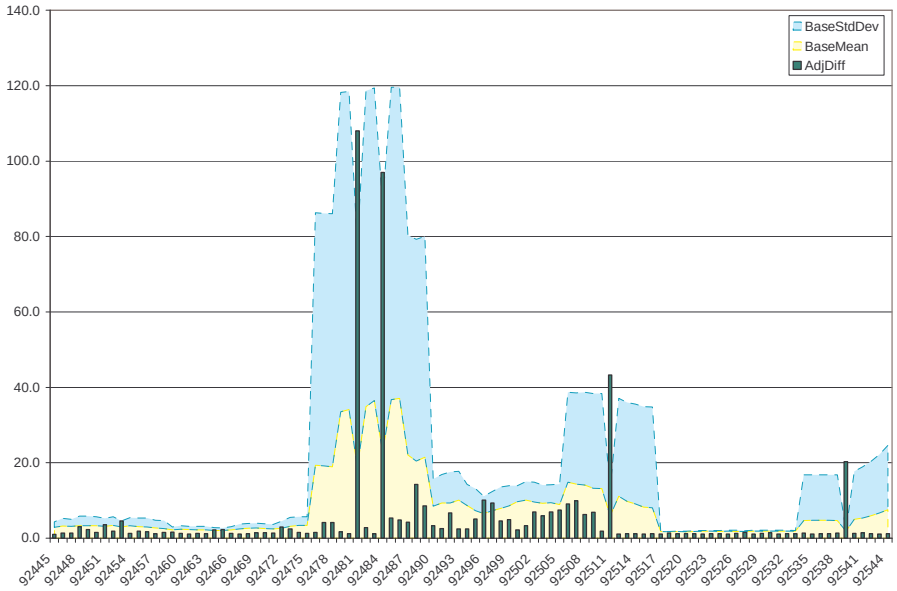

Figure 47 Adapted moment difference values for the sequence with two cuts in close proximity

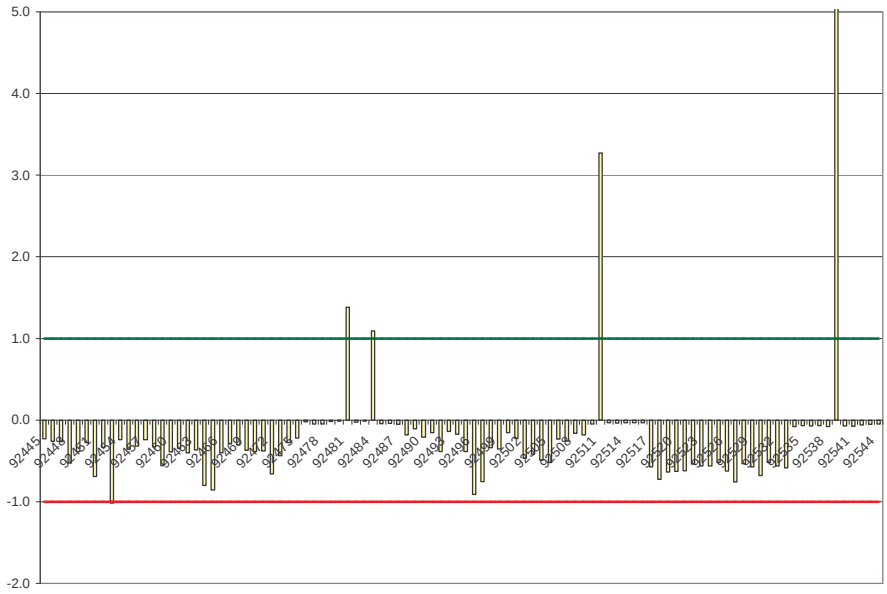

Figure 48 Moment difference ratios for the sequence with two cuts in close proximity

Our algorithm does not address the issue of cuts in the presence of rapid motion. A small percentage of cuts were missed due to the impact of motion, especially when the objects in motion appear close to the camera and obscure a large portion of the screen. When this happens the frame to frame moment differences are significant even between frames belonging to the same shot. Consequently, the frame difference for the cut frame does not considerably exceed differences for the surrounding frames, and is dominated by the weighted sum of mean and standard deviation. This is apparent in Figure 52 at frame 21455.

There appears to be nothing in the color moments domain alone that could alleviate this problem. Fortunately, this issue affects only a very small portion of video news footage.

Figure 49 Example of a cut in a sequence with rapid motion

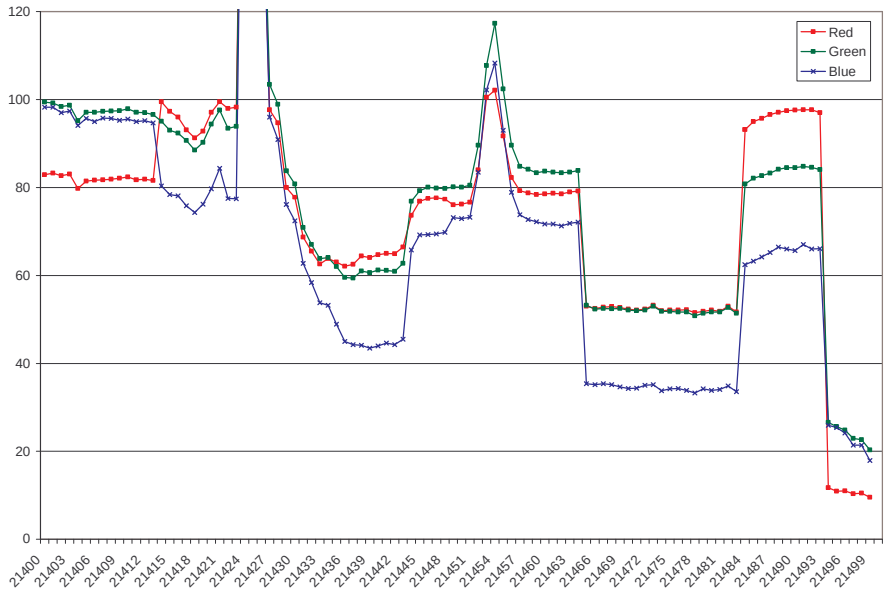

Figure 50 Mean curves for the sample sequence with rapid motion

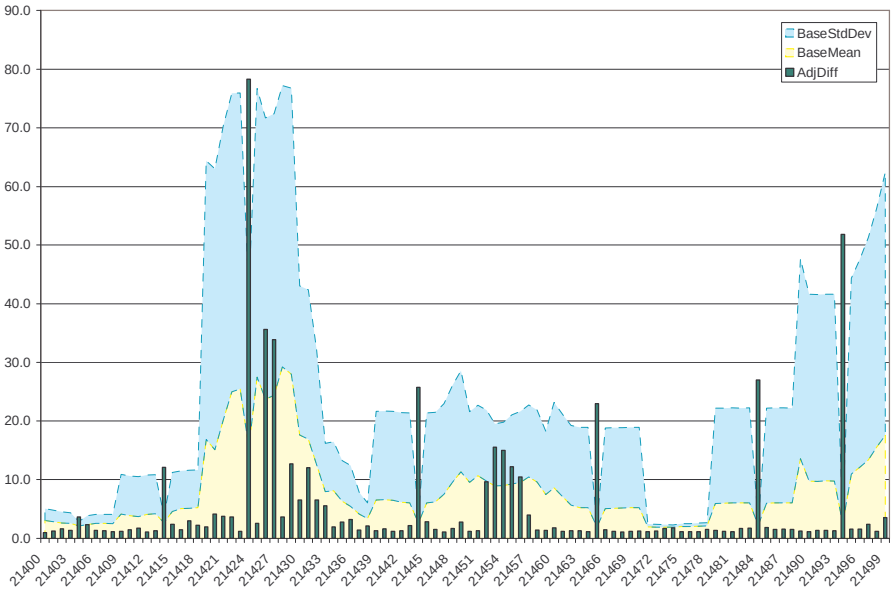

Figure 51 Adapted moment differences for the sample sequence with rapid motion

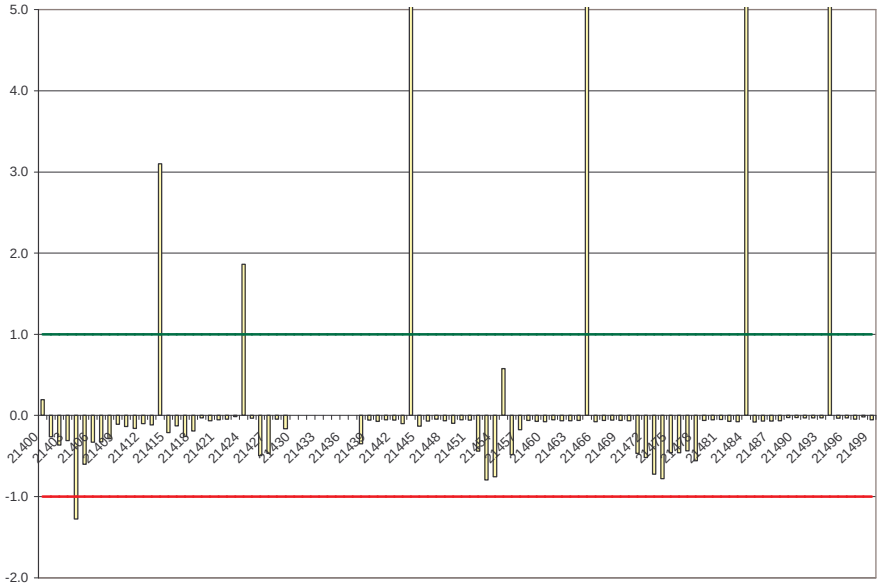

Figure 52 Moment difference ratios for the sample sequence with rapid motion

We observed a few occurrences of a curious artifact of video compression, which causes a single cut to be distributed over two frames (see Figure 53 through Figure 56). If this occurs, the difference values for both frames are high and may exceed the dynamic threshold. As a result, they may both be reported as cuts. This could be alleviated by requiring that the cut frame have a locally maximal value of moment difference, which would allow for only one of the two frames to be declared as cut. Currently, our transition matching method takes care of this issue. The manually annotated cuts use the frame before the cut as the start, and the frames after as the end, of the transition. Hence, if both frames are reported as cuts, they will overlap the same single manually annotated transition.

Figure 53 Example of a cut distorted by video compression

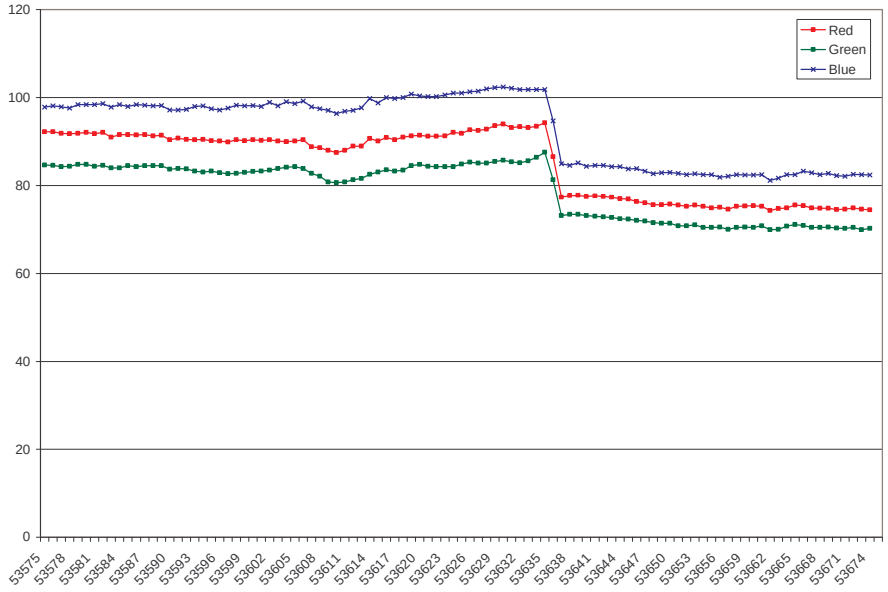

Figure 54 Mean curves for a sequence with a cut distorted by video compression

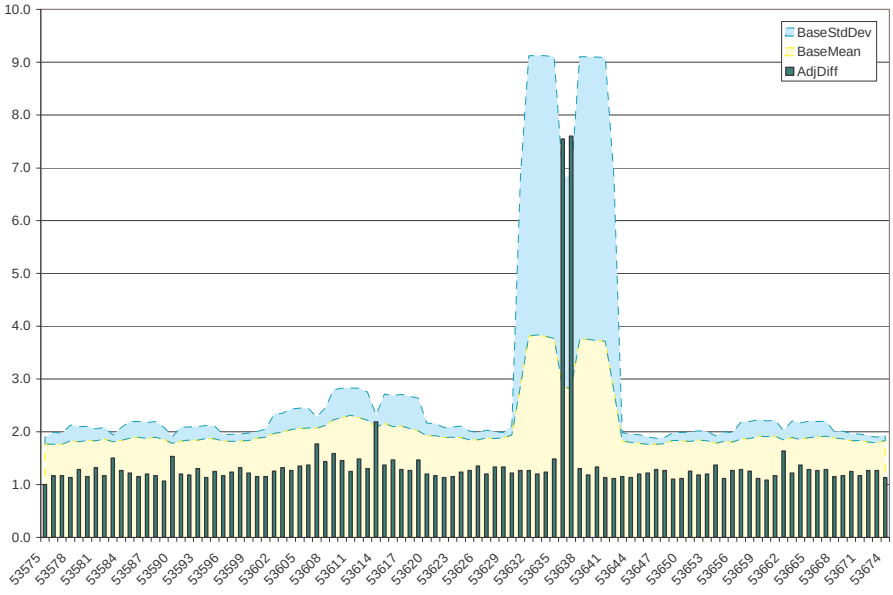

Figure 55 Adapted moment differences for a sequence with a cut distorted by video compression

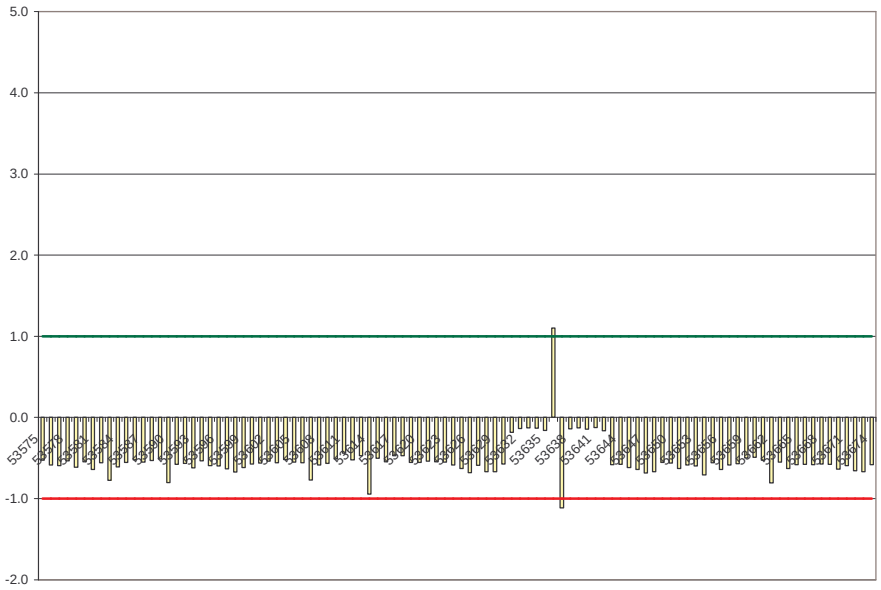

Figure 56 Moment difference ratios for a sequence with a cut distorted by video compression

Figure 57 Example of a camera flash interpreted as a cut

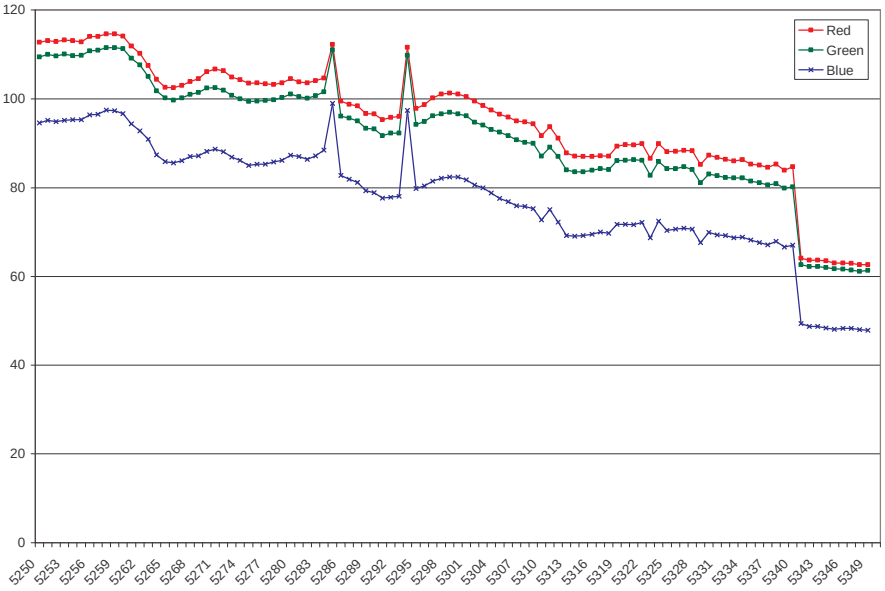

Figure 58 Mean curves for the sequence with a camera flash

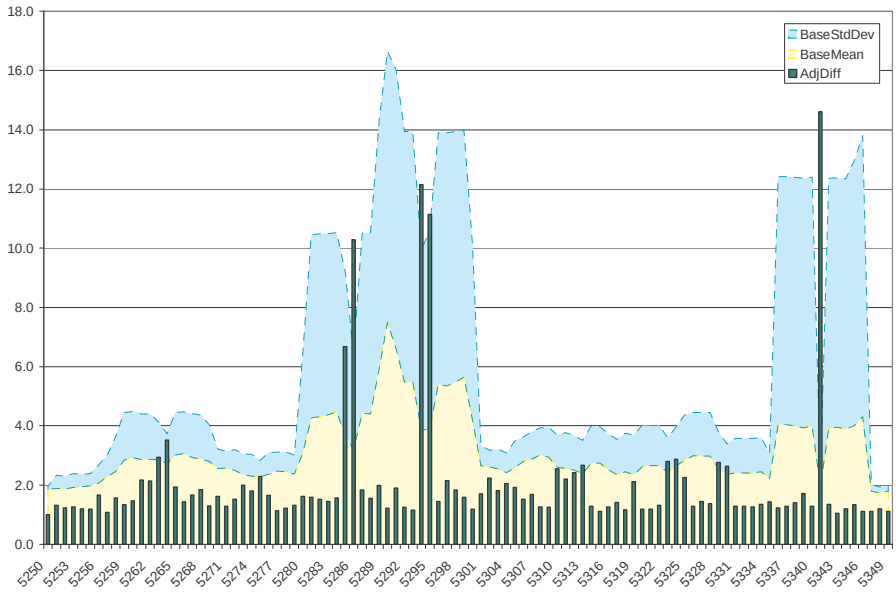

Figure 59 Adapted moment differences for the sample sequence with a camera flash

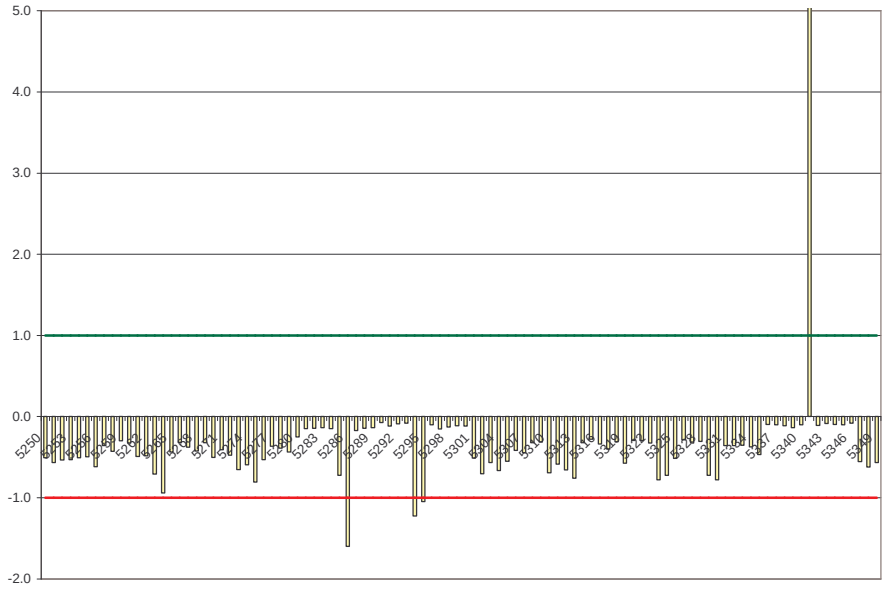

Figure 60 Moment difference ratios for the sample sequence with a camera flash

We have also identified camera flashes as a source of consistent false alarms raised by our algorithm. During a flash, the pixel intensity of the frame radically increases and then drops, as shown in Figure 58. This causes a significant change in color moments, and triggers cut detection. This problem could be addressed by comparing

frames immediately before and after such a spike in moments. If their color moment values are similar, then the cut could be rejected as a false positive. This approach would work well for relatively still video sequences. In the presence of motion, however, frames before and after the flash may be dissimilar and the flash will still be reported as a cut.

Conclusions

The experimental results and discussion presented above leads to conclude that the cut detection algorithm we adopted performs very well for video news broadcasts. Its performance is somewhat degraded in the presence of very short shots containing rapid motion, but this rarely occurs in the actual news footage, and hence remains without significant impact on our work.

2.6.2 Fade detection

Overview

As discussed in the introduction (section 2.1) fades are sequences of frames whose intensity gradually increases from black or decreases to black in time. Consequently, fade detection consists in recognizing sequences of monochrome frames (mostly black), and accurately marking the fade out and fade in slopes around them. The first task is relatively simple, when one observes that monochrome frames, regardless of their color, have practically zero variance, due to the fact that all pixels in the frame are of the same color. The second part presents more of a challenge. Given the start and end of the monochrome sequence, we must determine whether the sequence is preceded by a fade-out and whether it is succeeded by a fade-in. Once this is established we need to accurately determine the first frame of the fade-out and the last frame of the fade-in, i.e. at which point these transitions terminate and the actual shots begin.

It can be shown, that while the intensity of the pixels decreases gradually during fade out and increases gradually during fade in, so does the color standard deviation of the entire frames. In the ideal case, where the shot fading out or in contains no motion or any other change, the decrease or increase in standard deviation would be exactly linear. In practice, it tends to deviate somewhat from this theoretic model. Therefore, the existing fade detection methods seek to establish how well a given frame sequence matches the model.

Lienhart [Lie01a] proposes to use linear regression to assess how well the curve of standard deviation during fade in/out fits the linear model. While linear regression is a very good measure of linearity, it does not offer a satisfactory solution to the problem of detecting fade boundaries. Due to the nature of linear regression, if the number of frames that fit the linear model is large (i.e. the fade is relatively long), introduction of an additional point which does not fit the model well does not significantly perturb the slope and correlation. Therefore, a number of shot frames may be added to the curve, before the algorithm declares the end of a fade.

Truong *et al.* [Tru00a] used the second derivative of color variance to detect fade boundaries. They observed that the fade boundaries are accompanied by large spikes of second derivative curve. Indeed, an abrupt change in the slope of the variance curve at the beginning and end of a fade causes a large change in the first derivative, and consequently a large value of the second derivative. The same holds true for the curve of color standard deviation, which is shown in Figure 61 through Figure 64 (frames 21237 and 21253).

Truong argues that fades can be detected accurately by searching for such spikes around monochrome frame sequences. In their fade detection method they impose two additional conditions:

1. First derivative of color mean is relatively constant and does not change sign during the fade sequence.

2. The color variance of the frame immediately preceding a fade-out and the frame immediately succeeding a fade-in are above certain threshold.

Figure 61 Example of a fade-out and fade-in sequence

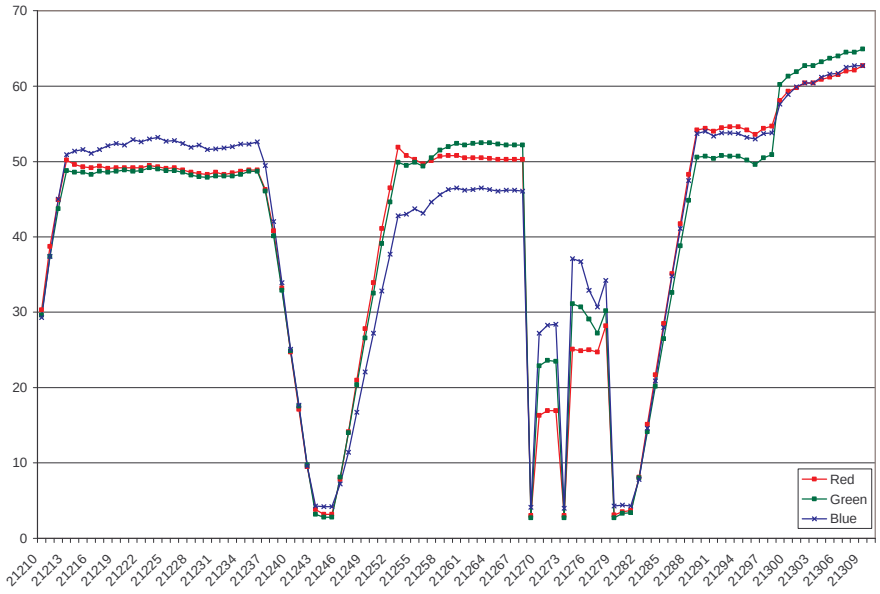

Figure 62 Standard deviation curves for the sample fade sequence

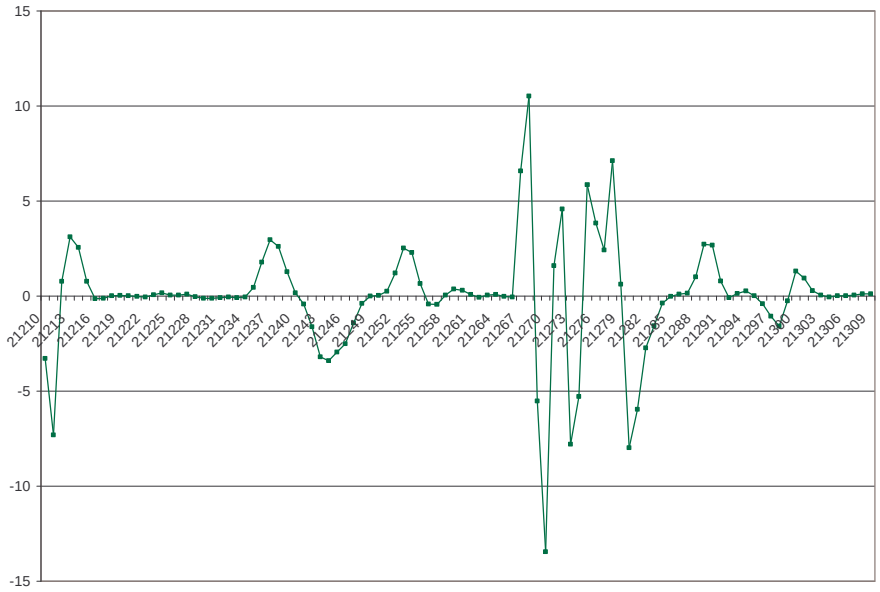

Figure 63 Smoothed second derivative of standard deviation for the sample fade sequence

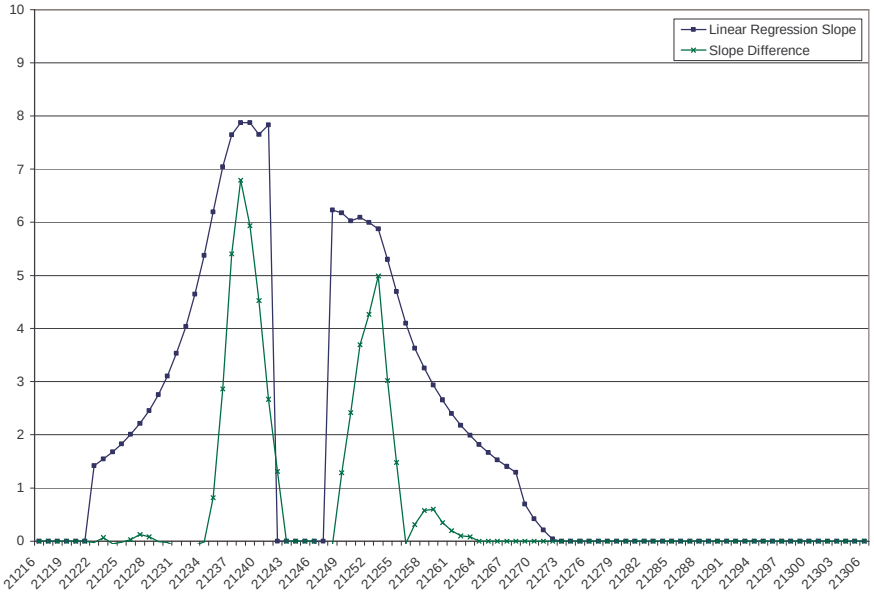

Figure 64 Slope difference vs. linear regression slope for the sample fade sequence

In this work, we chose a different approach, though also based on the maxima of second derivative of standard deviation. We approximate the second derivative of standard deviation as a difference in the smoothed first derivative. The smoothing operation is performed separately for frames before and after a given frame in the

potential fade sequence, so that we can reduce the effects of motion, while retaining the clarity of the extreme values of second derivative.

For every frame in the fade sequence we calculate the average first derivative over a certain number of frames prior to the current frame, and call this value the *inner slope*. We also calculate the average first derivative of a certain number of subsequent frames, and call it the *outer slope*.

In order to find the end point of the fade, we search for the frame at which the difference between the inner slope and the outer slope is maximal (i.e. the maximum of second derivative). This point corresponds very precisely to the end of fade in/out.

This solution is superior to using linear regression proposed in [Lie99] in two respects. First, it is computationally very simple, as it does not require computing powers and square roots. More importantly, it provides a precise cutoff point for the end of the transition. Figure 61 and Figure 65 show two typical combinations of fade-out and fade-in. In the former, both transitions are relatively short and their standard deviation curves are fairly steep. In the latter, the transitions are slow and smooth, yielding a more gradually decreasing and increasing curve. It is evident from the graphs in Figure 64 and Figure 68 that the slope obtained from linear regression drops off gradually, especially for slow transitions. In extreme cases, the difference in slope from one frame to another is virtually unnoticeable. This makes it difficult to detect the precise frame on which the transition terminates. Conversely, the slope difference obtained in our method shows a clear maximum at precisely the transition boundary. Hence, detecting these maxima guarantees high precision in fade boundary detection.

Figure 65 Example of a sequence with slow fade-out and fade-in

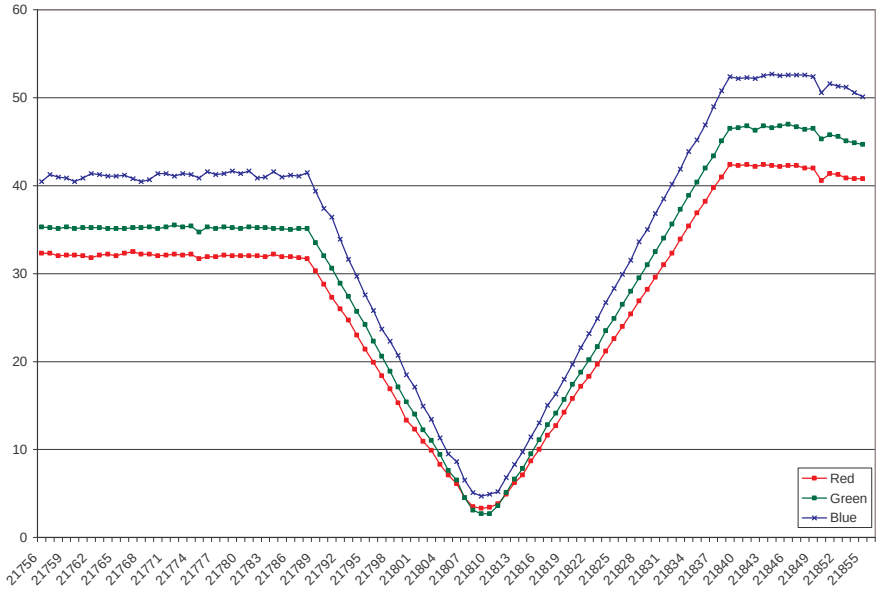

Figure 66 Standard deviation curves for the slow fade sequence

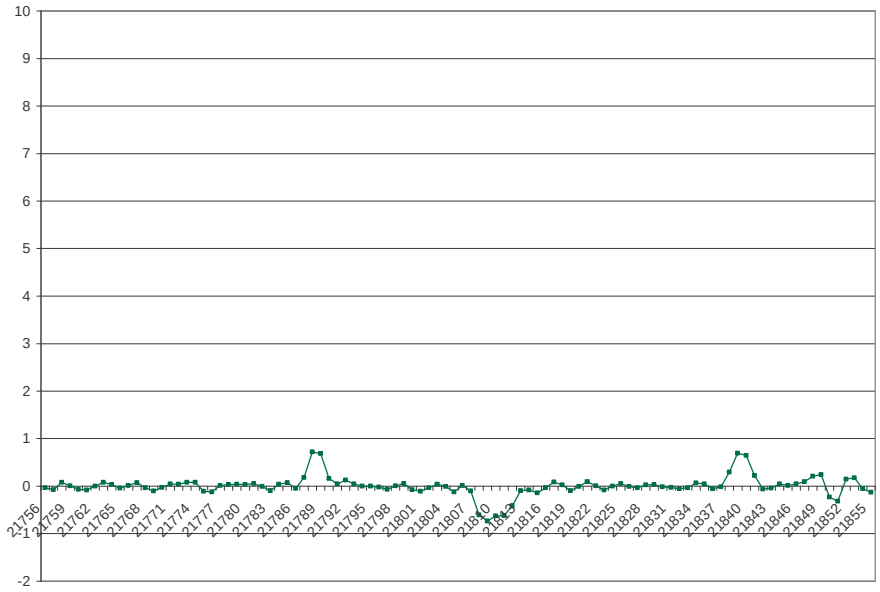

Figure 67 Smoothed second derivative for the slow fade sequence

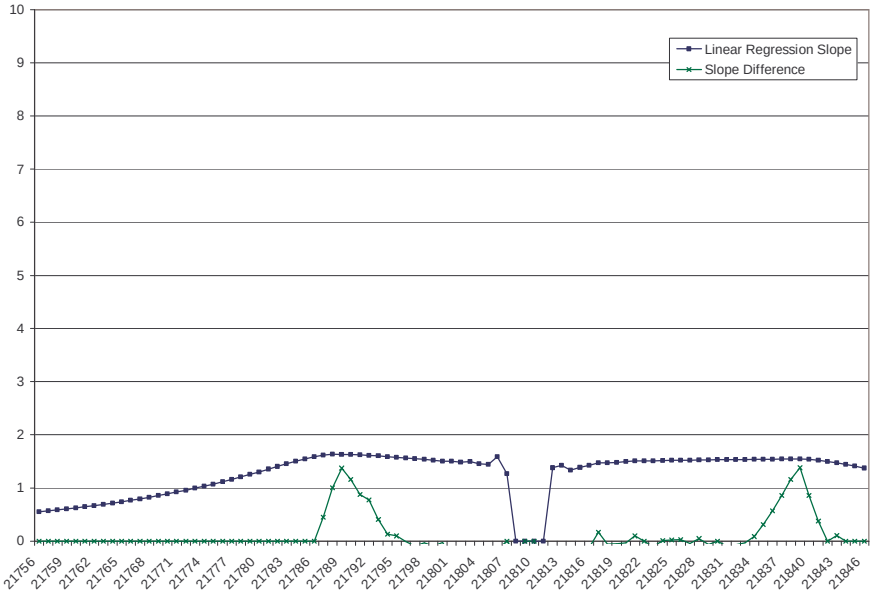

Figure 68 Slope difference vs. linear regression slope for the slow fade sequence

We believe that the additional conditions introduced by Truong may be eliminated, thus simplifying the fade detection, and reducing the potential for missing some fades which may not satisfy these criteria. We note that any sequence of monochrome frames must be preceded and succeeded by a transition. Clearly, three types of

transitions could be involved: cut, fade, or special effect. Since detecting a special effect transition is generally a good thing, we will not attempt to prevent our method from reporting them as fades. Given that, the fade detection can be reduced to distinguishing between cuts and fades. Our algorithm presented in the next section builds on this observation.

Algorithm

In this section, we describe in detail the fade detection method we developed. The algorithm consists of three major steps. First, we detect if the current frame marks the beginning of a sequence of monochrome frames. Then, we check whether the monochrome sequence is surrounded by linear slopes in standard deviation corresponding to fade-out and fade-in. And finally, we determine whether the slopes detected match certain predetermined criteria.

In the first step, we compare the three values of standard deviation for red, green, and blue against a monochrome frame threshold. We have experimentally established the optimal threshold value at 15.0. If all three standard deviation values fall below this threshold we consider the current frame monochrome, and mark it as the beginning of the monochrome sequence. We then proceed to test subsequent frames in the same manner until we reach a frame which is not monochrome. The last monochrome frame marks the end of the monochrome sequence.

Given the start and end of the monochrome sequence we analyze the frames before its start and after its end, to find a potential fade-out and fade-in. For this purpose, we calculate the inner and outer slope for every frame up to 60 frames away from the start (end), and as long as the inner slope exceeds the minimal fade slope threshold. We choose the frame for which the slope difference reaches maximum to be the start (end) of the fade-out (fade-in).

Finally, we determine if the potential fades satisfy the following criteria:

a) The ratio of the total standard deviation difference between the start and end of the fade to the maximal standard deviation difference between any consecutive frames during the potential fade exceeds certain threshold, which we call *slope dominance threshold*.

b) The slope of the standard deviation curve is between the minimum and maximum acceptable values, named *minimal* and *maximal fade slope*, respectively.

Figure 69 shows the algorithm in pseudo code.

Fade Detection Algorithm

```
Function IsFrameMonochrome(int frame)
    return RedStdDev(frame) < monoFrameThreshold And
        GreenStdDev(frame) < monoFrameThreshold And
        BlueStdDev(frame) < monoFrameThreshold
EndFunction

Function FindMonochromeSpanEnd(int frame)
    endFrame = frame;
    While (IsFrameMonochrome(endFrame))
        endFrame++;
    EndWhile
    return endFrame
EndFunction

Function CalcSlope(int startFrame, int endFrame)
    float slope = 0.0;
    For int frame = startFrame To endFrame
        slope += Abs(AvgStdDev(frame) – AvgStdDev(frame-1));
    EndFor
    slope = slope / (endFrame – startFrame + 1);
EndFunction

Function FindFadeOut(int monoStartFrame, int monoEndFrame)
    climbing = True;
    frame = monoStartFrame;
    While innerSlope > innerSlopeThresh Or climbing
        innerSlope = CalcSlope(frame, frame + innerSlopeSize)
        outerSlope = CalcSlope(frame – outerSlopeSize, frame)

        If innerSlope – outerSlope > maxSlopeDiff
            maxSlopeDiff = innerSlope – outerSlope
            maxSlopeFrame = frame
        EndIf

        If innerSlope > outerSlope
```

```
                climbing = False
            EndIf

            frame++;
        EndWhile

        fadeOutStart = monoStart;
        fadeOutEnd = monoStart

        If innerSlope between minSlopeThresh And maxSlopeThresh
            If innerSlope dominates maxFrameDiff
                fadeOutStart = maxSlopeFrame
                fadeOutEnd = monoStartFrame
            EndIf
        EndIf
EndFunction

Function FindFadeIn(int monoStartFrame, int monoEndFrame)
        Proceed analogically to FindFadeOut;
EndFunction

Function DetectFades
        ForEach frame in VideoClip
            If Not IsFrameMonochrome(frame) continue;

            monoStartFrame = frame;
            monoEndFrame = FindMonochromeSpanEnd(monoStartFrame)

            fadeOutStart, fadeOutEnd = FindFadeOut(monoStartFrame, monoEndFrame)
            fadeInStart, fadeInEnd = FindFadeIn(monoStartFrame, monoEndFrame)

            If Not FadeOutMeetsCriteria
                fadeOutStart = fadeOutEnd
            EndIf
            If Not FadeInMeetsCriteria
                fadeInStart = fadeInEnd
            EndIf

            If (fadeOutStart != fadeOutEnd)
                fades[fadeOutStart] = fadeStartTag
                fades[fadeOutEnd] = fadeEndTag
            EndIf
            If (fadeInStart != fadeInEnd)
                fades[fadeInStart] = fadeStartTag
                fades[fadeInEnd] = fadeEndTag
            EndIf

            frame = fadeInEnd;
        EndFor
EndFunction
```

Figure 69 Fade detection algorithm

Experimental Results

The fade detection algorithm is controlled by four parameters listed below.

1. ***Maximum detectable fade length***. This parameter determines how far from the start (end) of a monochrome sequence the algorithm searches for a fade boundary. The value of this parameter can be essentially arbitrary, and should be determined a priori. It should be large enough to accommodate the longest fade expected in the video sequence. We used 60 frames, or about 2 seconds, for the maximum fade length.

2. ***Minimal fade slope threshold***. The value of this parameter controls the minimum slope of the standard deviation curve during a fade. If the *inner slope* falls below this value, the algorithm stops its search for the maximum slope difference. Also, if the overall slope of the curve, between the start (end) of the monochrome sequence and the start (end) of the potential fade out (in) is below this threshold, the fade will not be reported. The optimal value of this parameter should be determined experimentally.

3. ***Maximal fade slope threshold***. This parameter serves as a sanity check on the steepness of the standard deviation curve during a potential fade. If the overall slope of the potential fade sequence exceeds this threshold, the sequence is not reported as a fade. The value of the threshold should be determined a priori, and should be relatively large. We selected 50.0 as the maximal fade slope threshold in the following experiments.

4. ***Slope dominance threshold***. This threshold protects against detecting cuts to and from monochrome frames as fade-ins or fade-outs. The value of this parameter determines the minimum ratio of the overall difference in standard deviation between the start and end of a potential fade sequence to the maximal standard deviation difference between any two consecutive frames in that sequence. If the ratio does not reach this threshold, then the sequence is

not reported as a fade. The optimal value of this parameter should be determined experimentally.

In order to evaluate the fade detection performance, we performed a set of experiments using the data and methodology described in section 2.4. We varied the values of *minimal fade slope threshold* and *slope dominance threshold*, to determine their optimal values and maximal achievable performance. The results are presented in Figure 70, Figure 71 and Table 4.

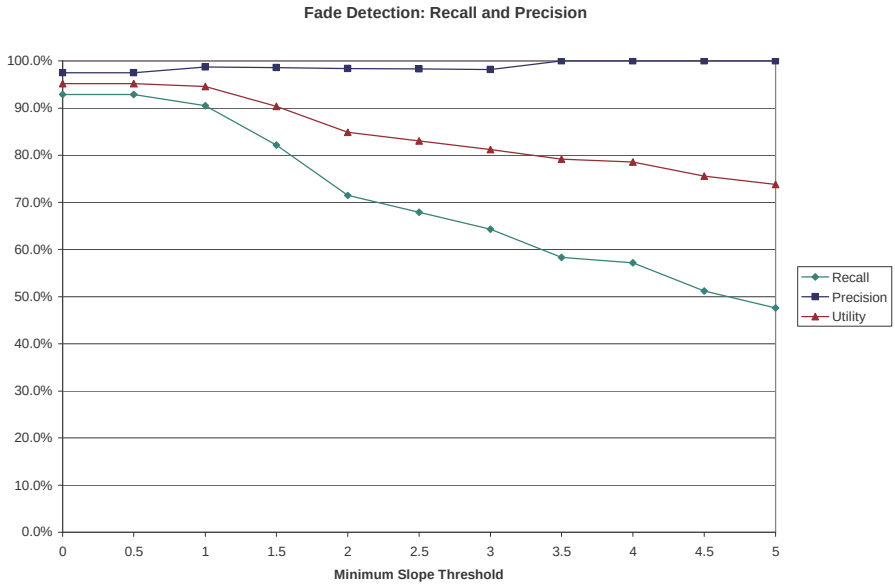

Figure 70 Fade detection performance as a function of the minimum slope threshold

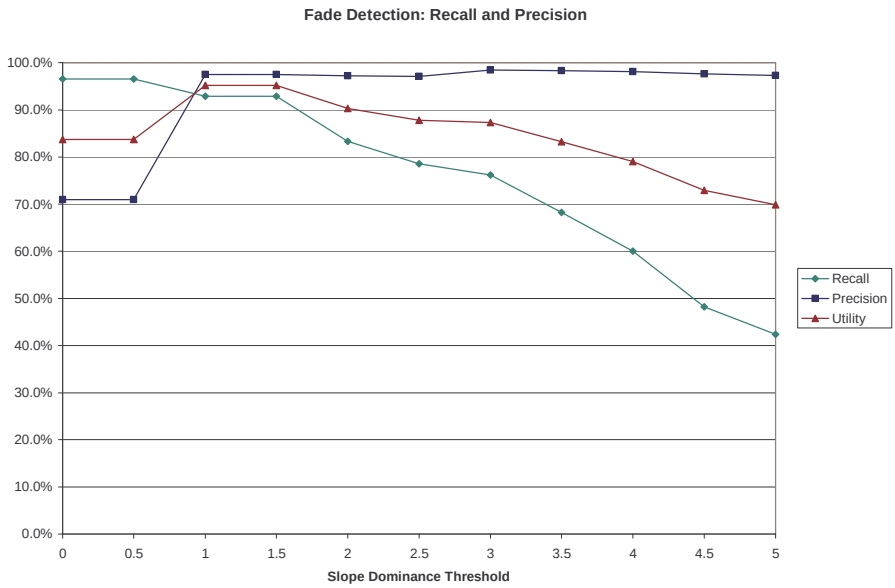

Figure 71 Fade detection performance as a function of the slope dominance threshold

Minimal Slope	Recall	Precision	Utility
0.0	92.9%	97.5%	95.18%
0.5	92.9%	97.5%	95.18%
1.0	90.5%	98.7%	94.59%
1.5	82.1%	98.6%	90.36%
2.0	71.4%	98.4%	84.89%
2.5	67.9%	98.3%	83.07%
3.0	64.3%	98.2%	81.23%
3.5	58.3%	100.0%	79.17%
4.0	57.1%	100.0%	78.57%
4.5	51.2%	100.0%	75.60%
5.0	47.6%	100.0%	73.81%

Table 4 Fade detection performance as a function of the minimal slope threshold

The experiments show high values of recall and precision reaching 92.9% and 97.5% respectively. Figure 71 shows the significance of the slope dominance threshold. If we do not require that the total difference be at least 1.0 times greater than the maximum consecutive frame difference, then some cuts are detected as fades, and

overall precision deteriorates. Otherwise, precision remains practically flat with the increase of both parameters, but recall decreases.

Figure 70 demonstrates that increasing the value of minimum slope threshold past 1.0 causes a visible decrease in recall. This indicates that many fades are relatively flat with a slope of 1.0 or less. In fact, the threshold value of 0.0 yields the highest overall value of utility, the same as for threshold of 0.5. We consider 0.5 to be an optimal value, as it may guard against reporting certain false positives.

Our results are comparable with those reported by Truong [Tru00a] for news sequences (recall 92.5% and precision 96.1%). In addition, further analysis will show that the fades our method actually missed are quite questionable. Some of them do not start or end with a strictly monochrome frame. Others do not fade gradually to a monochrome frame, but rather abruptly drop to it, more in the manner of a cut. If we eliminate such pseudo-fades, our method achieves 100% recall on the experimental data set.

The pseudo-fades which drop suddenly to a monochrome frame could be considered cuts. Our fade detection method could be adapted to report such transitions, and in the overall temporal video segmentation, cuts could be inserted if they have not been detected by the cut detection method.

The following section presents a detailed discussion of the fade detection experiments results.

Analysis

As noted earlier, our method consistently misses two types of pseudo-fades. The first one consists in a sequence of frames that starts to gradually fade out, but at a certain frame drops directly to monochrome, as illustrated in Figure 72. This poses a problem, because the maximum difference in the standard deviation slope (Figure 74) is found at this drop-off frame and the fade looks like a cut, i.e. it consists of a single large difference between consecutive frames. In this example, although the fade

stretches from frame 39019 to 39024, the sudden drop in standard deviation causes the maximum of the second derivative to fall on frame 39023. Therefore, this frame is recognized as the start of fade-out and the slope does not dominate the consecutive frame difference.

Figure 72 Example of a fade-out sequence ending with an abrupt cut to black

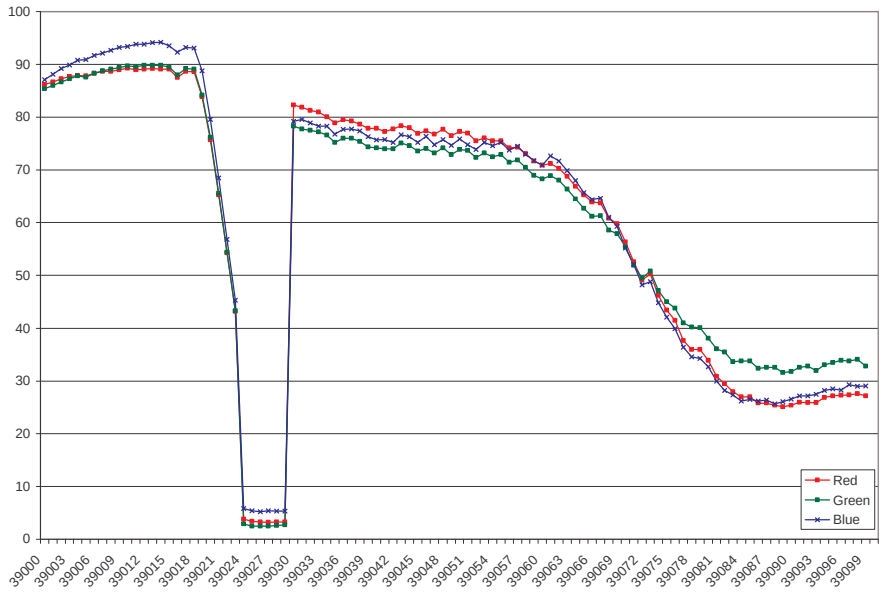

Figure 73 Standard deviation curves for the fade-out ending with an abrupt cut to black

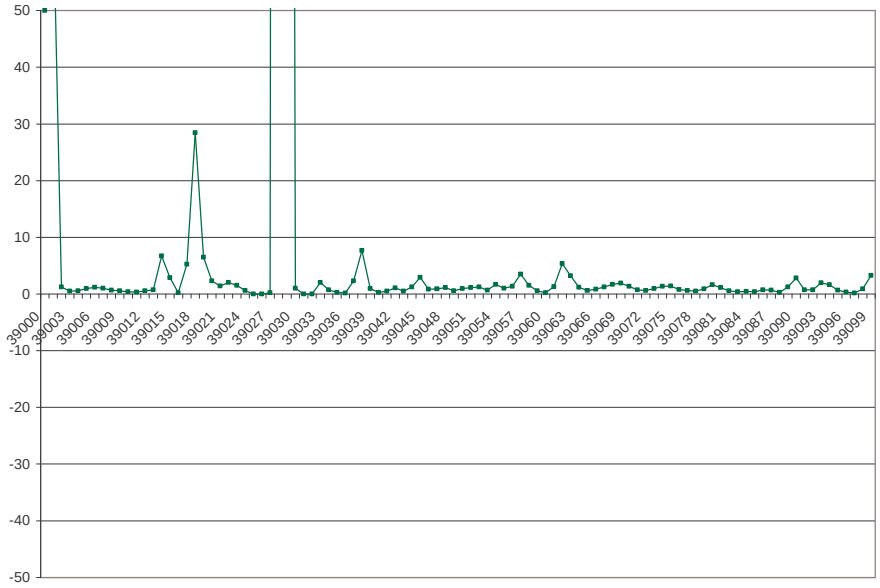

Figure 74 Slope difference of standard deviation for the fade-out ending with an abrupt cut to black

The other type of pseudo-fade not detected by our method is a fade-like sequence of frames which does not start or end with a truly monochrome frame, as in Figure 75. In this sequence, the value of standard deviation of the blue component at frame 20880 equals 16.0 and slightly exceeds the chosen monochrome threshold of 15.0 (Figure 76). Consequently, our algorithm does not find any monochrome frames, and does not attempt to detect potential fades surrounding it.

Figure 75 Example of a pseudo fade-in sequence which does not start with a monochrome frame

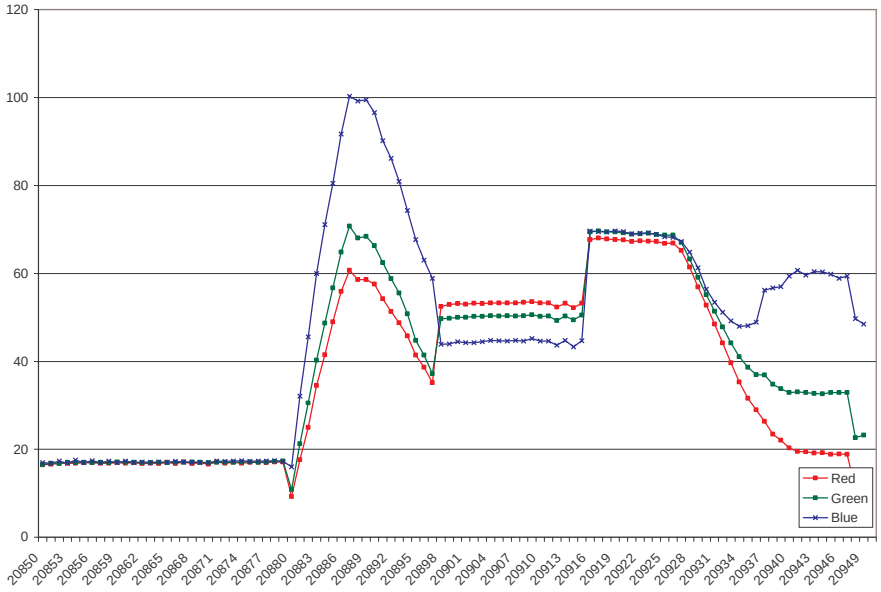

Figure 76 Standard deviation curves for the pseudo fade-in sequence

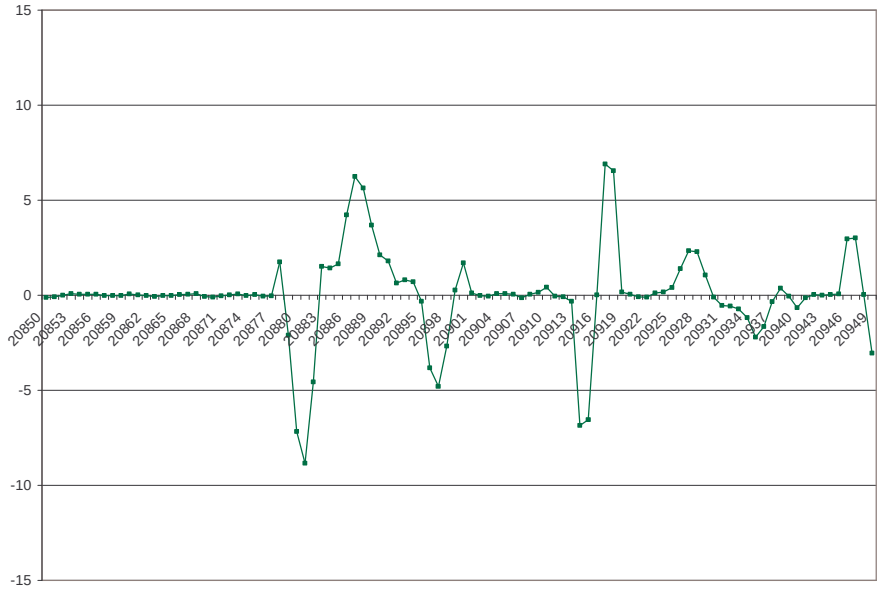

Figure 77 Slope difference of the standard deviation for the pseudo fade-in sequence

Finally, our method reports fade-ins and fade-outs at beginnings and ends of some computer generated transitions (see Figure 78). Such pseudo-fades do not count as false positives in our evaluation methodology, because they overlap a manually annotated transition, in accordance with definition in section 2.4. In fact, this

behavior should be considered beneficial, as it helps distinguish at least some shots separated by such complex transitions.

Figure 78 Example of a special effect sequence detected as a fade

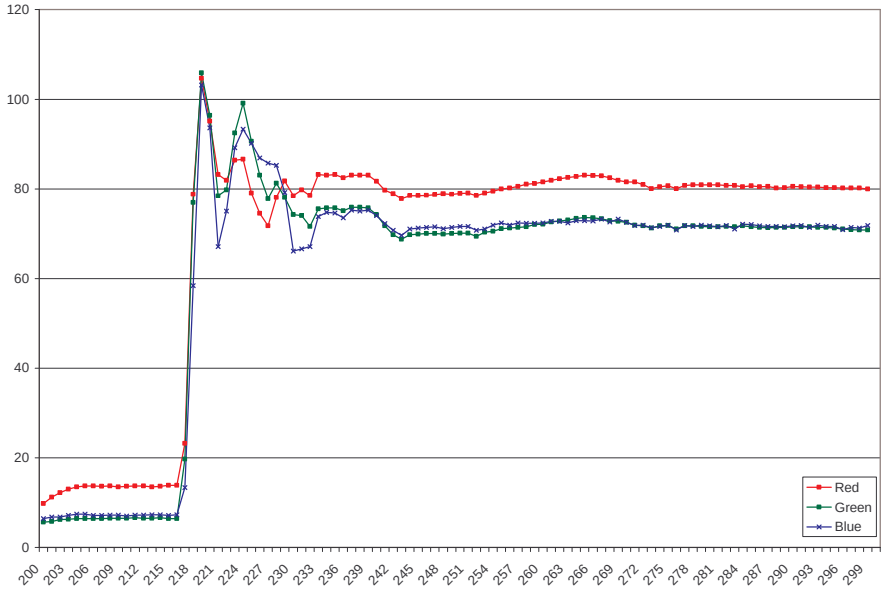

Figure 79 Standard deviation curves for the sample special effect sequence

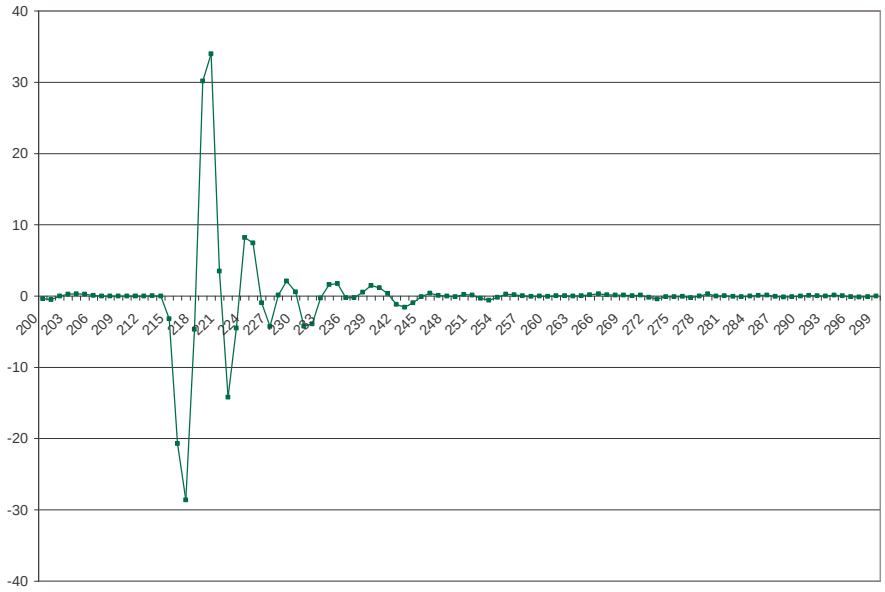

Figure 80 Standard deviation slope difference for the sample special effect sequence

Conclusions

The fade detection algorithm presented in this section is based on the average standard deviation of pixels in video frames. Our method is very simple, uses only two criteria on standard deviation, and achieves a very high performance level of recall and precision close to 100%.

Our approach to detecting fade boundaries is an improvement over the linear regression method proposed by Lienhart [Lie99], and is similar to the one introduced by Truong *et al.* [Tru00a]. Unlike Truong, we do not require color mean to change linearly, but rather introduce a condition which guards against cuts being detected as fades. Our algorithm performs equivalently to the methods developed by Lienhart and Truong.

2.6.3 Dissolve detection

Overview

Next to computer generated special effects, dissolves are the most difficult transitions to detect. As presented earlier, no spatial or temporal separation exists between the shots surrounding this type of transition.

In essence, dissolves are very similar in nature to fades. In fact, a fade can be thought of as a special kind of dissolve in which one of the shots involved consists of only black (monochrome) frames. We demonstrated earlier that fades can be detected effectively and accurately, by looking for certain typical features surrounding sequences of monochrome frames. Unfortunately, dissolve detection does not share the luxury of such well-defined initial conditions, and we cannot resort to detecting monochrome frames to trigger transition detection.

Dissolves are created by fading one shot to black while the next shot is faded in from black. In order to achieve this, pixel intensities of frames in shot A are modified by a monotonically decreasing function f_A, and pixel intensities of frames in shot *B* are modified by a monotonically increasing function f_B. At any frame during the transition, the intensity of every pixel is defined as the sum of the original intensities of that pixel in shot *A* and *B*.

Many researchers derived different characteristics of dissolve frame sequences, and built detection methods around them. These were described briefly in section 2.2.3. A method presented by Truong *et al.* [Tru00a] achieves the highest reported performance. Since their method relies on variance, which is readily available to us as the square of standard deviation, it became a good starting point for the development of our algorithm.

It can be shown that if the variances of shots *A* and *B* are constant, then the color variance of frames over time during transition should have a parabolic shape. This corresponds to the decrease in color intensity which occurs during fade-out of shot *A*

and fade-in of shot *B*. In practice, this idealized shape is somewhat perturbed by change (such as motion) within shots. Figure 81 shows a dissolve between two relatively static shots of similar variance. For such a transition, the parabolic shape of variance curve is apparent (see Figure 82).

Figure 81 Example of a dissolve between shots of similar variances

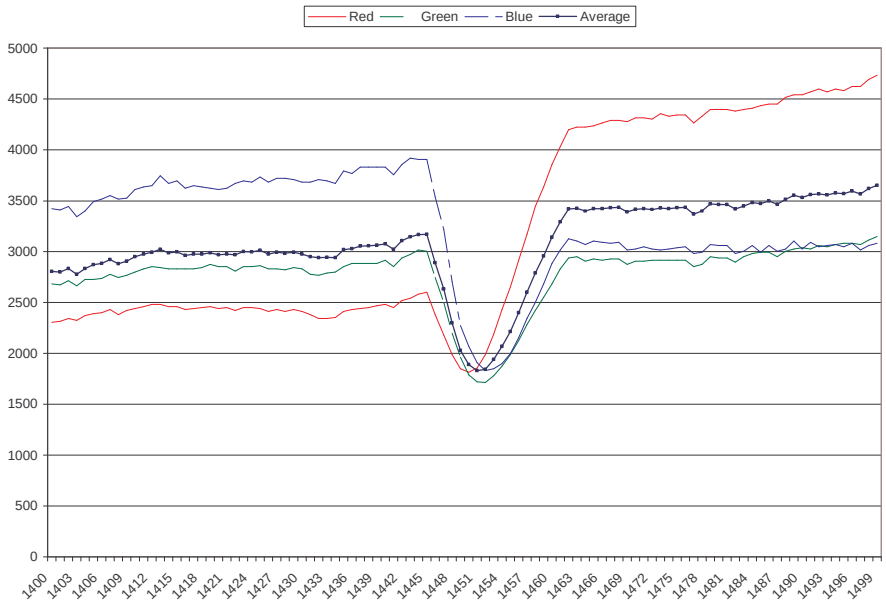

Figure 82 Variance curve for the sample dissolve between shots of similar variances

On the other hand, if the two shots involved have significantly different variances, then the parabolic shape of the variance curve becomes asymmetric, and is more difficult to identify. This situation often happens when a studio shot transitions into a field shot (see Figure 83). The former is filmed indoor, presents a close-up of anchor persons and contains several distinct objects of high color intensity, while the latter has been taken outside and may have been recorded in poor lighting conditions. Such asymmetry in the variance curve (see Figure 84) leads to difficulty in detecting the minimum, which usually is the starting point for dissolve detection. In the extreme

case, such as is shown in Figure 85, the parabola of the variance curve does not appear at all, and therefore no minimum exists during a dissolve (Figure 86).

Figure 83 Example of a dissolve between shots of different variances

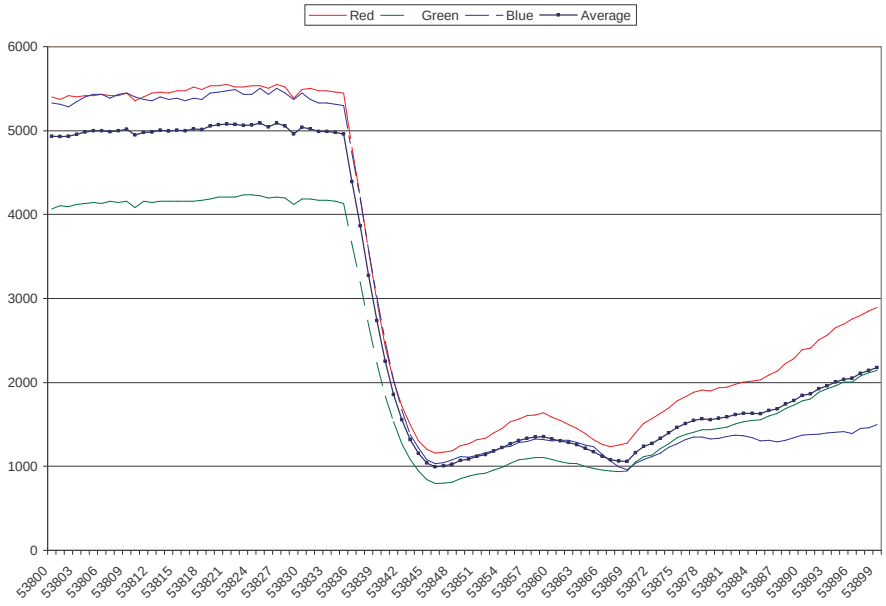

Figure 84 Variance curve for a dissolve between shots of different variances

Figure 85 Example of a dissolve between shots with extremely different variances

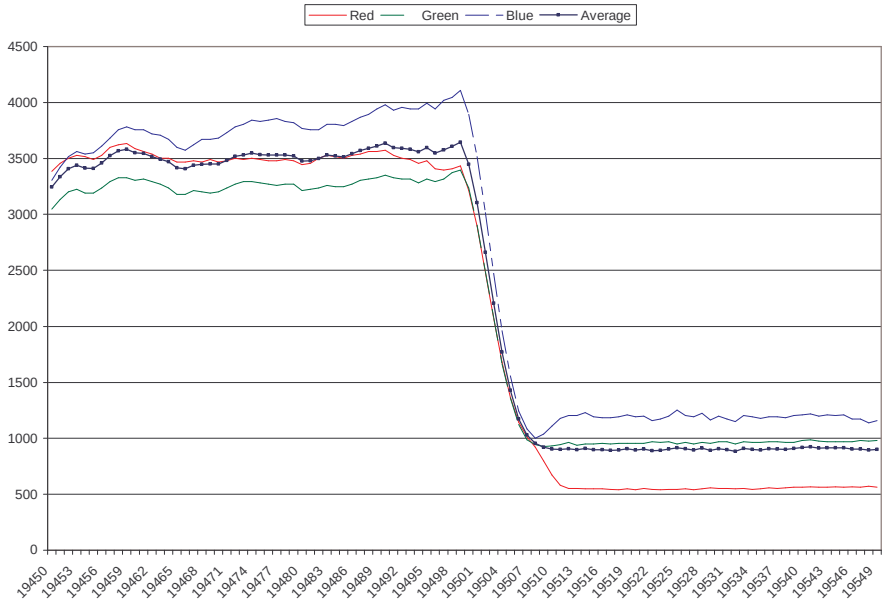

Figure 86 Variance curve for a dissolve between shots with extremely different variances

Television news stations tend to use dissolves to introduce prerecorded footage, as well as separate individual shots of that footage. Frequently, a sequence of live studio shots presenting an anchor person introduces the story, followed by a dissolve and a sequence of prerecorded shots, connected by dissolves. Finally, at the end of the prerecorded sequence, another dissolve brings back the anchor person in the studio. As we argued earlier, such prerecorded footage is essential to our task of story tracking, because it contains the visual clues connecting episodes of the same story. It follows that precise dissolve detection is very important in this work.

The task is complicated by the nature of the prerecorded video material. First, as we noted, the lower quality of the video, such as light exposure, causes difficulty in recognizing transitions between studio shots and field shots. In addition, field shots are often recorded from a hand-held camera, and contain rapid motion, due to the movement of either the camera, or the objects in view. Finally, the view is often temporarily occluded by persons passing directly in front of the camera. The effects of such occurrences on frame variance are very similar to the ones caused by dissolves, as it is shown in Figure 87. In both cases, the color variance function

gradually decreases, reaches a minimum (at maximum occlusion, or maximally monochrome background) and then gradually recovers.

Figure 87 Example of a camera motion which produces a dissolve-like shape of variance curve

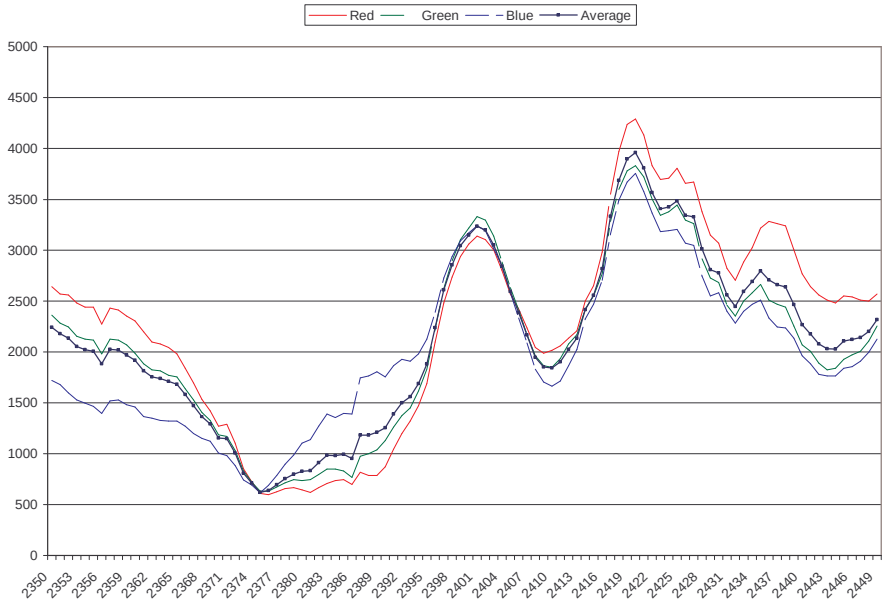

Figure 88 Variance curve for the sample camera motion sequence

The best dissolve detection performance reported in literature is reported by Truong *et al.* [Tru00a] who claim recall and precision of approximately 65% for news sequences. Considering the importance of precise dissolve detection in our work, we need to create an algorithm which performs better. We decided to begin with Truong's method, analyze it in detail, and improve upon it.

Essentially, his approach focuses on identifying trigger points for all potential dissolves and applying multiple conditions separating actual dissolves from effects of motion and noise. Truong introduces the following set of conditions:

1. The variances of the two shots surrounding a potential dissolve must exceed a predetermined threshold.
2. The first order difference of the variance must have two negative spikes on either side of the potential dissolve. The minimal values should exceed a given threshold.
3. The variance differences between the bottom of the variance curve and the left and right ends of the dissolve must exceed another threshold whose value is proportional to the variance of the corresponding shot.
4. Average variance difference between start (end) and the bottom of the curve should exceed half the average variance at the start and end of the potential dissolve.

All of the above conditions were derived algebraically from the mathematical model of an idealized dissolve. Subsequently, however, Truong relaxed some of them in order to deal with real world data. We examine Truong's conditions on our experimental data, and make improvements where needed. First, we assume that dissolve sequences must have certain minimal length. In our experimental data we found that virtually all dissolves contained at least five frames, and assume this value as the required minimal length. Although shorter dissolve sequences do exist, they are extremely rare. In addition, the color moment differences between consecutive frames of such short dissolves are very large, which leads to their detection as cuts.

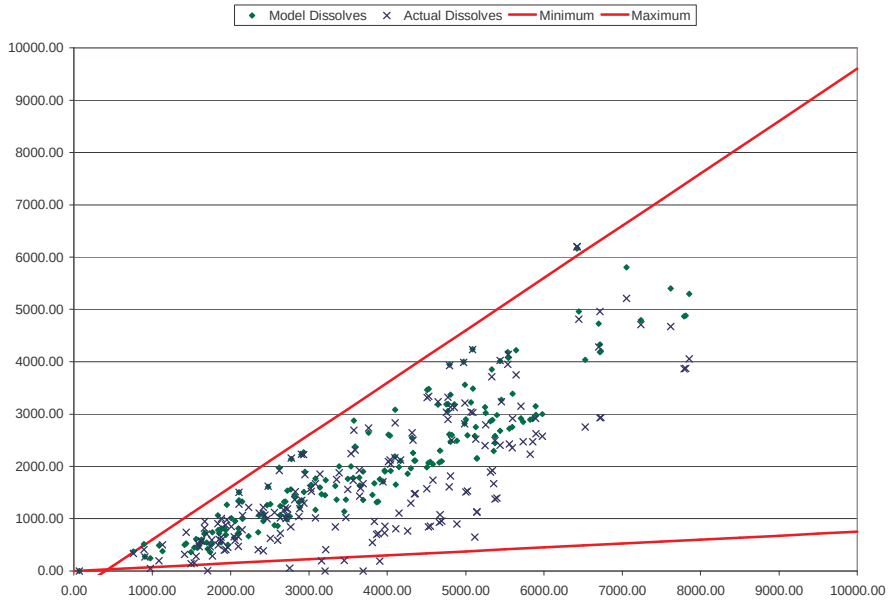

Figure 89 Differences between the start and the bottom of the variance curve during dissolve as a function of the variance of the start frame

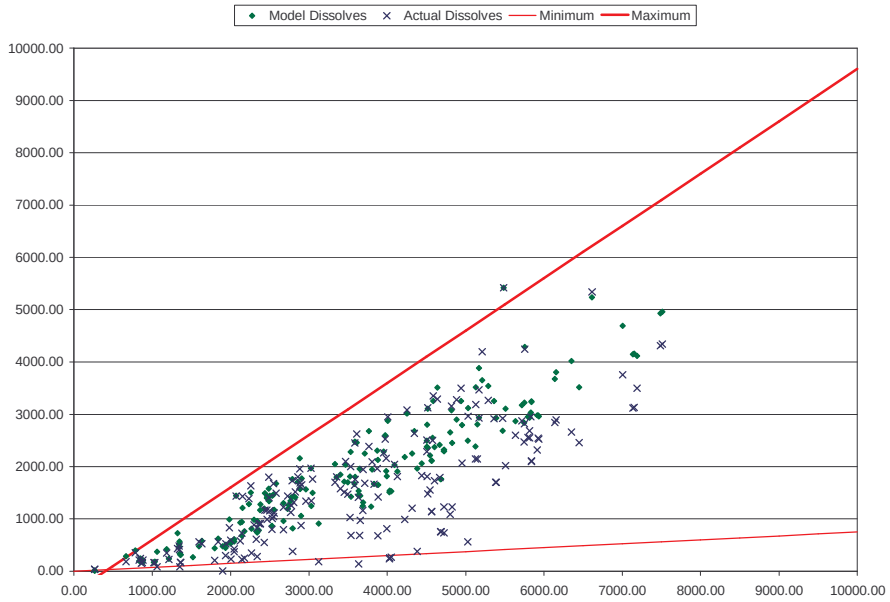

Figure 90 Differences between the end and the bottom of the variance curve during dissolve as a function of the variance of the end frame

Figure 89 and Figure 90 show plots of variance differences between the bottom of the variance curve and its start and end, respectively. The plots show that the theoretical predictions made by Truong do not match the actual values very well. Most of the

time the model overestimates the difference between the minimum and either start or end of the dissolve. As a result, we adjusted the cut-off line proposed by Truong to accommodate the lower difference values.

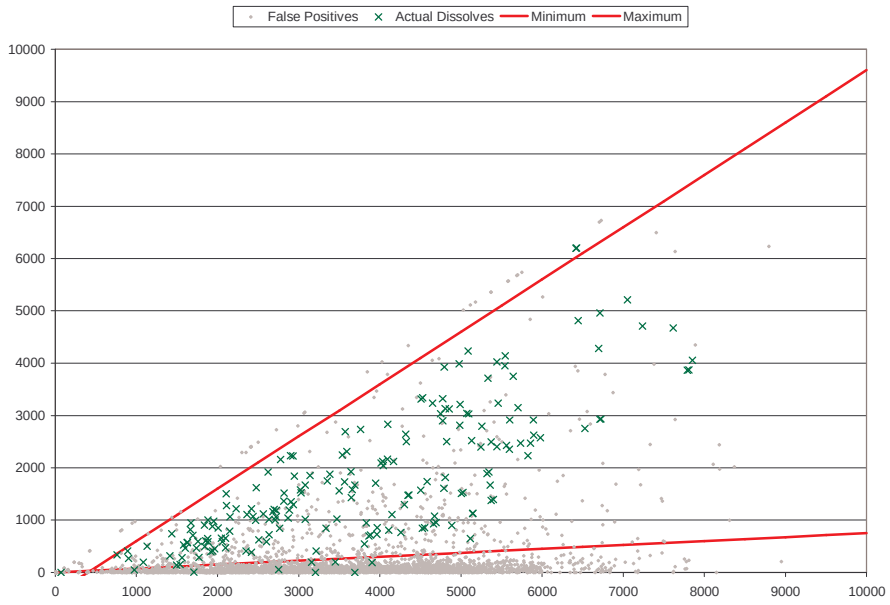

Figure 91 Differences between the start and the bottom of the variance curve during dissolves and non-dissolve sequences as a function of the variance of the start frame

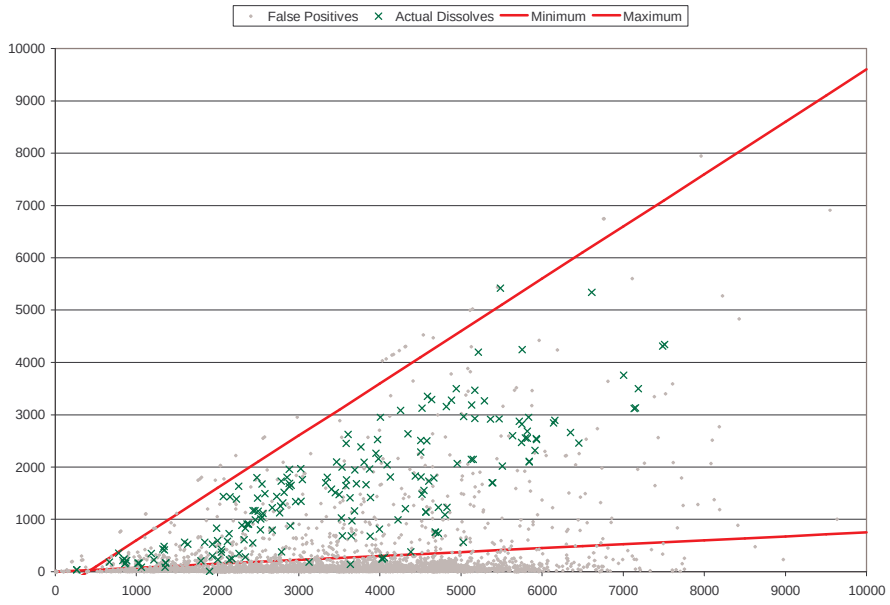

Figure 92 Differences between the end and the bottom of the variance curve during dissolves and non-dissolve sequences as a function of the variance of the end frame

The plots in Figure 91 and Figure 92 show the variance difference values for the actual dissolves along with the values for false positives. It is apparent that the actual dissolves do not differ significantly from the false alarms, as their corresponding points are clearly intermixed. We can observe, however, that the vast majority of false positives have very low difference values. Therefore, we can introduce a threshold line of a very small slope, proportional to the variance of the corresponding shot. Such a cut-off allows us to eliminate the majority of false alarms, while excluding only a small number of actual dissolves. We found the slope value of 0.075 to be effective for our experimental data.

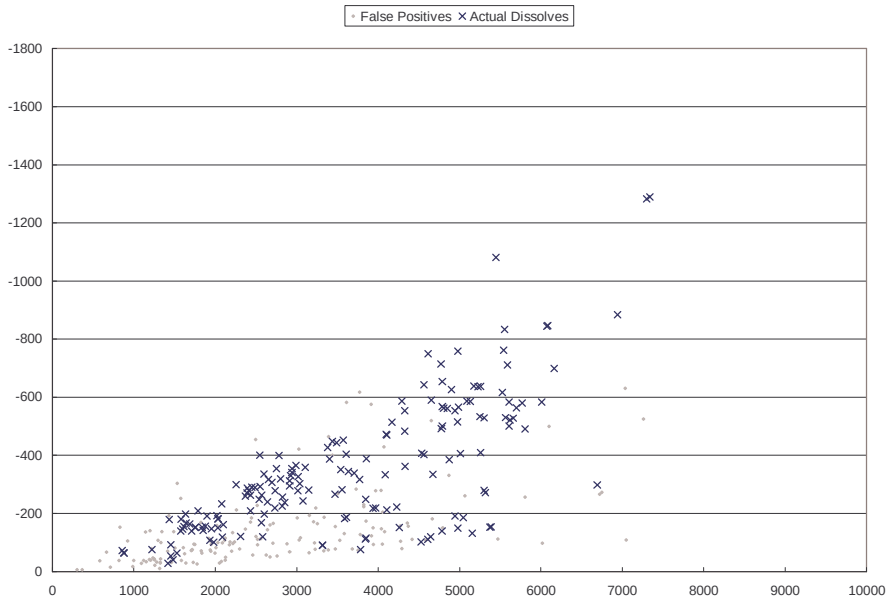

Figure 93 Minima of first derivative of variance at the start of a potential dissolve

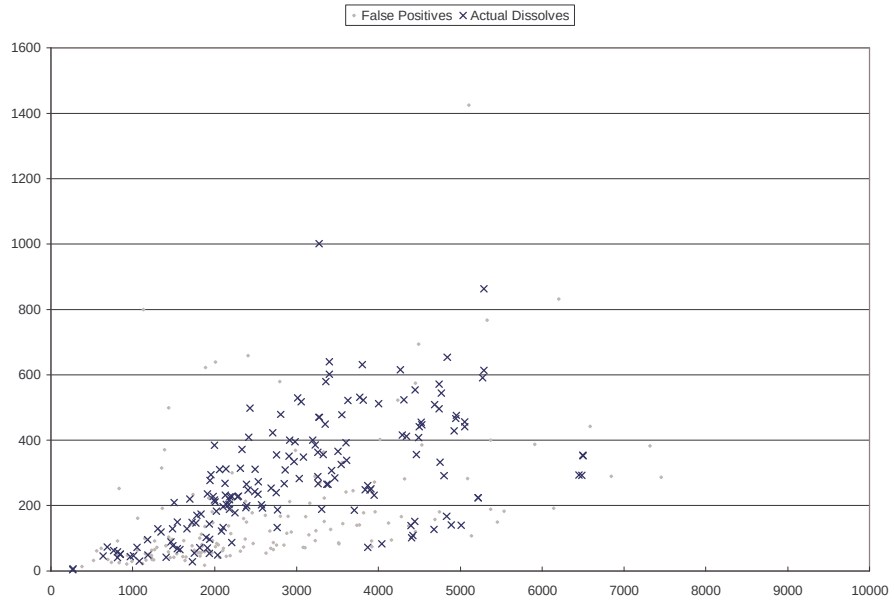

Figure 94 Minima of first derivative of variance at the end of a potential dissolve

Again, no separation is offered by the condition on the minima of the first order variance difference. In fact, the plots in Figure 93 and Figure 94 show that there is no way to separate the actual dissolves from the false positives which remain after the previous condition has been applied. Therefore, we elect not to use this condition in our algorithm.

Then, we examine the condition of average difference in variance. Figure 95 shows that this condition offers some degree of separation. Truong proposes a linear threshold dependent on the average variance at start and end. While this works, a constant threshold offers better overall improvement, if we measure improvement as the ratio of increase in precision to decrease in recall.

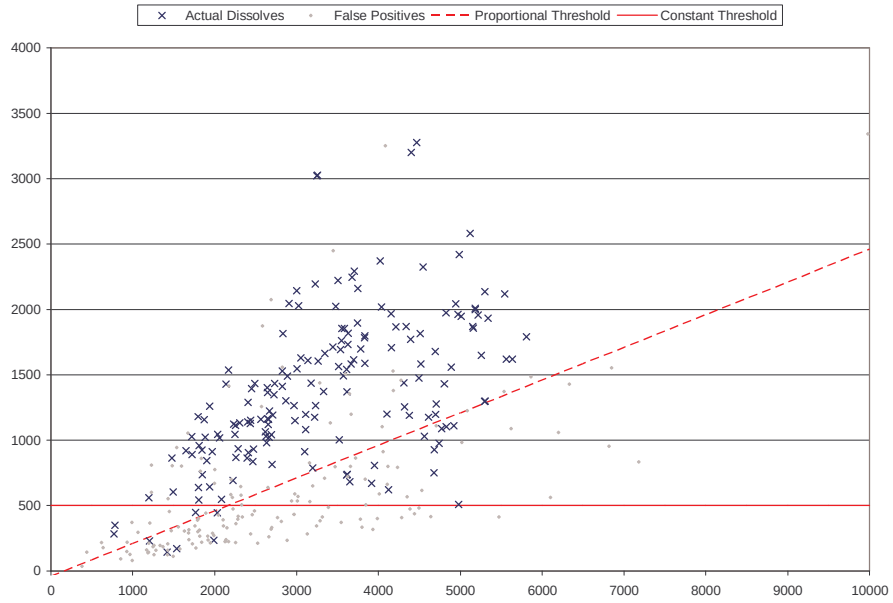

Figure 95 Average variance difference as a function of the average variance at both ends of a potential dissolve

After all of the conditions proposed by Truong have been applied, a large number of false positives remain. In an effort to reduce this number, we analyze the characteristic features of the color mean of video frames during a dissolve. It can be shown that during an idealized dissolve the mean values of individual color components change linearly from the mean of the shot before the dissolve, to the mean of the shot after the dissolve.

We also observe that false alarms are often caused (and impossible to distinguish from dissolves based on variance alone) by large objects passing before the camera, or in the background. If such an object is relatively dark and monochrome, then it causes a gradual decrease in variance as it comes into view, followed by a gradual increase as it leaves the view. This also induces a gradual decrease in color mean.

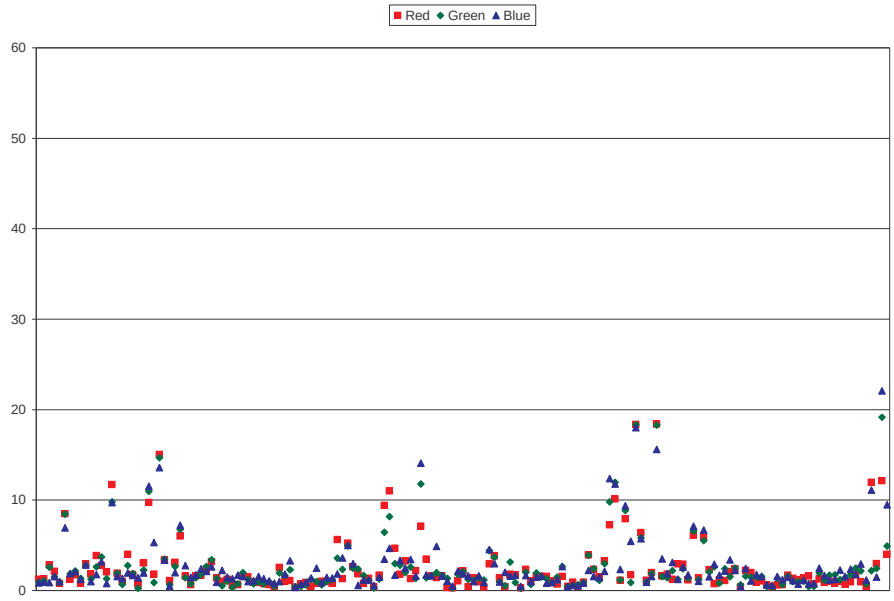

Figure 96 Aberration of the mean curve its linear interpolation for true dissolves

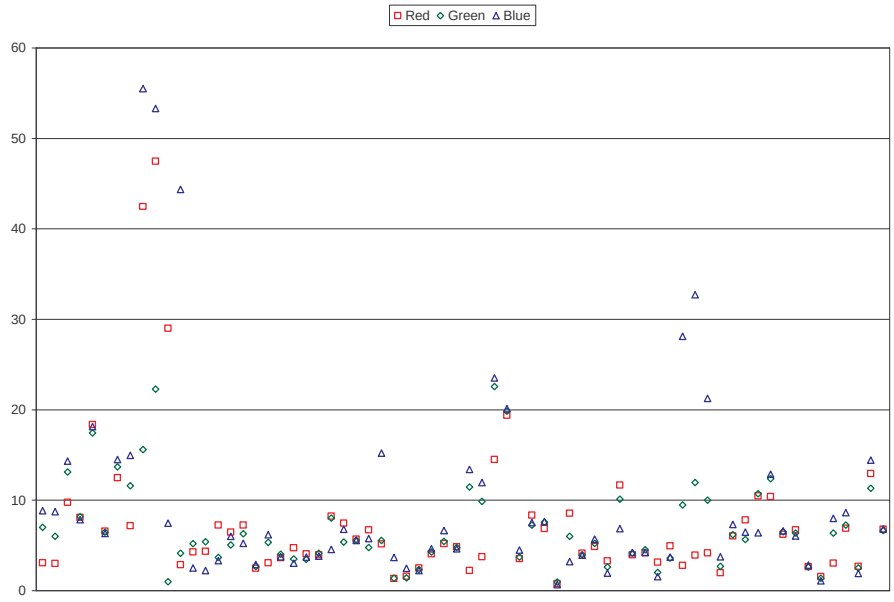

Figure 97 Aberration of the mean curve its linear interpolation for non-dissolve sequences

We use these observations to add a new criterion for dissolve detection. We compare the value of mean at the center of the potential dissolve to the minimum of the values at the start and end of the dissolve. We require that the center value exceed the minimum by an amount proportional to the absolute difference between the mean at

the start and the end of the dissolve. The graphs in Figure 98 through Figure 100 present the value of this difference as a function of the total difference between the means of the two shots involved.

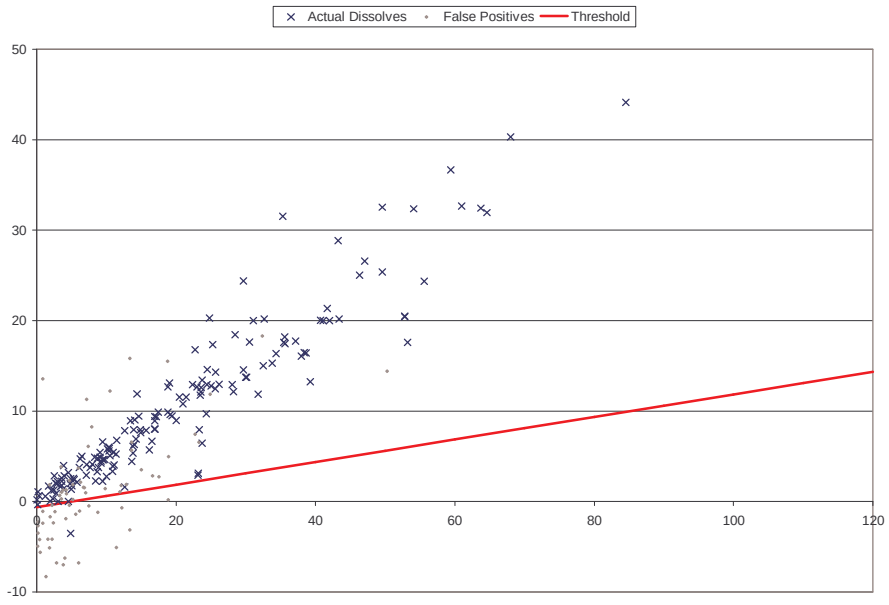

Figure 98 Center mean difference of the red component

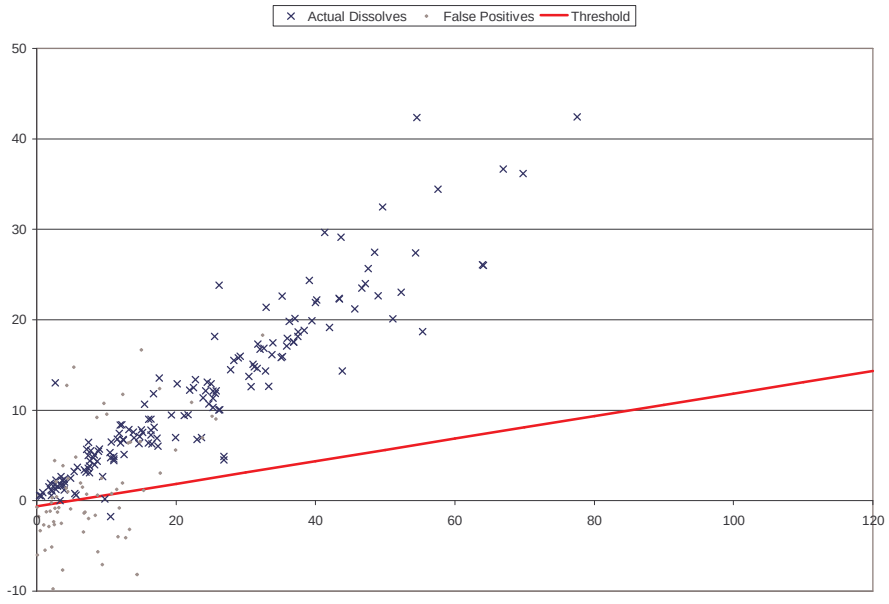

Figure 99 Center mean difference of the green component

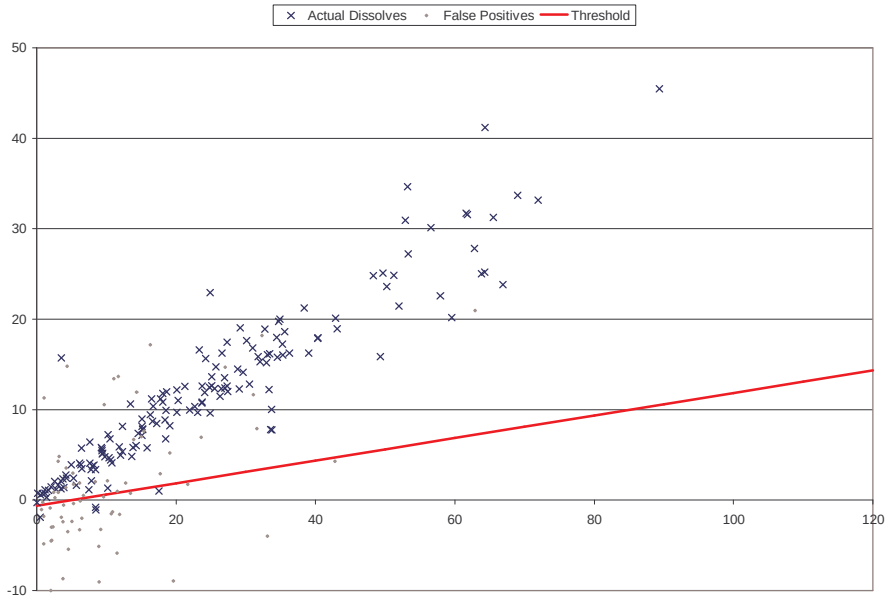

Figure 100 Center mean difference of the blue component

Algorithm

In this section, we describe in detail the dissolve detection algorithm. Due to a large number of criteria used by the algorithm, as well as a variety of calculated values, presenting the algorithm in pseudo code would be impractical. Instead, we provide a detailed list of steps taken by the algorithm.

For every frame in the video clip, perform the following steps.

1) Check if the average standard deviation of the current frame is minimal within +/- 2 frames. If it is, proceed to step 2. Otherwise, move on to the next frame.

2) Make sure the standard deviation of the current frame exceeds the monochrome threshold of 15.0. If it does, proceed to step 3. Otherwise, move on to the next frame.

3) Establish a window of 90 frames around the current frame. For every frame in the window:

   a) Calculate average color variance as the square of the average standard deviation.

b) Calculate first order variance difference, by taking the difference in variance between frame *i* and *i-1*.

c) Compute first derivative of variance by smoothing the first order difference using weighted average with mask [1, 2, 4, 2, 1].

4) Starting at the current frame find the first minimum of first derivative of variance to the left (*dVarMin*), and the first maximum to the right (*dVarMax*). Mark their respective frame numbers as *dVarMinFrm* and *dVarMaxFrm*, and calculate the center frame (*centerFrm*) as the average of the two.

5) Calculate the second derivative of variance as the difference between average first variances of 3 frames to the right and to the left of frame *i*.

6) Find the first minimum of second derivative to the right of *dVarMax* and to the left of *dVarMin*. Mark their respective frames as *dissolveStartFrm* and *dissolveEndFrm*.

7) Calculate variance differences between the *dissolveStartFrm* and current frame (*startVarDiff*), as well as *dissolveEndFrm* and current frame (*endVarDiff*). Compute the average variance difference (*avgVarDiff*) as the average of *startVarDiff* and *endVarDiff*.

8) Test if the sequence of frames between *dissolveStartFrm* and *dissolveEndFrm* meets the following criteria. If all conditions hold, declare a dissolve and move on to the first frame after *dissolveEndFrm*. Otherwise, move on to the next frame.

a) Dissolve length (*dissolveEndFrm – dissolveStartFrm + 1*) exceeds 5 frames.

b) Start variance difference and end variance difference exceed their respective thresholds:

*i) startVarDiff >= 0.075 * startVar*

*ii) endVarDiff >= 0.075 * endVar*

c) Average variance difference is greater or equal to *500.0*.

d) The value of mean for every color component of centerFrm exceeds its respective threshold

   i) *ThreshC = 0.15 * Abs(dissolveStartMean – dissolveEndMean) – 0.5*

   ii) *MeanC >= Min(Mean(dissolveStartFrm), Mean(dissolveEndFrm)) + ThreshC*

e) Mean aberration for every color component is less than or equal to *20.0*

Experimental Results

In this section, we present and discuss the results obtained by using our dissolve detection algorithm on the experimental data. Due to a large number of parameters controlling the algorithm we decided not to optimize the value of each parameter by performing detection with a range of values. Instead, we analyzed the graphical representation of each parameter, as shown earlier in this section. We believe that further adjustment of the parameters to obtain somewhat better results, although possible, would likely lead to over fitting of the model to the experimental data.

Moreover, we present and discuss the performance gain offered by each of the conditions introduced earlier. Table 5 summarizes the results.

It is easy to see that dissolve detection lacks the benefit of monochrome frames which triggered the fade detection algorithm. Dissolve detection algorithm is initiated for every minimum of standard deviation. Evidently, due to motion and general changes on screen, video sequences contain an enormous number of such minima. Reporting each of them as a dissolve would lead to very poor precision. Therefore, a dissolve detection algorithm must apply additional criteria to reduce the number of false positives, and improve precision.

Condition	Match	False Alarm	Missed	Recall	Precision	Utility
Minimum Variance	186	5786	3	98.4%	3.1%	50.76%
Minimum Length	185	3410	4	97.9%	5.1%	51.51%
Min Bottom Variance	184	3345	5	97.4%	5.2%	51.28%
Start/End Variance Diff	170	194	19	89.9%	46.7%	68.33%
Average Variance Diff	164	95	25	86.8%	63.3%	75.05%
Center Mean	158	45	31	83.6%	77.8%	80.72%
Mean Aberration	157	42	32	83.1%	78.9%	80.98%

Table 5 Dissolve detection performance with increasing criteria set

We first restricted the length of dissolves we would like to detect by introducing a minimum threshold of 5 frames. Dissolves shorter than this do occur, but were very rare. Table 5 shows that imposing the minimum length threshold made the algorithm miss just one additional dissolve in our experimental data, decreasing recall by 0.5 %. Conversely, increase in precision due to this condition is almost twofold.

Subsequently we required that the bottom frame of every dissolve had the standard deviation of at least 15.0. This requirement is consistent with the one applied in fade detection to recognize monochrome frame. Hence, this condition eliminates fades that may have otherwise been recognized as dissolves. The effect of this restriction is demonstrated in Figure 91 and Figure 92 as the straight line of larger slope. Clearly, we can eliminate several false positives, with only minimal loss of actual dissolves.

In the next step, we apply the criterion on start and end variance difference proposed by Truong *et al.* Upon examination of Figure 91 and Figure 92 as well as numerical optimization, we arrived at the optimal threshold line $y = 0.075 * x$. It is apparent that the slope of this line is significantly lower than the value 0.25 suggested by Truong. This adjustment is necessary to account for the discrepancy between the theoretical model and empirical data. Limiting dissolve detection to only such sequences whose variance difference between start and bottom, as well as end and bottom frames falls above the chosen threshold line dramatically increases precision,

which now exceeds 45%. On the other hand, we needed to accept a 7.5% drop in recall.

From this point on, we need to considerably reduce the number of false positives in order to achieve acceptable precision, but we cannot afford to lose much more recall, as it is already down to below 90%. Therefore, as discussed earlier, we reject Truong's condition on minimum and maximum of the first derivative of variance. Any gain in precision from using it would come at the price of substantially reduced recall (see discussion earlier in this section). Hence, we move on to the average variance difference threshold. Analyzing the plot in Figure 95, we determined that the cutoff value of 500.0 offers a very good separation of false positives from dissolves. Applying this threshold, we can reduce the number of false positives by 50%, while missing only 6 additional dissolves. This leads to a recall and precision of 86.8% and 63.3%, respectively. Careful examination of the same graph reveals that the threshold value could be increased to 550.0 in order to reduce the number of false alarms. It would, however, most likely lead to over fitting of the model to the experimental data, and would ultimately result in decreased performance for other video sequences.

We note that using a constant threshold for the average variance difference produces much better results than applying any type of threshold proportional to the average start and end frame variance, which Truong proposes. Figure 94 demonstrates that any line of slope greater than 0.0 which eliminates a substantial number of false positives, simultaneously excludes a large number of true dissolves.

Precision of 63.3% offered by the current set of conditions is far from satisfying. Therefore, we attempt to improve its value further, by employing the criterion of mean linearity during dissolve. Figure 98 through Figure 100 show clearly that for many false positives, the values of mean of individual color channels drop below the minimum values at the beginning and the end of the sequence. Whereas, during true dissolves the mean changes approximately linearly from start to end. We choose to impose a threshold on the value of the mean at the center of a potential dissolve

proportional to the absolute mean difference between the start and end frame. Specifically, we use the line $y = 0.15 * x - 0.5$, which has been determined experimentally to give optimal results. This reduces the number of false positives by over 50% while excluding 6 additional dissolves, yielding recall of 83.6% and precision of 77.8%.

Finally, we limit on mean aberration, i.e. the distance between the actual mean curve and the linear interpolation between the values at the start and the end of a potential dissolve. The graphs of mean aberration for red, green, and blue channels, for both true dissolves and false positives, are presented in Figure 96 and Figure 97, respectively. From the two graphs, we can obtain the value of 20.0 as a good threshold. Three more false positives can be eliminated by applying this threshold.

Performance improvements due to all of the restricting conditions discussed in this section are shown graphically in Figure 101. Utilizing all of these conditions our algorithm achieves an overall recall of 83.1% with 78.9% precision, which constitutes a substantial improvement over results presented in literature.

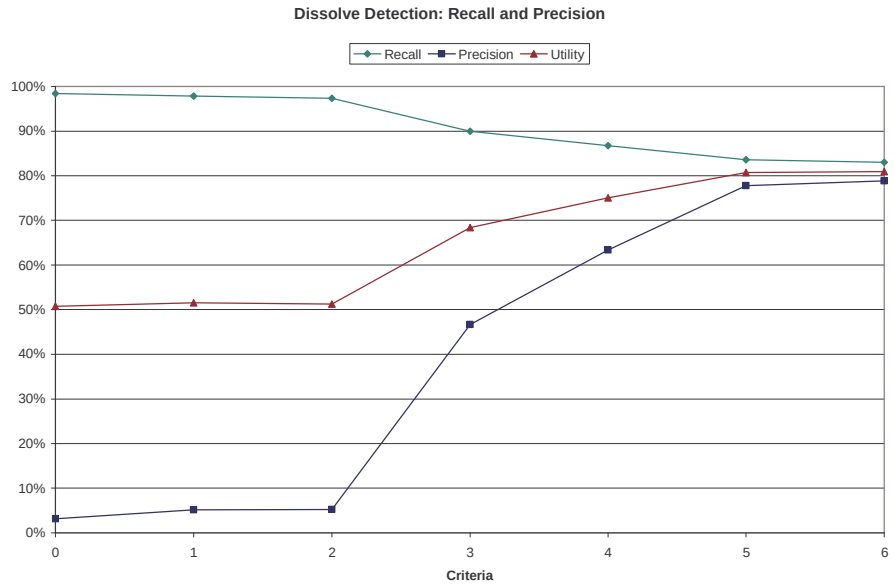

Figure 101 Dissolve detection performance with increasing number of criteria applied

Conclusions

In this section, we presented a dissolve detection algorithm based on mean and variance of color components of video frames. Our method is rooted in the approach proposed by Truong *et al*. [Tru00a], but contains several improvements.

Our contribution has two aspects. First, we conducted a detailed analysis of the model of the dissolve transition and confronted it with empirical data. We present criteria used by Truong and evaluate them with respect to our experimental data. As a result of this evaluation we drop one of criteria as not useful, and modify another to achieve better performance. Second, we introduce two additional criteria, derived from the properties of color mean, which vastly reduce the number of false alarms, and thus increase precision.

2.6.4 Combining transition detection algorithms

In this section we first present the combined performance of our temporal segmentation algorithm. The summary of recall and precision of the detection techniques for all three types of transitions is presented in Table 6. These results were obtained for the experimental 1-hour sequence with the optimal set of parameters.

	Match	False Alarm	Missed	Recall	Precision	Utility
Cuts	528	31	59	90%	94%	92%
Fades	78	2	6	93%	98%	95%
Dissolves	157	42	32	83%	79%	81%
Total				**89%**	**91%**	**90%**

Table 6 Combined temporal segmentation performance on the 1-hour experimental sequence

In order to verify the performance of our method we chose a different 10-minute sequence of CNN News and ran the transition detection algorithm with the same set of parameter values. The results of this experiment show (see Table 7) that our technique attains similar and better performance in detection of cuts and dissolves. Fade detection achieves 100% recall, but its precision is lower than on the original

sequence. The test sequence contained only three actual fades, while also including two unusual clips in which a camera pan produced a fade-like effect. These two clips were erroneously detected as fades, and their impact on precision was exaggerated by the small number of actual fades. In a larger group of fades we expect a high precision value.

The two experiments conducted allow us to conclude that our temporal segmentation technique works consistently well on different news video sequences.

	Match	False Alarm	Missed	Recall	Precision	Utility
Cuts	46	2	0	100%	96%	98%
Fades	3	2	0	100%	60%	80%
Dissolves	28	3	2	93%	90%	92%
Total				**97%**	**92%**	**95%**

Table 7 Combined temporal segmentation performance on a 10-minute test sequence

After all individual types of transitions have been detected, there remains one final step before the temporal segmentation is fully accomplished. The disparate sets of transition frames must be combined, so that a consistent set of shot frames can be established. Since transition detection for cuts, fades, and dissolves is performed independently, it is possible that some frames marked as belonging to one type of transition are also denoted as belonging to another type. All such conflicts must be resolved before the temporal segmentation is completed.

Our approach to transition conflict resolution consists in assigning priorities to different types of transitions according to our confidence in their respective detection method. Due to a high performance of our cut detection method, we give cuts the highest priority. Fades receive a medium priority, and finally dissolves have low priority. If any two transitions of different types overlap, we keep only the one with higher priority, while ignoring the other.

The remaining transition frames are combined to form a single set *T*. The set of all frames that do not belong to *T* forms the set of shot frames *S*. These two sets define the final temporal segmentation of the video clip.

2.7 Conclusions

In this chapter, we described the problem of temporal video segmentation with emphasis on its application to video news broadcasts. We presented some approaches available in literature and proposed two of our own solutions. Both solutions were based on color moments as video frame representation, and adhere to our requirement of real-time execution. We tested both algorithms on a one hour block of video from a typical day of CNN News broadcast and analyzed their performance.

First, we introduced a simple algorithm, which attempted to detect all types of transitions using one method consisting in assessing a cross-difference between video frames up to a certain distance apart. We analyzed its performance on the experimental data, and concluded that due to inter-frame differences caused by noise and motion, the algorithm cannot achieve acceptable levels of recall and precision.

In order to provide a better distinction between the effects of noise (or motion), and actual transitions, we adopted a method based on transition model. This method aims to detect different types of transitions by identifying certain characteristics of frame sequences, which can be derived from the transition models. The algorithm was tested on the experimental data, and achieved very good performance, especially for cut and fade detection. For these two types of transitions, both recall and precision exceeded 90%. Dissolve detection proved to be the most challenging, and the algorithm attained a recall and precision of approximately 80%.

Our contribution to the field of temporal video segmentation is twofold. First, we introduce an effective temporal segmentation method based entirely on color moments. This unique approach offers a very compact representation of video, as we only need to store and process nine floating point numbers for every frame. Using this representation we created a very fast one-pass algorithm, which executes in a fraction of real time, and performs at least as well as other approaches presented in literature.

Second, we proposed several improvements in cut and dissolve detection. We introduced a statistically grounded approach to cut detection, which eliminates some systemic problems with existing algorithms and offers improved performance. We also introduced a few additional criteria for dissolve detection, as well as analyzed and modified certain existing conditions. Application of these additional and modified conditions produced a substantially improved performance of dissolve detection (15% increase in recall and precision).

In the following chapters, we will use the temporal video segmentation as a stepping stone to detecting repeated shots and combining them into stories. Therefore, the quality of transition detection will have direct impact on the quality of story tracking we perform. Our story tracking method is based on detecting repeated footage used by news station to provide visual background for the story. In the process of developing and testing our temporal segmentation methods, we observed that such footage is often introduced and separated by dissolves. Thus, the improvements to dissolve detection were essential for the remainder of our work. On the other hand, the 80% recall and precision of dissolve detection may still prove insufficient for a reliable repeated footage detection.

.

Chapter 3

Repeated Video Sequence Detection

3.1 Introduction

As it was discussed in Chapter 1, modern television news stations often reuse video footage when relating a news story. As the story develops and new segments are shown by the station, often the same footage is displayed. This means that a number of video clips presented at some point in time are later repeated as a whole or in part. Detection of such repetitions is a good indication that two news segments are in fact related and convey the same story. It is the focus of our work to detect repeated video footage and use this information to track the development of news stories.

In this chapter, we present the problem of repeated footage detection as it pertains to the task of story tracking. We discuss challenges involved in identifying fully or partially repeated video sequences, and present a number of methods for effective detection of repeated video material.

3.1.1 Problem Definition

First, we introduce some basic terminology and notation. Throughout this chapter we frequently refer to video sequences, subsequences, clips, and shots. In order to avoid confusion, we provide their precise definitions.

Definition 1: A video sequence S of length N_S is a sequence of N_S consecutive video frames:

$$S = \left\langle f^1, f^2, \ldots, f^{N_S} \right\rangle$$

The length of the video sequence is denoted by |S|.

Thus, f^t denotes the t-th frame in a video sequence and is equivalent to *S(t)*. In the context in which video frames are considered independently of any video sequence the superscript is omitted. In addition, if more than one video sequence is considered, frames belonging to different sequences are distinguished by the subscript, for instance f_a denotes a frame from sequence S_a. Finally, in certain situations it is useful to consider a video sequence as a four-dimensional intensity function *I(t,x,y,c)* which assigns a value to every component – red, green, and blue – of every pixel in every frame of the sequence. Summarizing, we can write

$$S_a(t) = f_a^t = I_a(t, x, y, c).$$

Definition 2: A subsequence $S^{g,h}$ of the video sequence S is a sequence of consecutive video frames from S:

$$S^{g,h} = \left\langle f_S^g, f_S^{g+1}, \ldots, f_S^h \right\rangle \text{ and } 1 \le g \le h \le |S|$$

The terms *video sequence* and *video clip* are synonymous and are used interchangeably in this chapter. On the other hand a *video shot*, whose definition was

provided in the previous chapter, is a *video sequence* that was taken from a single camera working continuously, in a single span of time. Equivalently, a *shot* is a *video sequence* which contains no transition frames.

In this chapter, we examine methods of detecting occurrences of video sequences (*query sequences*) in other video sequences (*source sequences*). In order to accurately define the problem at hand, we distinguish two classes of repeated sequence detection:

1. ***Single repetition detection***. In this task, we are interested in finding a query sequence (usually short) in a long source sequence. The query sequence usually contains a small number of shots, and may represent a commercial, for instance.
2. ***Exhaustive repetition detection***. The goal of this task is to detect all repetitions of all subsequences of a given long video sequence. In this case, the query and source sequence may be the same, and may come from a live broadcast. As a result, the detection algorithm must run in real-time.

These two classes correspond to the following definitions.

Definition 3: Let S and Q be sequences of video frames of length N_S and N_Q, respectively. The task of ***single repeated footage detection*** *consists in identifying all subsequences of $S^{k,k+n}$ of the source sequence S, such that Q matches $S^{k,k+n}$ and $n = N_Q$. The task of* ***exhaustive repeated footage detection*** *consists in identifying all subsequences $Q^{g,g+n}$ of Q, such that there exists a subsequence $S^{k,k+n}$ of S which matches $Q^{k,k+n}$.*

The concept of matching video sequences will be discussed in depth in section 3.2.3. For now, we can assume that sequences of frames that match are simply identical. The notion of frame sequence identity appears intuitive, but requires a formal definition, which will be given later in section 3.2.1.

Each type of repeated sequence detection task may have two additional aspects, which influence the choice and performance of detection techniques.

1. ***Temporal segmentation***. The results of temporal segmentation of video sequences *Q* and *S* may be available, thus providing a list of shots contained in both sequences.

2. ***Partial repetition***. One may be interested in identifying partial repetitions of the query sequence *Q* (for single detection task) or partial repetition of the clips in *Q* (for exhaustive detection task).

The techniques used for repetition detection depend on the class of detection task involved, as well as the two aspects listed above. Naturally, exhaustive detection is much more demanding than single detection. Both tasks are simplified by the presence of the shot structure of *Q* and *S*, and different approaches are required for complete and partial detection. Further details regarding repeated footage detection algorithms will be given in section 3.3.

In this work, we are primarily interested in repeated sequence detection as an element of a method for story tracking in live video news broadcasts. Therefore, our detection methods must solve the problem of exhaustive detection, in which the query sequences *Q* and *S* are the same.

3.1.2 Related Work

The problem of repeated video sequence detection falls within the scope of the broader domain of video retrieval, which is concerned (in the most general sense) with identifying video material relevant to some information need. Much of the research on video retrieval has focused on search for conceptually similar material. For example, when given an image or video clip of a sailing boat, any clips of sailing might be regarded as a match. A standard method for addressing this task is to use image comparison techniques to seek frames with similar content. In contrast, a different type of similarity is considered when searching for clips with the same

footage – other footage on a similar topic is not a match. For example, given material from the movie "*Apollo 13*", news footage of the Apollo program is not a match, nor is material from "*The Right Stuff*". However, the "*Apollo 13*" material is a match in widescreen, standard format, or after removal of the color signal. This task can be described as matching of co-derivatives. Thus, two distinct types of video similarity may be distinguished:

1. ***Semantic similarity***. Two video sequences are semantically similar if they represent or describe the same or similar concept.
2. ***Co-derivative similarity***. Two video clips are co-derivatives if they have been derived from the same original video sequence.

Semantic similarity has been the focus of considerable research efforts generally classified as content-based video retrieval. Matching and retrieval of co-derivatives has been explored in other domains, such as text [Man94, Shi95], but the problem has received little attention in the multimedia domain.

Semantic Video Retrieval

Semantic video retrieval comprises efforts in a broad area of research concerned with providing people with effective and intuitive access to information contained in video. In order to make video intuitively available, video retrieval systems must respond to the information need of their users expressed in terms of high level concepts natural to human beings. This, in turn, requires that video be first automatically processed and appropriately organized. Therefore, research in the area may be further classified into the following categories.

i. ***Video comprehension***, which comprises efforts to understand the semantic content of video material and capture the intuitive notion of what the video is "about".

ii. ***Video organization***, which deals with issues of recognizing certain structural properties of video material, such as related video clips, spatial and

temporal relations in video, etc. Video organization also investigates mechanisms of efficient video storage.

iii. ***Video retrieval***, which attempts to provide methods of effective and intuitive searching and browsing of video content.

All of the research categories above are closely related. Naturally, no effective video retrieval method can be proposed without some degree of video comprehension. Similarly, video retrieval systems must, in most cases, be coupled with some underlying video organization and storage.

It has long been known that a cognitive gap exists between humans and machines. The conceptual reasoning and semantic understanding which comes naturally to the former, is generally unavailable to the latter. This problem is addressed by the research in the field of video comprehension, which aims to enable computers to process visual information according to its semantic content. This ultimate goal can only be accomplished gradually, and video must be analyzed on multiple levels of "comprehension" [Pet01]:

i. Raw video material and metadata: At this level, only raw video units (frames) and some metadata, such as video stream format, frame-rate, and resolution, are available.

ii. Basic visual features: Here belong certain statistical features of the video stream, such as color content and distribution, shapes, textures, and motion.

iii. Conceptual content: This level corresponds closely to human understanding. The video content is described in terms of objects, persons, and events,

At the current stage of development in video comprehension, the raw video material and metadata are easily obtainable from practically every video stream. Basic visual features can generally also be extracted automatically without human assistance. However, the conceptual content extraction requires some level of human interaction.

In recent years, some advances have been made in obtaining limited semantic content from video [Nap00a, Nap03], but the available methods are far from automatically extracting the large variety of concepts accessible intuitively to humans.

Due to the limitations in automated video comprehension, the research in the field of video retrieval relies on lower level video features or metadata associated with the video. A number of video retrieval systems are built around basic video features, such as color, texture, or motion (*VisualSEEk* [Smi96], *Virage* [Ham97], *VideoQ* [Cha98]). Such systems require their user to formulate their information need, as a query expressed in terms of the basic features, which is not intuitive to a human user. Moreover, searches performed using such queries tend to retrieve diverse and conceptually unrelated video clips. Other systems exist, which utilize manually created annotations. Searching in such a system is more intuitive, but limited to the information provided manually. Considering that the annotation process is tedious and time consuming, such markups tend to be very brief, and thus do not represent the depth of the video content. Lately, advances in automatic speech recognition allowed for the text retrieval techniques to be used in video [Pet96, Hac00, Hua00].

With the advances in video retrieval, it became important to provide a standardized performance measurement of video indexing and retrieval systems. Efforts in this area were initiated in 2001 as part of the TREC Video Track (TRECVID). TRECVID workshop provides a large corpus of video material from a variety of news sources. It also offers a consistent performance evaluation methodology for a number of video retrieval tasks. In the last two years, TRECVID emphasized the issues of extraction of high level video features, as well as retrieval based on high level concepts [Hau03, Nap03b].

All three aspects of the semantic video retrieval research are combined in an effort to create comprehensive Digital Video Libraries, which in turn are part of the broader Digital Libraries Initiative funded jointly by several national institutions. The main research project in this area is Carnegie Mellon University's *Informedia* project

[InfWeb], which – in addition to video retrieval – also explores automated speech recognition and annotation, as well as effective means of video presentation.

Co-Derivative Video Retrieval

The domain of co-derivative video retrieval is relatively new and has not been extensively studied. The research efforts in the field concentrate primarily around detection of occurrences of known commercials.

Gauch [GauWeb] developed a commercial authentication system (VidWatch). This system monitors two continuous live video streams in order to determine whether the video content provided for distribution by its owners reaches the audience unchanged. Gauch uses color moments to represent video frames. Given information about the airing times for any given commercial, VidWatch compares the two video sequences at those times and detects potential discrepancies, by comparing the moment representations of their corresponding frames. The system achieves very high accuracy and has been deployed commercially at a major U.S. television broadcasting company.

Pua [Pua02, Pua04] describes a real-time video sequence identification and tracking system, which detects repeated video sequences in a continuous live video stream. Like Gauch, Pua also uses color moments for video frame representation and matching. In order to achieve real-time performance, Pua employs a video frame hashing technique to reduce the number of sequence comparisons. His technique was tested on television broadcasts from two different documentary channels, and produced recall and precision rates of over 90%. Pua also demonstrates that correct identification and removal of repeated video sequences can lead to significant compression of video archives.

Hoad and Zobel [Hoa03a, Hoa03b] focus on identifying repeated video material which may have undergone certain degradation or modification, such as a change in brightness or contrast, and difference in frame rate or frame resolution. They develop three different video similarity detection methods and evaluate their robustness to

such changes. The first of their methods compares video clips according to the pattern of cuts. If the positions of cuts in the two clips are identical, then the clips are considered matching. This method requires that the clips contained at least 5 to 10 cuts. The second approach relies on tracking movement of the centroid of the brightest or darkest pixels across video frames. For every frame in a video clip, a centroid of the lightest 5% or darkest 5% of the pixels is computed, and its displacement from frame to frame is stored as the magnitude of the vector containing the respective centroid positions. Identification of matching video clips then consists in comparing their sequences of vector magnitudes. Finally, the last method uses changes in color between frames as the clip signature, and compares the sequences of changes in order to find matching clips.

Hoad and Zobel tested their methods on a relatively small data sets of about 3 hours of video. The tests consisted of detecting repetitions of a commercial in the video stream, after the original has been perturbed in a variety of ways. The experiments showed that the centroid approach was most robust to video degradation, but even this method was unable to cope with all types of modifications.

In this work, we employ repeated video sequence detection as the central stage in a broader task of story tracking in television news broadcasts. As such, our goals are similar to Pua's, but accomplishing them in the context of news broadcasts provides numerous additional challenges. The domain of television news also substantially limits the applicability of the methods developed by Hoad and Zobel.

3.1.3 Contribution

In this work, we focus on detection of repeated video footage in live broadcasts of modern television news stations for purposes of story tracking. This task is considerably more complex than the problems presented in literature so far. The video footage reused by the television stations is usually quite limited, and tends to be interspersed piece-wise between live studio shots. As a result, video sequences which must be detected in order to link segments of the same story are very short, often

restricted to individual shots only a few seconds in length. This fact substantially reduces applicability and performance of many video sequence matching methods. For instance, the cut pattern method presented by Hoad and Zobel is clearly useless for matching individual shots, which by definition contain no cuts or other transitions.

The limited sequence length places increased requirements on temporal video segmentation. Imprecision and errors in shot detection are of little impact if repeated sequences are several shots long, but become very problematic when repetitions involve only individual shots. Pua's system relies on a very simple segmentation technique, which proves sufficient for the detection of long video sequences his system deals with. The same segmentation method would render our detection task impossible.

In addition, new video footage is often shown not as a whole, but rather in parts whose length is adjusted according to the demands of the live news programming. Consequently, video sequences which need to be compared and matched frequently differ in length and may contain very little overlap. This issue has not been addressed at all by the research in the field.

Finally, news broadcasts are composed, in large part, of the video shots taken in a studio and contain one or more anchor persons directly facing the camera. Such video sequences tend to be relatively static, and in general quite similar to one another. This also limits the choice of video matching techniques that may be used for repetition detection in news videos.

Summarizing, we can identify four major challenges and areas where improvements are needed:

1. Detection of very short video clips, which may consist of only a single shot not exceeding a few seconds in length.
2. Detection of partially repeated sequences.
3. Video similarity techniques capable of dealing with studio sequences.

4. Real-time execution for exhaustive repetition detection.

In this chapter, we address these challenges. First, we analyze issues and develop mechanisms for detection of very short video clips. Then we provide solutions for the problem of partial clip repetition. We introduce a number of video sequence similarity metrics, and examine their application to repeated clip detection. For very short clips, the question of precise temporal segmentation becomes very important. Although we introduced improvements in temporal segmentation in the previous chapter, the automated shot detection is not perfectly accurate. In this chapter, we analyze the impact of imperfect segmentation on detection of repeated footage.

Since we are interested in live real-time detection of repeated material, we focus on improving detection speed. We analyze the heuristic approach introduced by Pua, and make improvements in quantization and hashing, as well as adapt the solution to detection of partial sequence repetition.

3.1.4 Chapter Organization

The remainder of this chapter is organized as follows. Section 3.2 introduces the notion of video sequence similarity as the foundation for the repeated footage detection. The concept is then formalized by a number of frame and sequence similarity metrics. This section is closed by a discussion of the application of these metrics to the problem of complete and partial sequence repetition detection. An overview of the algorithmic methods of repeated sequence detection is presented in section 3.3. The following two sections (3.4 and 3.5) describe, in detail, detection techniques in the absence and presence of the shot structure of the query and source sequences. The heuristic repetition detection approach is introduced in section 3.6. The section first presents the idea of color moment quantization, and discusses its implications for sequence similarity. Later the hashing technique for color moments is presented, and followed by the details of the heuristic video sequence repetition detection algorithm. Evaluation of the detection methods developed in this chapter is given in section 3.7. The chapter closes with concluding remarks in section 0.

3.2 Video Clip Similarity Metrics

3.2.1 Overview

The notion of video clip similarity and identity appears intuitively straightforward. Two video clips are similar or identical if they appear so to the human eye. Such a naïve definition, however, is insufficient for automatic detection of repeated video footage, and a more formal notion of similarity and identity must be developed. In this section, we introduce a number of similarity and identity metrics and discuss their advantages and shortcomings. First, we precisely define the notion of clip identity, which is followed by a number of practical measures of frame similarity. We close by discussing several clip similarity metrics derived from frame similarity.

Before we can detect repeated footage, we need to formally establish the notion of video footage repetition. Intuitively, we could define video footage repetition as the appearance of the same video clip at two different times in a video sequence. While this concept seems clear, it leaves open the question of "sameness" of two video clips. From the human perspective, we can assess "sameness" as identical content in terms of objects, persons, background, camera angle, etc. Such description, however, lacks precision needed for automated detection methods.

In this work, we will adopt the following definition of "sameness":

*Definition 4: Two video clips are the same (**identical**) if they were taken from the same camera, at the same location and in the same time span.*

Hence, we can recognize identical video clips if we know exactly where and when the individual clips were filmed. Unfortunately, we do not generally have direct access to this type of information, and so making a direct determination whether any given two clips are identical using this definition is impossible. Consequently, we need to develop concepts of clip identity which closely reflect Definition 4, and yet can be directly verified using only the two video clips involved. The remainder of this section presents and discusses several alternatives.

Given two video clips S_a and S_b of length N, a straightforward definition of identity could be formulated as follows.

Definition 5: Two video clips S_a and S_b of length N are identical if their corresponding pixels of the corresponding frames are identical, i.e. the values of their primary color components are equal:

$$\forall t = 1,...,N \; \forall x, y, c : I_a(t, x, y, c) = I_b(t, x, y, c).$$

In order to determine whether two clips are identical, one could compare every pixel of every frame of S_a to the corresponding pixel of the corresponding frame of S_b. Such notion of clip similarity, though valid, is impractical due to extremely long computation time and lack of robustness to noise. Obviously, even clips of relatively small length contain a very large number of pixels, and direct pixel comparison would require a very large number of operations. Moreover, in most practical applications, clips compared contain a certain amount of noise, which alters color components of individual pixels. Therefore, we need to develop a more robust and computationally simpler definition of video clip identity.

To accomplish this goal, we develop a number of video clip similarity metrics, which allow us to determine how close in appearance any two clips are. The definition of clip identity can then be derived as sufficient similarity, i.e. two video clips are identical if they are sufficiently similar.

In the remainder of this section, we will frequently make use of color moments: mean (M), standard deviation (S) and skew (K), which were defined in section 2.3. Earlier in this work, we used pixel color components in the range of 0 to 255, and consequently assumed color moments to have the same range. In this section, we will adopt a range of 0 to 1, which simplifies some of the definitions.

3.2.2 Frame Similarity Metrics

Individual Frame Similarity Metrics

We generally express clip similarity based on the similarity of their corresponding frames. The definition above implicitly contains a straightforward notion of frame similarity. Specifically, it deems two frames identical if their corresponding pixels are identical. This could be extended to define frame similarity.

Definition 6: Similarity of two frames f_a and f_b is inversely proportional to the average pixel-wise difference between them:

$$FrmSim_{\alpha}(f_a, f_b) = 1 - FrameAvgPxlDiff(f_a, f_b), \tag{16}$$

where

$$FrameAvgPxlDiff(f_a, f_b) = \frac{1}{3N}\sum_{xyc}\left|I_a(t,x,y,c) - I_b(t,x,y,c)\right|. \tag{17}$$

This notion of frame similarity has a number of shortcomings. First, it is computationally intensive, as it requires a number of operations equal to the number of pixels in the frames compared. More importantly, it is very sensitive to even slight distortions of the compared frames. For instance, if one of the frames was shifted by even one pixel with respect to the other, the resulting value of frame similarity may be large, thus implying that the frames are dissimilar.

In order to alleviate this problem, we can represent video frames by means of global image features. A global feature characterizes certain statistical property of an image, such overall brightness or color composition, and is therefore more robust to local changes and noise. Research in the fields of image processing and retrieval has produced several global features, such as color histogram, texture, or color moments. All these features could be considered intra-image characteristics, as they represent an

image by its internal properties. In addition, if an image is part of a video sequence, and is preceded and succeeded by other images, inter-image features may be determined. Most notably, one can estimate apparent motion between consecutive images. Any of such features (or a combination thereof) may be used in measuring frame similarity. The use of specific features depends on the application, and may be determined by the computational complexity of the feature calculation. In the domain of video retrieval, this aspect becomes particularly important, as the number of individual images is very large. In this work we aim to detect repeated footage in a live video broadcast, and so must be able to compute frame features in real time. Therefore, we focus on the computationally simple features, such as color histogram and color moments.

Color histogram representation divides the spectrum of values of primary color components (usually the range of 0 to 255) into a number of smaller ranges, called *histogram buckets*. The color histogram is the set of values representing the number of pixels in the video frame whose color components belong to the corresponding buckets. The size of the set of values is determined by the number of buckets. The color moments representation may be thought of as an approximation of the color histogram. The three primary moments: mean, standard deviation and skew are computed for each of the three color components: red, green and blue. Each set of moments describes a statistical distribution uniquely determined by their values, and approximates the histogram for the respective color component. Due to the nature of approximation, the color moment representation introduces certain error, as depicted in Figure 102. On the other hand, color moments require only nine numbers to represent a video frame, which is considerably more compact than the color histogram representation. Considering this fact, we chose color moments as frame representation.

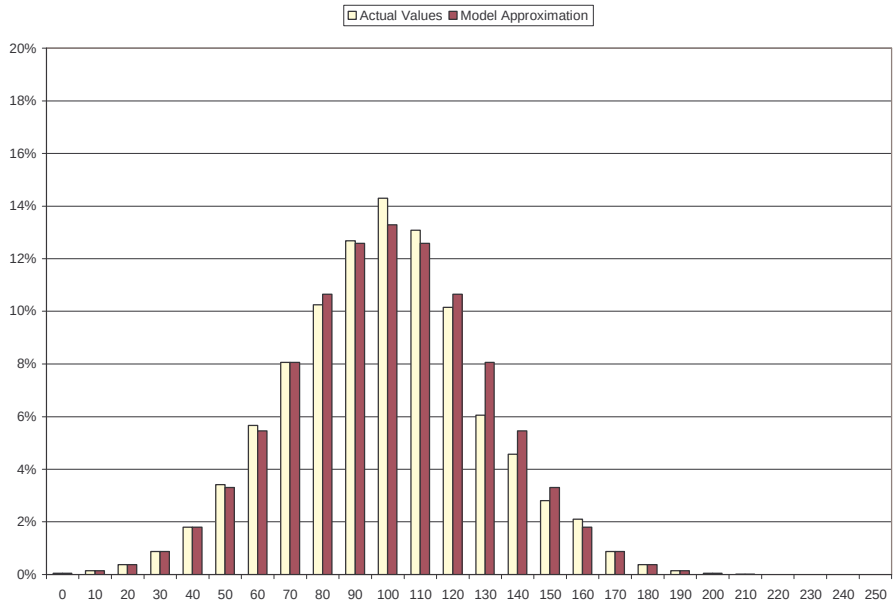

Figure 102 An actual histogram and its approximation by a normal distribution with mean = 10 and standard deviation = 30.

Based on the choice of color moments for frame representation, we introduce the following definition of frame similarity.

Definition 7: Similarity of two frames f_a and f_b is inversely proportional the average color moment difference between them.

$$FrmSim_{\beta}(f_a, f_b) = 1 - FrameAvgMomentDiff(f_a, f_b) \tag{18}$$

Definition 8: Let V_x be the vector of color moments of frame f_x:

$$V_x = \langle M_x(r), M_x(g), M_x(b), S_x(r), S_x(g), S_x(b), K_x(r), K_x(g), K_x(b) \rangle$$

The average color moment difference is given by the following equation:

$$FrameAvgMomentDiff(f_a, f_b) = \frac{1}{9} L_p(V_a, V_b) \tag{19}$$

where L_p is the chosen distance metric given by

$$L_p(V_a, V_b) = \left[\sum_{i=1}^{9} \left(\left| V_{a,i} - V_{b,i} \right| \right)^p \right]^{\frac{1}{p}} \tag{20}$$

Global frame feature representations have one potential shortcoming. Since they represent the entire image by a small set of numbers, they certainly are incapable of capturing every detail of the image content. Consequently, it may be possible for pairs of images whose content is quite different to have very similar or even identical feature representations. Specifically, the three primary color moments approximate the histograms of the red, green and blue components of a video frame by a Gaussian curve of a certain mean, standard deviation, and skew. One can easily imagine one frame whose color composition is such that the color histograms are bimodal, for example, as well as another one whose color histograms are Gaussian, as shown in Figure 103. Such two frames would be represented by the same values of mean, standard deviation, and skew. This issue may have significant impact of comparison of individual frames. However, its impact is greatly reduced when entire sequences of frames are considered (see section 3.2.3).

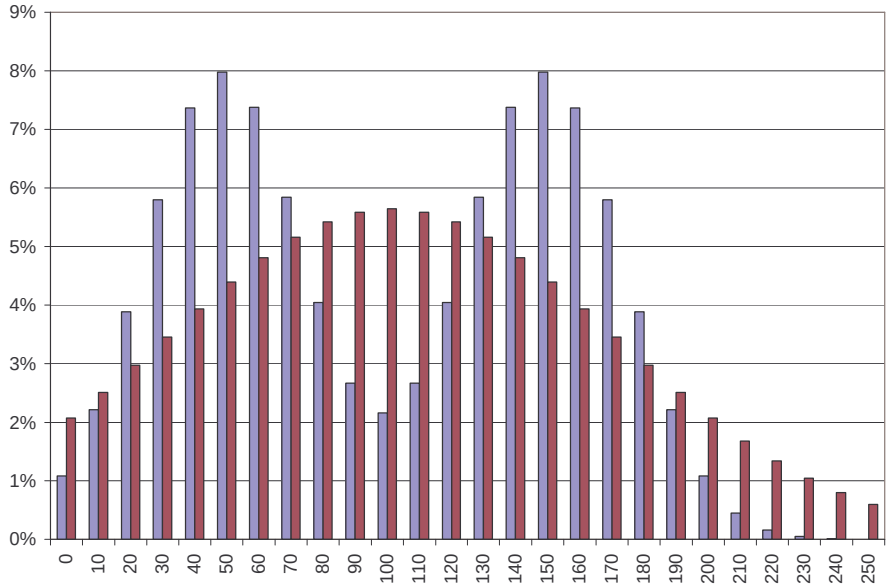

Figure 103 Different color histograms with identical mean and standard deviation

While Definition 7 provides more resilience to local distortions of the video frames, it is still quite sensitive to global changes, such as increase or decrease in brightness. If the two frames compared are identical, except that one has increased brightness, the differences $V_{a,i} - V_{b,i}$ will be directly related to the difference in brightness. Consequently, such two frames may be declared dissimilar.

Both of the problems presented above may be alleviated if the two frames compared are considered in the broader context of the video clips they belong to. In the next section we introduce frame similarity metrics which utilize this contextual information.

Contextual Frame Similarity Metrics

Global differences in brightness or color intensity may be eliminated by normalizing color moment values by their averages in the entire clip. Consider two clips S_a and S_b which are identical in everything except brightness. Let us assume that the brightness of S_b has been increased by certain amount Δb. As a consequence, the color components of all pixels in S_b will increase by that amount, and so will $M_b(t,c)$ for all colors.

$$I_b(t,x,y,c) = I_a(t,x,y,c) + \Delta b \tag{21}$$

$$M_b(t,c) = \frac{1}{N}\sum_{xy} I_b(t,x,y,c) = \frac{1}{N}\sum_{xy}\left(I_a(t,x,y,c) + \Delta b\right) = M_a(t,c) + \Delta b \tag{22}$$

If we now calculate the average values of color mean for the whole clip S_b, we obtain:

$$\overline{M}_b(c) = \frac{1}{N}\sum_{t=1}^{N_b} M_b(t,c) = \frac{1}{N}\sum_{t=1}^{N_b}\left(M_a(t,c) + \Delta b\right) = \overline{M}_a(c) + \Delta b \tag{23}$$

Consequently, we can normalize the values of color mean of S_a and S_b by subtracting their mean values for the entire clips, as follows:

$$\tilde{M}_a(t,c) = M_a(t,c) - \overline{M}_a(c) \quad \textit{and} \quad \tilde{M}_b(t,c) = M_b(t,c) - \overline{M}_b(c) \tag{24}$$

Finally, by substituting (22) and (23) into (24) we obtain:

$$\tilde{M}_b(t,c) = M_a(t,c) + \Delta b - \overline{M}_a(c) - \Delta b = M_a(t,c) - \overline{M}_a(c) = \tilde{M}_a(t,c) \quad (25)$$

If we now compute the *FrameAvgMomentDiff(S_a, S_b)* on the normalized values of color moments we will obtain zero, and the two clips will be considered identical. Therefore, we introduce the following definition.

Definition 9: Similarity of two frames f_a and f_b is inversely proportional to the normalized average color moment difference between them, as follows:

$$FrmSim_{\chi}(f_a, f_b) = 1 - NormFrameAvgMomentDiff(f_a, f_b), \quad (26)$$

where

$$NormFrameAvgMomentDiff(f_a, f_b) = \frac{1}{9}\left[\sum_{i=1}^{9}\left(\left|V_{a,i} - V_{b,i}\right|\right)^p\right]^{\frac{1}{p}} \quad (27)$$

Moment normalization is very effective in eliminating global changes in brightness. However, its direct use is limited by the requirement that precise clip boundaries be known. Consider two identical clips *A* and *A'*. The average color moment values are also identical for both clips. However, if clip *A'* was extended to include additional frames from another clip *B*, the average moment values calculated for this new clip *A'B* may be substantially different. As a result, the normalized moments for frames in *A* and *A'* will also differ, and the similarity between the two clips may not be established.

Another way of utilizing contextual information to rectify the effects of global changes is to consider inter frame differences in color moments. For every frame in a video clip, we can calculate the first order difference (discrete approximation of the first derivative) of all color moments as follows.

$$dV_{x,i}(t) = V_{x,i}(t) - V_{x,i}(t-1) \quad (28)$$

In this case, the values representing each frame depend only on the current frame and the frame directly preceding it. This eliminates the requirement on precise clip boundaries. Also, if the values of color moments change by a constant, the first order differences remain the same, which allows us to formulate the following frame similarity measure.

Definition 10: Similarity of two frames f_a and f_b is inversely proportional to the average difference in color moment derivative between them, as follows:

$$FrmSim_{\delta}(f_a, f_b) = 1 - NormFrameAvgMomentDerivDiff(f_a, f_b) \tag{29}$$

where

$$FrameAvgMomentDerivDiff(f_a, f_b) = \frac{1}{9}\left[\sum_{i=1}^{9}\left(\left|dV_{a,i} - dV_{b,i}\right|\right)^p\right]^{\frac{1}{p}} \tag{30}$$

For all of the frame similarity metrics presented in this section, we can define the corresponding concept of *frame match*.

Definition 11: We say that frame f_a matches frame f_b according to the similarity metric α if and only if $FrmSim_{\alpha}(f_a, f_b)$ greater or equal to a predefined threshold.

$$f_a \overset{\alpha}{=} f_b \quad \Leftrightarrow \quad FrmSim_{\alpha}(f_a, f_b) \geq FrmMatchThresh_{\alpha} \tag{31}$$

The frame similarity metrics introduced in this section allow us to compare individual video frames. In practice, we are mainly interested in comparing entire sequences of frames, i.e. video clips. In the next section, we introduce the notion of video clip similarity, and develop a number of corresponding metrics.

3.2.3 Video Clip Similarity Metrics

In order to identify repetitions of video clips in a large video sequence, we need a way of comparing pairs of video clips so as to establish their similarity. In a straightforward approach, one can compare all pairs of clips in the video sequence, and declare repetitions for the pairs which are sufficiently similar, that is, matching. For this purpose, we develop a number of clip similarity measures, which are discussed in this section. We begin by generally classifying clip comparison techniques. One can distinguish two approaches:

1. ***Frame-by-frame comparison***, which consists in comparing corresponding frames in the two clips involved, and aggregating the frame similarity over the length of the clips.
2. ***Clip-wide representation comparison***, which relies on features describing a video clip as a whole. Such features may be directly compared to establish clip similarity.

Although both techniques may be applied to complete clip comparison, only the first is suitable for partial clip comparison. Given two clips S_A and S_B which share a sequence of frames S_{AB}, one can expect that the frames in S_{AB} to be identical. Conversely, a clip-wide features derived from S_A is likely to be substantially different from those derived from S_B. Since in this work we are interested in detecting partial clip repetition, we focus our attention on frame-by-frame comparison methods. First, we discuss the problem of comparing complete clips, and present suitable similarity metrics. Later, we address the issues of partial clip repetition and show how the metrics may be adapted to deal with them.

Average Moment Difference

Using the color moment representation of video frames one can derive a clip similarity measure from the total difference in moments between corresponding frames.

Definition 12: Consider two video clips S_a and S_b of length N. Clip similarity between them may be defined as the average absolute moment difference between their corresponding frames:

$$ClipSim_{\alpha}(S_a, S_b) = 1 - ClipMomentDiff(S_a, S_b), \tag{32}$$

where

$$ClipMomentDiff(S_a, S_b) = \frac{1}{N}\sum_{i=1}^{N} FrameAvgMomentDiff\left(f_a^i, f_b^i\right), \tag{33}$$

and FrameAvgMomentDiff(f_a, f_b) is given by (19).

Definition of the *FrameAvgMomentDiff* allows for different distance metrics L_p to be used. The value of *p* in the distance metric determines the relative impact of large differences over small differences. Given clip S_a and its repetition S_a', a small number of frames may differ substantially due to a number of factors, such as noise. The impact of such isolated large differences on the average difference is more pronounced for distance metrics with larger values of *p*. Therefore, in order to reduce that impact we chose distance metric L_1.

Two clips S_a and S_b match according to the average moment difference metric if $ClipSim_{\alpha}(S_a, S_b)$ is greater or equal to a threshold, i.e.

$$S_a \overset{\alpha}{\approx} S_b \quad \Leftrightarrow \quad ClipSim_{\alpha}(S_a, S_b) \geq ClipMatchThresh_{\alpha}, \tag{34}$$

Matching Frame Percentage

The average moment difference metric described above makes use of the continuous values of the average moment difference between frames. These values may be quantized into two ranges by introducing a *frame match threshold*. As a result, for every pair of corresponding frames, one can determine whether they match by

comparing the moment difference between them to the threshold. The percentage of matching frames may then be used to measure clip similarity, as follows.

Definition 13: Given two video clips S_a and S_b of length N, the similarity between them is measured by the ratio of the number of matching frames to the total number of frames N, as defined by the following equation.

$$ClipSim_{\beta}(S_a, S_b) = \frac{1}{N}\sum_{i=1}^{N} frameMatch(f_a^i, f_b^i), \tag{35}$$

where

$$frameMatch(f_a^i, f_b^i) = \begin{cases} 1 & if \quad f_a^i \overset{\beta}{\approx} f_b^i \\ 0 & Otherwise \end{cases} \tag{36}$$

Analogically, two clips S_a and S_b match according to the matching frame percentage metric if $ClipSim_{\beta}(S_a, S_b)$ is greater or equal to a threshold, i.e.

$$S_a \overset{\beta}{\approx} S_b \quad \Leftrightarrow \quad ClipSim_{\beta}(S_a, S_b) \geq ClipMatchThresh_{\beta}, \tag{37}$$

Partial Similarity

Video clip similarity metrics presented so far dealt only with clips of precisely equal length. In practice, the clips we need to compare almost always differ in length. Even if an entire video clip was reused in a video sequence, it is likely that its difference was altered slightly due to different transitions used to separate it from other clips. In addition, if automatic temporal segmentation was used to obtain clips to compare, the imprecision of its methods most certainly led to a change in the clip lengths. And finally, news broadcast stations often reuse video footage in part rather than in entirety. Consequently, the metrics introduced earlier could rarely be applied

directly. Instead, another measure of clip similarity is needed, which will allow us to recognize partial similarity between video sequences.

Consider two video clips *A* and *B*, for which no temporal structure is available. Either of the clips may consist of any number of shots, of which some may match shots in the other clip. Theoretically, virtually any pattern of repetitions between the two clips may exist. However, in practical situations, the pair of clips for which partial similarity must be evaluated contains only one matching subsequence, which is shown in Figure 104.

Figure 104 Example of partial repetition

Since the length of the subsequence or its location in clips *A* and *B* is unknown, one must consider every possible pair of subsequences in order to find the best match, which leads to the following definition.

Definition 14: Given two clips S_a and S_b of lengths N_a and N_b respectively, partial similarity between the two clips is equal to the maximal value of similarity between any pair of frame subsequences $S_a^{g,g+n}$ and $S_b^{k,k+n}$ taken from S_a and S_b, respectively, such that $|S_a^{g,g+n}| = |S_b^{k,k+n}| = n$ and n exceeds the significant length threshold L:

$$PartialClipSim_\alpha = \max\left(ClipSim_\alpha\left(S_a^{g,g+n}, S_b^{k,k+n}\right)\right),$$

where

$$1 \le g \le N_a - n, \quad 1 \le k \le N_b - n, \quad and \quad n \ge L$$

The value of the significant length threshold must be chosen in such a way that it reduces the possibility of accidental similarity between different clips. As discussed earlier in this section, depending on the frame similarity metric used, accidental

similarity between individual frames is possible. In practice, the likelihood of accidental similarity decreases very fast with the increase in the number of consecutive frames compared, and for sufficiently long sequence of frames is virtually zero. For the remainder of this work we will use 30 frames as the value of the significant length threshold.

3.2.4 Summary

In this section, we proposed a number of similarity metrics, which allow us to assess similarity between individual video frames, as well as video sequences. We developed clip similarity definitions for video sequences of exactly the same length, and extended them to be applicable to partially similar sequences. We also introduced the notion of frame and clip match, which we defined as similarity exceeding a predetermined threshold. In the next section we present an overview of repeated video sequence detection methods which utilize these concepts.

3.3 Overview of Methods

In the previous section, we discussed video clip similarity metrics, which provide the foundation for repeated clip detection by means of comparing pairs of clips. Now we will present an overview of the algorithmic methods of repeated clip detection.

In section 3.1.1, we defined the task of repeated clip detection as identifying all subsequences of a query sequence Q in a video sequence S. We also introduced three additional requirements that may be placed on the detection task: segmentation of the query sequence, segmentation of the source sequence, and detection of complete shots only. Depending on which of these requirements are present, different detection techniques may be used.

Let us consider the detection task whose objective is to detect all complete repetitions of the query sequence Q. Since sequence Q may be repeated starting at any frame of the source sequence S, the detection algorithm must perform a clip comparison

between *Q* and a sequence of the same length starting at every frame in *S*. This straightforward detection method becomes considerably more complicated, if one is interested in detecting occurrences of all subsequences of the query *Q*. If no structure is imposed on *Q*, it is possible that any subsequence *q* of *Q* may be repeated starting at any frame of *S*. Consequently, identifying all such repeated occurrences requires comparing every subsequence of *Q* to a subsequence of *S* of the same length starting at every frame of *S*. This is the principle of the brute force repeated sequence detection algorithm, which is discussed in section 3.4.

This daunting task may be somewhat simplified by introducing temporal structure in the query and source sequences. If both *Q* and *S* are divided into distinct shots and only complete repetitions are to be detected, then the detection method must simply compare every shot in *Q* with every shot in *S*. This vastly reduces the number of necessary comparisons. In addition, one can filter out pairs of shots of different lengths without performing a detailed comparison. If one is interested in detection of partially repeated shots, the problem becomes more complicated, but is still considerably easier than detection of partial repetition of the unsegmented query clip *Q*. The detailed discussion of the repeated shot detection method will be presented in section 3.5.

Although the presence of temporal segmentation reduces the number of comparisons necessary for repeated sequence detection, the straightforward shot detection techniques are still too slow to perform exhaustive repetition detection in real-time on commodity hardware, which is needed for story tracking in live news broadcasts. Therefore, we need a technique to further restrict the group of shots that must be directly compared. To accomplish this, we reach for a heuristic method based on quantization and hashing of color moments, which is described in section 3.6.

3.4 Repeated Clip Detection Algorithm

The repeated clip detection algorithm presented in this section is the most direct implementation of video sequence comparison. It is the most time consuming of all the methods presented in this chapter, but may be applied in the absence of temporal segmentation results. The algorithm may be considered for both single and exhaustive detection, and identifies both complete and partial repetitions.

Let us consider the single detection task in which only occurrences of the whole query sequence Q are to be detected. For this task, the algorithm must compute the video clip similarity metric between Q and a sequence of length $|Q|$ beginning at every frame of the source sequence S. If the two match according to the metric, a repeat is reported, and the algorithm moves on to the first frame after the matching clip. If not, the method tries the next frame in the sequence S. The pseudo-code version of the algorithm is shown in Figure 105.

Complete Sequence Detection Algorithm

```
Function ExhaustivelyCompareSequence(Clip q, Clip s)
    ForEach frame in s
        ss = Clip(frame, frame + q.Length)
        similarity[frame] = ClipSim(q, ss)
    EndFor
    return similarity
EndFunction

Function DetectCompleteSequence(Clip q, Clip s)
    similarity = ExhaustivelyCompareSequence(q, s)
    For i = 0 to similarity.Length
        matches[i] = similarity[i] <= matchThreshod
    EndFor
    return matches
EndFunction
```

Figure 105 Complete sequence detection algorithm

The remaining two detection tasks: single with partial repetitions and exhaustive are conceptually quite different. The former aims to establish the similarity between the query sequence Q and the source sequence S, whereas the latter focuses on identifying all repeated subsequences of Q in S. Both tasks, however, may be

accomplished using very similar computational methods. The definition of partial similarity requires that all subsequences of *Q* be considered and compared to all subsequences of *S*, by computing a value of a clip similarity metric. After all such comparisons have been performed, the maximum value of similarity metric is chosen as the measure of similarity of *Q* and *S*. The exhaustive detection task also calls for comparison of all subsequences of *Q* to all subsequences of *S*. Once this is done, matching pairs are identified as those whose similarity exceeds a match threshold. Hence, large portions of the respective algorithms are identical. Both algorithms are presented in Figure 106 and Figure 107.

Partial Sequence Detection Algorithm

```
Function DetectPartialSequence(Clip q, Clip s)
    maxSim = 0
    For i = 0 to q.Length
        For j = i to q.Length
            qs = Clip(i, j)
            similarity = ExhaustivelyCompareSequence(qs, s)
            tmpMaxSim = Math.Max(similarity)
            maxSim = Math.Max(tmpMaxSim, maxSim)
        EndFor
    EndFor
    return maxSim
EndFunction
```

Figure 106 Partial sequence detection algorithm

Exhaustive Subsequence Detection Algorithm

```
Function DetectAllSubsequences(Clip q, Clip s)
    maxSim = 0
    For i = 0 to q.Length
        For j = i to q.Length
            qs = Clip(i, j)
            similarity = ExhaustivelyCompareSequence(qs, s)
            For k = 0 to similarity.Length
                matches[i, j, k] = similarity[k] <= matchThreshod
            EndFor
        EndFor
    EndFor
    return matches
EndFunction
```

Figure 107 Exhaustive Subsequence detection algorithm

Time Complexity

For each of the three algorithms, the central operation is the computation of the value of similarity between two sequences of frames. Every such operation consists in turn of a series of frame similarity evaluations. We consider the latter to be the atomic operation of unit cost, and estimate computation complexity by the number of frame comparisons required. The complete sequence detection algorithm requires |*S*| comparisons of sequences of length |*Q*|, so its computational cost is $O(|S| \cdot |Q|)$. The other two algorithms perform the same number of sequence comparisons for every subsequence of *Q*. Since the number of such subsequences is given by $\frac{1}{2}|Q| \cdot (|Q| - 1)$, and the average subsequence length is $\frac{1}{2}|Q|$, the total computational cost is $O(|S| \cdot |Q|^3)$. If one considers *Q* equal to *S*, and denotes their length in frames as *n*, which for 24-hour video sequence exceeds 2.5 million, then the computational cost of partial or exhaustive detection becomes an overwhelming $O(n^4)$.

Given the complexity of the exhaustive sequence detection algorithm, it becomes obvious that performing this task on live news video broadcasts in real-time is not possible on commodity hardware. Consequently, we need to develop different techniques that require less computation.

3.5 Repeated Shot Detection Algorithm

3.5.1 Overview

In the previous section, we described a repeated sequence detection algorithm which does not utilize temporal segmentation results. Estimation of computational complexity of the algorithm showed that the problem quickly becomes intractable with the increase in the length of the source and query sequences. In this section, we demonstrate that using temporal segmentation, we can vastly reduce the overall

number of sequence comparisons that need to be performed. We present an algorithm which compares sequences only at shot boundaries, and detects both complete and partial shot repetitions.

3.5.2 Algorithm

In the presence of temporal segmentation of both the query sequence *Q* and the source sequence *S*, repeated clip detection consists in comparing all shots in *Q* to all shots in *S*.

Repeated Shot Detection Algorithm

```
Function RepeatedShotDetection(Shot[] queryShots, Shot[] sourceShots)
    For qi = 0 to queryShots.Length
        For si = 0 to sourceShots.Length
            repetitions[qi, si] = ShotSimilarity(queryShots[qi], sourceShots[si]) >= matchThreshod
        EndFor
    EndFor
    return repetitions
EndFunction
```

Figure 108 Repeated shot detection algorithm

In the algorithm above, implementation of the *ShotSimilarity* function depends on whether complete repetition is required.

Complete Repetition Detection

If only complete shot repetition is allowed, then calculation of shot similarity reduces to a direct implementation of the clip similarity metric, i.e. comparing the two shots frame by frame. In addition, even before the direct comparison is performed, one must reject pairs of clips of different length.

Complete Shot Similarity Algorithm

```
Function CompleteShotSimilarity(Shot q, Shot s)
    if (q.Length != s.Length) return false
    For i = 0 to q.Length
        clipMomentDiff+= FrameAvgMomentDiff(q[i], s[i])
    EndFor
    shotSimilarity = 1 – clipMomentDiff / q.Length
    return shotSimilarity
EndFunction
```

Figure 109 Complete shot similarity algorithm

Partial Repetition Detection

If partial repetition is allowed, then calculating shot similarity becomes somewhat more complicated. According to the definition of partial similarity one must determine the most similar pair of subsequences of the two shots. Thanks to the nature of video shots, however, this task is relatively simple. In section 3.1 a video shot is defined as a sequence of successive video frames taken from a single camera working continuously. It turns out that this definition places an important restriction on partial repetition between shots. In fact, there are only two distinct ways in which any shot *A* may be partially repeated, which is conceptually illustrated in Figure 110.

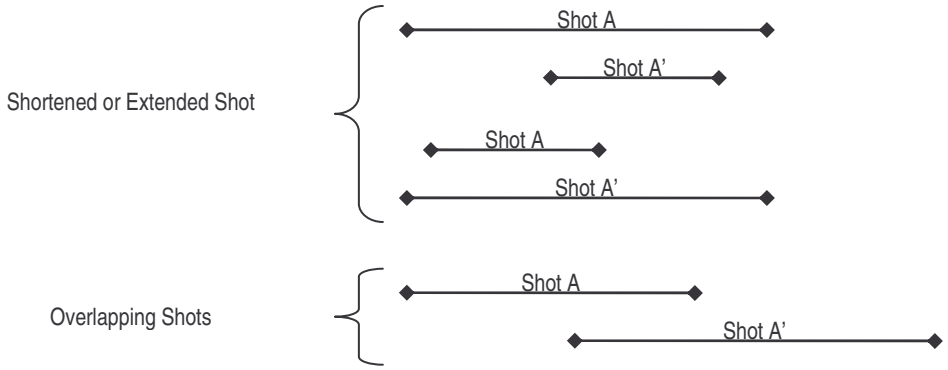

Figure 110 Two distinct ways of partial shot repetition

According to the shot definition, every frame *f* in the shot is immediately preceded by the frame taken from the same camera directly before *f*, and succeeded by the frame taken from the same camera immediately after *f*. Consider two shots $A = \langle f_A^1, \ldots, f_A^{N_A} \rangle$ *and* $B = \langle f_B^1, \ldots, f_B^{N_B} \rangle$, and assume $f_A^i = f_B^j$ for some *i* and *j*, i.e. the two frames were taken from the same camera the exact same time. If the two frames are not the last ones in their respective sequences (i.e. $i < N_A$ and $j < N_B$), then we also have $f_A^{i+1} = f_B^{j+1}$. Analogically, if neither of the frames is first in its sequence ($i > 0$ and $j > 0$), then $f_A^{i-1} = f_B^{j-1}$. Hence, we can conclude that if two shots

A and *B* contain an identical sequence of frames *S*, this sequence must begin one of the shots and end one of the shots.

This reasoning implies that one can determine all potentially similar subsequences of the two shots by sliding one of the shots over the other from left to right and taking all pairs of overlapping sequences. One starts at the point where there is only one frame of overlap, then the overlap increases, reaches the maximum value equal to the length of the shorter of the two clips, and then decreases to a single frame on the other side. This is depicted conceptually in Figure 111. In practice, in order to avoid accidental similarity between very short subsequences, one usually requires that the overlapping subsequence was at least *L* in length.

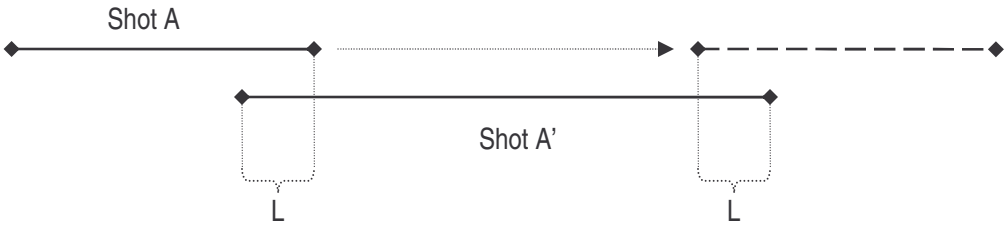

Figure 111 Partial shot similarity computation diagram

For every such overlapping subsequence, the value of clip similarity is calculated, and the maximal value is taken to represent the partial similarity of the two shots. The partial shot similarity algorithm in pseudo-code is shown below.

Partial Shot Similarity Algorithm

```
Function PartialShotSimilarity(Shot q, Shot s)
    sigLenThresh = 30

    leftShiftBound = - (s.Length - sigLenThresh)
    rightShiftBound = q.Length – sigLenThresh

    bestMatch = 0.0
    bestMatchShift = 0
    bestMatchLength = 0

    For shift = leftShiftBound to rightShiftBound
        shift1 = shift > 0 ? shift : 0
        shift2 = shift < 0 ? shift : 0
        overlapLength = Math.Max(q.Length - shift1, s.Length - shift2)
```

```
            curMatch = CompleteShotSimilarity(
                        Shot(start1 + shift1, start1 + shift1 + overlapLength),
                        Shot(start2 + shift2, start2 + shift2 + overlapLength))
            If (curMatch > bestMatch)
                  bestMatch = curMatch
                  bestMatchShift = shift
                  bestMatchLength = overlapLength
            EndIf
      EndFor

      return bestMatch, bestMatchShift, bestMatchLength
EndFunction
```

Time Complexity

If we assume that the complexity of the shot similarity algorithm (whether complete or partial) is constant, then the overall complexity of the repeated shot detection algorithm would be given by $O(c_Q \cdot c_S)$, where c_Q and c_S are the number of shots in the query sequence *Q* and the source sequence *S*, respectively. In practice, the cost of calculating shot similarity is proportional to the length of the compared shots, and depends on whether partial repetition is allowed. The complexity of calculating the complete shot similarity is *O(p)*, where *p* is the shot length, because a single frame comparison must be performed for every frame of the shots in question. In addition, if partial similarity is allowed, then complete sequence similarity must be calculated at multiple offsets, whose number is proportional to the combined length of the shots involved. Hence, the partial shot similarity algorithm complexity is $O(p^2)$.

Thus, the overall complexity of partially repeated shot detection is $O(c_Q \cdot c_S \cdot p^2)$. If we consider sequences Q and S to be equal, and denote the number of shots they contain as *c*, we obtain $O(c^2 \cdot p^2)$. While the number of shots *c* is proportional to the number of frames *n* in the sequence, in a typical video news broadcast it is two orders of magnitude smaller ($c << n$). Furthermore, the average shot length *p* is independent of *n* ($p = n/c$). Consequently, the repeated shot detection method is computationally less intensive than the repeated sequence detection algorithm discussed in the previous section. Therefore, we conclude that the availability of the

results of temporal segmentation greatly simplifies the task of repeated footage detection.

3.5.3 Impact of Segmentation Errors

Thus far in this section, we assumed availability of a perfect temporal segmentation method. In practice, no such automated method exists (see Chapter 2). Therefore, any repeated shot detection technique must deal with imprecision and errors introduced by imperfect temporal segmentation. In this section, we discuss the impact of imperfect segmentation on repeated shot detection.

Automated temporal segmentation methods introduce two types of imperfections listed below.

1. ***Imprecise shot boundaries***. Gradual shot transitions often do not have very well defined boundaries, which makes it difficult to determine where a transition starts and a shot ends, and vice versa. As a consequence, even completely repeated shots may differ in length by a few frames.
2. ***Transition detection errors***. At times, temporal segmentation fails to detect a transition entirely, or reports a transition were none occurred. As a result, some automatically reported shots may, in fact, be composed of more than one actual shot, while some actual shots may be split into two or more reported shots.

The impact of imprecise shot boundaries is limited and relatively easy to deal with due to two factors. First, since transitions between shots are usually quite short, the error in detecting their boundary must by definition be small, and typically involves only few frames. Second, transition boundary errors occur only for gradual transitions, such as fades or dissolves. During these transitions, the frames close to the transition boundaries are indeed very similar to the neighboring frames in the surrounding shots. Consequently, if one compares shots whose boundaries have not been precisely detected, one may inadvertently compare a small number of frames belonging to an adjacent transition, but these frames are quite similar to the frames in

the nearby shot. Therefore, the similarity calculated between such shots should not differ much from the actual similarity between the shots.

Hence, the algorithm for partial shot similarity may be applied without changes. In fact, it helps overcome some of the segmentation errors. In contrast, the complete shot similarity method must be slightly adjusted. Due to imprecise boundaries, even two identical shots may differ in length by a few frames. In order to allow for such shots to be recognized as identical, one must relax the requirement on equal length. This can be done by introducing a threshold length ΔL by which two clips are allowed to differ. In addition, in order to calculate the maximum value of shot similarity, one can slide the two shots with respect to each other in a manner similar to the one used in the partial shot similarity method within the bounds of $+/- \Delta L$.

The other type of imprecision in temporal segmentation comes from errors in transition detection, and presents a significant challenge for repeated shot detection methods. Two types of errors may occur: a) omission of a true transition, and b) introduction of a false transition. The former leads to two or more shots that are combined into a single sequence (under-segmentation), while the latter results in a single shot being split into two or more separate sequences (over-segmentation). These two types have different influence on repeated shot detection, and will be discussed separately.

First, consider the problem of over-segmentation. Assume shot *A* was erroneously divided into two sequences SA_1 and SA_2 which now appear to be separate shots. Assume also that shot *A'* is a repetition of *A*, as depicted in Figure 112.

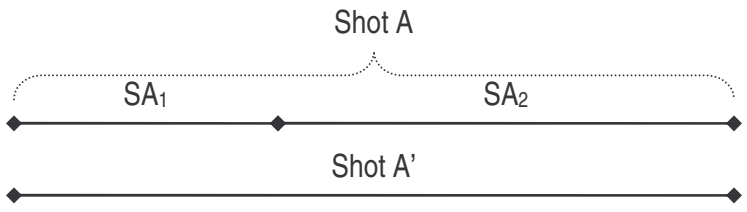

Figure 112 Example of over-segmentation with a single falsely detected transition

If only complete repetition is allowed, then the similarity between shots *A* and *A'* will likely never be detected due to the difference in length between SA_1 and *A'*, as well as SA_2 and *A'*. Therefore under-segmentation is a considerable obstacle for complete repetition detection. In contrast, our partial shot similarity algorithm will report similarity between both pairs SA_1 and *A'*, and SA_2 and *A'*, provided that the length of SA_1 and SA_2 exceeds the significant length threshold. Hence, under-segmentation does not pose a problem for the detection of partial shot repetition.

In contrast, under-segmentation proves to be more challenging. Consider four shots *A*, *B*, *B'*, and *C*, as presented in Figure 113, where *B'* is a repetition of *B*. Assume that transitions between shots *A* and *B*, as well as *B'* and *C*, were not detected by temporal segmentation, and as a result the four shots appear as two sequences *AB* and *B'C*.

Figure 113 Example of under-segmentation with a single undetected transition

Let us examine the behavior of the complete repetition detection algorithm in this situation. The algorithm first tests if the shot lengths are the same with the margin of error of ΔL. Clearly, if the lengths of *A* and *C* differ by more than ΔL, this test will fail. Moreover, even if the lengths of the combined shots were similar enough, the complete shot similarity would likely be fairly low. If the similarity algorithm is applied, it will compare corresponding frames in shots *AB* and *B'C*. It is unlikely that either frames in *A* are similar to frames in *B'*, or frames in *B* are similar to frame in *C*. As a result, the difference between *AB* and *B'C* will be large, and no repetition will be detected. Thus, the complete repetition detection algorithm performs very poorly in the presence of under-segmentation.

In the same situation, the detection of partial shot repetition works very well. If one examines the diagram in Figure 113, one observes that shot *B* closes the combined

shot *AB*, and shot *B'* opens the combined shot *B'C*. If shot *B'* is a complete repetition of *B*, the two shots match the type of subsequence that the partial shot similarity method is designed to recognize. Hence, the repetition will be detected. On the contrary, if shot *B* ends with a sequence of frames absent from *B'*, or if *B'* begins with a sequence of frames absent from *B*, the similarity between them will not be recognized. Analogically, a repetition between shots for which both surrounding transitions have been missed will not be detected (see Figure 114).

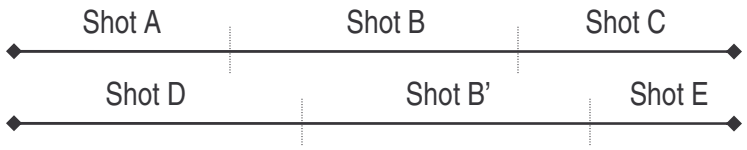

Figure 114 Example of under-segmentation with two undetected transitions

In summary, imperfect temporal segmentation makes repeated shot detection more difficult. The severity of its impact differs for different types of errors. Imprecision in shot boundaries does not pose a problem, while over-segmentation and under-segmentation are more difficult to deal with. In general, the methods of partial repetition detection can handle these types of errors much better than complete repetition techniques.

3.5.4 Summary

In this section, we presented our repeated shot detection algorithm, a repeated sequence detection technique which takes advantage of the results of temporal segmentation. We demonstrated that the availability of shot boundaries in both query and source sequences allows us to significantly reduce the number of sequence comparisons necessary to detect repetitions in the source video. Because sequence comparison is a costly operation, this reduction constitutes a substantial decrease in execution time. In addition, the division of video sequences into shots greatly simplifies detection of partially similar sequences.

Because automatic temporal segmentation methods are not perfect, we also considered the impact of imperfections in the shot detection data on the working of the algorithm. We examined the influence of different types of inaccuracies in temporal detection. We showed that our method can deal well with imprecision in the gradual transition boundaries. The errors of over-segmentation and under-segmentation proved to be more challenging, and can be handled reasonably well only by the partial repetition detection techniques. We concluded that our method performs better in the context of over-segmentation, rather than under-segmentation.

This allows us to draw a conclusion regarding our temporal segmentation methods. The performance of these methods is measured by two factors: recall and precision. Recall represents the percentage of true shot transitions detected by temporal segmentation, and may be thought of as a measure of under-segmentation. Analogically, precision stands for the percentage of true transitions among all transitions reported by the segmentation, and may be regarded as a measure of over-segmentation. Considering the greater difficulty in dealing with under-segmentation, we are more interested in the recall than we are in precision. Therefore, when we perform temporal segmentation for the purposes of repeated sequence detection, we can fine tune the parameters governing the process so as to increase recall at the expense of precision.

3.6 Hashing and Filtering Algorithm

3.6.1 Overview

In the previous section, we demonstrated that the availability of shot boundaries allows us to significantly reduce the number of direct sequence comparisons, and consequently greatly improve execution time. While the algorithms presented in the previous section are fast enough for the task of interactive repetition detection, they

still are not fast enough for the exhaustive repetition detection on live news broadcasts. If we examine one hour of typical television broadcasting, we will see that it consists of hundreds of video clips. Some may be as long as one minute in length, while others are as short as one second. Thus, if we temporally segment 24 hours worth of video, we might find as many as 10,000 video clips. If we attempt a brute force video comparison approach to detect repeated video clips, we would need to perform approximately 10,000 video comparisons for every new clip. To complete this task in real time, we would need to make these comparisons in less than 10 seconds. This is not viable using current commodity computers.

Since the direct shot comparison is a costly operation, we seek to further reduce the number of comparisons needed for detection of repeated shots. In this section, we introduce a filtering method which allows us to effectively eliminate shots whose similarity to a given shot can be ruled out. Naturally, this approach will only be advantageous if the filtering can be performed much faster than the direct shot comparison, which means the method must be very fast. In order to develop the filtering method, we reach for a heuristic technique based on quantization and hashing of color moments.

In the remainder of this section, we describe this technique in detail. First, we discuss color moment quantization and its impact on video sequence similarity. Later, we examine hashing as the means to reduce spatial requirements of our algorithm. We follow, by presenting the complete shot repetition algorithm using filtering, and we close by presenting time and spatial constraints of the algorithm.

3.6.2 Color Moment Quantization

In the effort to limit the number of direct shot comparisons, we want to find ways of quickly ruling out similarity between pairs of shots. Two shots can certainly be deemed dissimilar if they contain no (or very few) similar frames. Therefore, if we can devise a method of quickly determining for a given shot, which other shots have a substantial number of similar frames, then we can later compare this shot only to

those shots. The trick is to do it without having to compare pairs of frames one by one.

To accomplish this goal, we employ a quantization technique. We observe that the color moment representation of every frame may be considered a point in a 9-dimensional space. The space is finite in size, because every color moment has finite range of values from 0 to 255. If we divide the color moment space into a number of disjoint 9-dimensional hyper-cubes, every frame can be uniquely assigned to one of them. Hence, given any frame of the video sequence we can quickly determine which other frames belong to the same hyper-cube. Such division of color moment space can be performed by quantizing the values of all nine moments. Each hyper-cube is then defined by the unique set of quantized color moment values. Also, a pair of frames belongs to the same hyper-cube if the quantized values of all their color moments are equal. Therefore, we can formally introduce the following definition.

Definition 15: Two frames f_a and f_b are q-matching if the values of their corresponding color moments quantized with step qs are equal:

$$f_a \overset{q}{\approx} f_b \Leftrightarrow \forall i : qV_{a,i} = qV_{b,i}, \text{ where } qV_{x,i} = \left\lfloor \frac{V_{x,i}}{qs} \right\rfloor \tag{38}$$

We note that this definition of frame match is substantially different from the ones introduced in section 3.2.2. Due to quantization of the color moment values, two frames which differ by more than the quantization step will always be considered non-matching. On the other hand, frames whose moments differ by less than the quantization step may or may not be regarded as matching depending on their values. Let us consider the average moment difference similarity metric, for instance, and assume we have two pairs of frames whose color moment values are identical, except for mean of the red component M(r). For frames f_a, f_b, and f_c let the values of the mean be: $M_a(r) = 4.1$, $M_b(r) = 3.9$, $M_c(r) = 5.9$. Clearly, frames f_a and f_b are more

similar than f_a and f_c. However, if we use quantization step $qs = 2.0$, we get $M_a(r) = 2.0$, $M_b(r) = 1.0$, $M_c(r) = 2.0$, which means that frames f_a and f_c q-match, while f_a and f_b do not. This artifact of quantization is not desirable and may cause a small number of pairs of very similar frames to be deemed dissimilar.

One could attempt to derive a video sequence similarity metric from the definition of frame q-match analogically to $ClipSim_\beta$ in Definition 13. However, due to the artifact of quantization described above, such a metric would be inadequate because the value of sequence similarity it produces depends not only on the difference in color moments between the corresponding frames, but also on the relative distance of those moments from the quantization thresholds.

Therefore, another video sequence similarity metric must be developed. We will devise such a metric from the total number of q-matching pairs of frames, regardless of their temporal ordering. In order to do it, we first make the following observation. Color moments of individual frames in a video sequence tend to change gradually over time. As a result, the moment values usually differ slightly from frame to frame. Consider two shots A and B, such that $A \overset{\beta}{\approx} B$. If there exists a pair of frames f_A^i *and* f_B^i, such that $f_A^i \overset{\beta}{\approx} f_B^i$, but $f_A^i \overset{q}{\neq} f_B^i$, we can reasonably assume that there exists a frame f^j in shot B such that $f_A^i \overset{q}{\approx} f_B^j$, and conversely there exists a frame f^k in shot A such that $f_A^k \overset{q}{\approx} f_B^j$. These assumptions are heuristic in nature, and we are not guaranteed that for every frame in shot A we can find a q-matching frame in shot B, and vice versa. However, the probability of finding such q-matching pairs of frames increases with the value of the quantization step. This relationship is depicted in Figure 115, which shows the number of q-matching pairs of frames as a function of the quantization step for a sample video sequence and two of its repetitions.

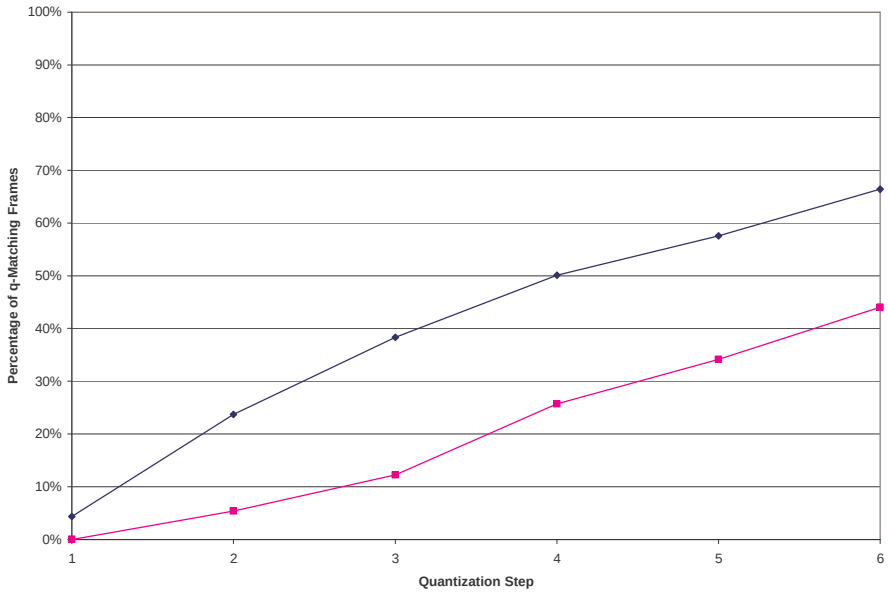

Figure 115 Dependency of the sequence q-similarity on the quantization step

The discussion above allows us to formulate the following definition of heuristic video sequence similarity, called *q-similarity*.

Definition 16: Given two video sequences S_a and S_b of length N, the q-similarity between them is equal to the total number of q-matching pairs of frames, as defined by the following equation.

$$ClipSim_Q(S_a, S_b) = \sum_{i=1}^{N} qFrameMatch(f_a^i, f_b^i), \tag{39}$$

where

$$qFrameMatch(f_a^i, f_b^i) = \begin{cases} 1 & if \quad f_a^i \overset{q}{\approx} f_b^i \\ 0 & Otherwise \end{cases} \tag{40}$$

Clearly, q-similarity between any given pair of video sequences depends on the quantization step chosen. If the value of the quantization step is too large, quite

dissimilar frames will q-match, and consequently very different sequences may score a high q-similarity. Conversely, if the value is too small, many similar frames will not q-match, and so some repeated video sequences may not be very q-similar. In fact, the value of quantization step should be closely related to the value of the frame match threshold chosen for the clip similarity metric ($ClipSim_{\beta}$).

The goal of quantizing color moments for video frames is to allow us to filter out dissimilar pairs of clips without directly comparing them frame by frame. So far in this section, we discussed quantization and q-similarity in abstract terms. Now we turn to the method which will allow us to perform filtering based on quantized color moments. First, we note that given the definition of q-similarity between video sequences, we can establish a threshold on q-similarity, and for every pair of video sequences declare them dissimilar (or non-q-matching) if their q-similarity is below the threshold. We could take every pair of video sequences and calculate their q-similarity. This, however, would not be any faster than directly calculating similarity between all pairs of sequences.

Instead, the advantage of using color moment quantization to assign frames to unique hyper-cubes lies in the fact that this operation may be performed once only for each frame. Namely, one can store a list of frames for every hyper-cube in memory. Later, when detecting repetitions of a certain shot *A*, one can check the list of frames assigned to every hyper-cube that contains at least one frame from shot *A*. Combining frames from all these lists, one can quickly establish which shots in the long video sequence q-match shot *A*. The actual frame-by-frame comparison may then be performed only on shot *A* and all q-matching shots found. Assuming that the number of q-matching shots is substantially smaller than the total number of shots in the sequence, we conclude that the number of direct comparisons decreases substantially, and so does the execution time. Detailed analysis of the achievable speed-up is provided in section 3.7.

It is clear that in order to accelerate repeated shot detection by using this filtering mechanism, we need to store lists of frames assigned to hyper-cubes of the quantized

space. However, if we consider quantization of color moments with step 2.0, we can calculate that in the 9-dimensional space, the number of hyper-cubes is given as:

$$H = \left(\tfrac{1}{2} * 256\right)^9 = 2^{63} \approx 9.2 * 10^{18}$$

Obviously, storing a list of frames for every such cube would exceed memory capacity of any currently available computer. However, the number of frames in a 24-hour video sequence recorded at 30 fps is only on the order of 10^6. Therefore, a vast majority of the hyper-cubes are empty, and no data needs to be stored for them. Hence, one can achieve a high degree of storage requirement compression if memory is allocated only for the non-empty hyper-cubes. Clearly, many dynamic memory structures could satisfy this requirement. Not all of them, however, provide direct constant-time access to data, which is crucial in making the filtering algorithm fast. One data structure which satisfies both criteria is hash table, which proves ideal for our purposes. The constant-time access to the appropriate hyper-cube for any given frame is available by means of computing a hash value of the quantized moments of this frame. This hash value serves as an index to a flat array. The size of this array is proportional to the total number of frames in the sequence, and is therefore many times smaller than the total number of hyper-cubes in the color moment space. The performance of hash table depends on the properties of the hashing function, which will be discussed in the next section.

3.6.3 Color Moment Hashing

A number of techniques have been developed for hashing images and video clips to improve the performance of image and video retrieval [Chi01a, Oos01, Sab95]. Image hashing has also been used in image watermarking applications to uniquely identify images for authentication purposes [Kal01]. Given the space/time constraints on exhaustive repeated sequence detection, the constant time insertion/lookup provided by image hashing makes it an ideal solution for our application.

The practical performance of hash table algorithms depends on the properties of the hash function used to generate array indexes for the elements of a given data set. Indeed, the very premise of constant-time insertion and lookup is dependent on the distribution of the values generated by the function. For a given data set, in which every element is represented by a certain key, an optimal hash function generates a unique index for every unique key. In the past, much research has been devoted to developing optimal hash functions for a variety of applications, and numerous classical solutions exist. A good exposition of the basic hashing algorithms may be found in [Knu97], which also provides additional references for more advanced hashing techniques.

In this work, the data set is the sequence of video frames, each represented by the vector of quantized color moment values, which serves as the key. We need to devise a hash function which will map this set of keys into a set of indexes into an array. In order to fully realize the potential savings in execution time, we need a good hash function. However, we are not concerned with devising an optimal function, but rather a computationally simple function of satisfying quality. We aim to create a function which distributes the unique vectors of quantized color moments approximately evenly between the array entries.

Pua [Pua02] devised a hashing function which relied on a string representation of the frame color moments. He first quantized the nine color moment values with step 10 to integers in the range of 0 to 25. Quantized moment values for each video frame are then converted to their string representation and concatenated to create fixed length character strings. The hash value for the given frame is calculated by multiplying the ASCII values of all characters in the string.

The conversion of integer moment values to strings in Pua's method is unnecessary. The operation is rather expensive computationally, and does not improve hash value distribution. Therefore, in this work we employed a more straightforward approach. In order to obtain a hash value for a given frame, we calculate the product of the quantized moment values. To avoid multiplying by zero, we add one to every

quantized value before multiplication. In addition, we multiply the value of each moment by its position in the moment vector, which introduces a difference in hash value between moment vectors with the same set of values in different order. The complete hash function is given by the following formula:

$$hv = \prod_{i=1}^{9} i \cdot (qV_i + 1) \bmod hashTableSize \tag{41}$$

Experiments is section 3.7.2 demonstrate that the hashing function posses the desired distribution properties for a typical sequence of video frames.

3.6.4 Algorithm

Our hashing and filtering repeated sequence detection algorithm consists of four major components (see Figure 116): moment quantization, video frame hashing, video sequence filtering, and video sequence comparison. In the first step we quantize color moments for every frame of the source video sequence. We then insert all frames into the hash table according to the hash value calculated for their quantized color moments. In the sequence filtering phase, we identify all potentially similar video sequences. Finally, in the last step we directly compare pairs of sequences to determine if they are truly matching. The remainder of this section describes the components of our algorithm and data structures in more detail.

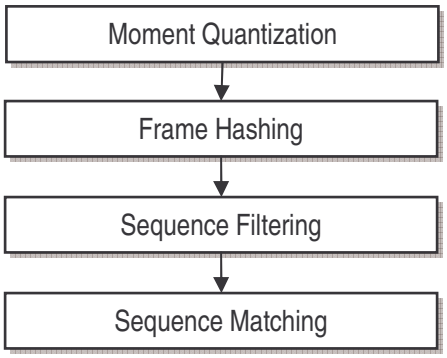

Figure 116 Hashing and filtering repeated sequence detection algorithm diagram

Color Moment Quantization

In this phase of the algorithm, we uniformly quantize color moment values for every frame in the video sequence. As discussed in section 3.6.2, the quantization step was chosen in relation to the frame match threshold, and determined as follows:

$$qs = 2 \cdot frameMatchThreshold \tag{42}$$

Frame Hashing

In order to perform hashing on a video frame level, we reuse the nine color moments which we calculated prior to video segmentation. For every frame, we compute the hash value according to (41), and store the frame number in the hash table slot according to the hash value. Since we perform repeated sequence detection on a 24-hour news broadcast at 30 frames per second, there are 1440 * 1800 = 2,592,000 video frames in our source video. Hence, the storage requirement on our hash table of video frame moments is substantial.

Our hash table must provide space for at least 2.5 million frames. In addition, we need to handle multiple frames with identical hash values. Such duplicate frames may be the result of either moment quantization or frame hashing. In the first case, we encounter truly duplicated frames. In the second, a collision occurs due to non-unique mapping of the hash function. Regardless of the cause, the hash table must be able to accommodate multiple entries with the same hash value. This can be done using one of the two methods: separate chaining, multiple probing. While separate chaining is often the preferred technique, it proved less desirable in this case due to the overhead of dynamic memory handling. First, the amount of data stored for every frame is very small, i.e. only the frame number. Therefore, the cost of using pointers in the dynamically linked lists required for separate chaining effectively doubles the size of the structure. More importantly, the linked list entries must be allocated dynamically, which incurs the overhead of memory management. Taking these factors into account, we decided to use linear probing to handle duplicate frames. In

order to alleviate problems with congestion, we allocated approximately three times as many entries as there are frames in the source sequence: 10,000,001.

Experiments show (see section 3.7.2) that the average number of frames in a single hash table bucket is 6.7, while the largest bucket contains over 1000 frames. Clearly, this distribution is non-uniform, which introduces time jitter in the amount of time needed to process individual video frames, but our overall performance is still well within real time constraints.

Video Sequence Filtering

In this phase of the repeated shot detection process, we determine the pairs of shots with a potential for a match, and eliminate all others. To this end, we examine all shots in the query sequence one at a time, and find all shots in the source sequence whose q-similarity to the given query shot exceeds a predetermined threshold. We maintain a shot q-similarity counter array with a value for every shot in the source sequence, as well as a Boolean array of encountered frames with an entry for every frame of the query shot. For every frame in the query shot that has not been encountered, we obtain the list of q-matching frames from the hash table. This is done by first getting the list of all frames in the same bucket in the hash table, and removing those whose quantized moments are different (i.e. the results of collisions). We count the number of q-similar frames which came from the query shot, as well as calculate the number of frames that came from each source shot. The entry of every source shot in the q-similarity counter array is increased by the minimum of the number of q-similar frames of the query shots and the number of q-similar frames of the source shot. In addition, every q-similar frame of the query shot is marked as encountered. This procedure guarantees that every q-similar frame is counted only once. It also ensures that very still source shots do not receive artificially high q-similarity scores, which could otherwise occur if many of their frames (all q-similar to each other) shared a hyper-cube with at least one frame of the query shot. Once this process is performed for every frame of the query shot, the q-similarity counter

array contains q-similarity scores for every shot in the source sequence. At this point, all source shots whose q-similarity is less than the predetermined threshold are removed.

In the next stage, if only complete shot repetitions are allowed, then we further narrow down the list of q-similar source shots by imposing a restriction on the length of the shot. We eliminate all source shots whose length does not match that of the query shot. To account for imprecision in shot boundary detection, we allow a difference in length of ΔL. On the other hand, if partial repetition is permitted, no further restrictions are imposed.

Once the list of source shots has been filtered, we proceed to directly compare the remaining source shots to the query shot.

Video Sequence Matching

After the completion of the previous phase, the list of pairs of shots is reduced to only the q-matching pairs (Q^{gh}, S^{jk}). In this final step, the sequences Q^{gh} and S^{jk} are compared using the complete or partial shot similarity algorithms described in section 3.5.2. If their similarity exceeds the predefined match threshold, then the shots are reported as matching, and S^{jk} is considered a repetition Q^{gh}.

3.6.5 Time and Space Considerations

The video hashing and filtering method we proposed offers very significant reduction of processing time at the expense of increasing the algorithm's spatial requirements. In order to achieve desirable performance, the size of the hash table must exceed the number of frames by a factor of three. In our experiments, we operated on a 24-hour video sequence.

Increase in sequence length requires an increase in memory size. In order to search longer sequences for repetitions, we must impose a search window of certain length.

This window can slide along the video sequence. We can remove frames that are outside of this window to accommodate new incoming frames.

3.7 Experiments and Discussion

In this section, we discuss the results of several experiments we conducted to establish certain parameters of our algorithms, critically examine our assumptions, as well as evaluate the performance of our methods. We begin by testing our assumption that corresponding frames of repeated video sequences have very similar color moments. We then analyze the behavior of different clip similarity metrics introduced in section 3.2.3 in the context of video news broadcasts. Later, we analyze the properties of quantization and hashing using both similarity metrics. We end with the discussion of the execution time reduction due to the filtering process, and evaluation of the overall performance of our detection algorithm.

All experiments were done using the 24-hour video sequence recorded on a typical day of the CNN News broadcast. In order to minimize the influence of on-screen captions, all video frame features were calculated using only the top 75% of the pixels in the frame.

3.7.1 Clip Similarity Metrics

In the previous section, we considered repeated sequence detection based on raw color moments. This approach is consistent with the assumption that video sequences taken from the same source do not undergo significant global changes or degradation, which we adopted in the introduction to this chapter. In this section, we test this assumption, and consider other similarity metrics and their application in our techniques.

Metric Analysis

First, we compare the color moments of several repetitions of the same video clip. We found that in the majority of repetitions, the moment values for the corresponding frames in the matching clips are very similar. Occasionally, however, when the clip is used in a different news program, the overall brightness differs somewhat, which is illustrated in Figure 117. This issue was discussed in section 3.2.2, where we also proposed to use a different similarity metric to deal with this problem. We showed that for every frame in a given clip, one can normalize each color moment by subtracting the average value of that moment calculated over the entire clip. Similarity between two clips may then be measured by the difference in normalized moments between their corresponding frames. We showed that this similarity measure is invariant to global changes in brightness.

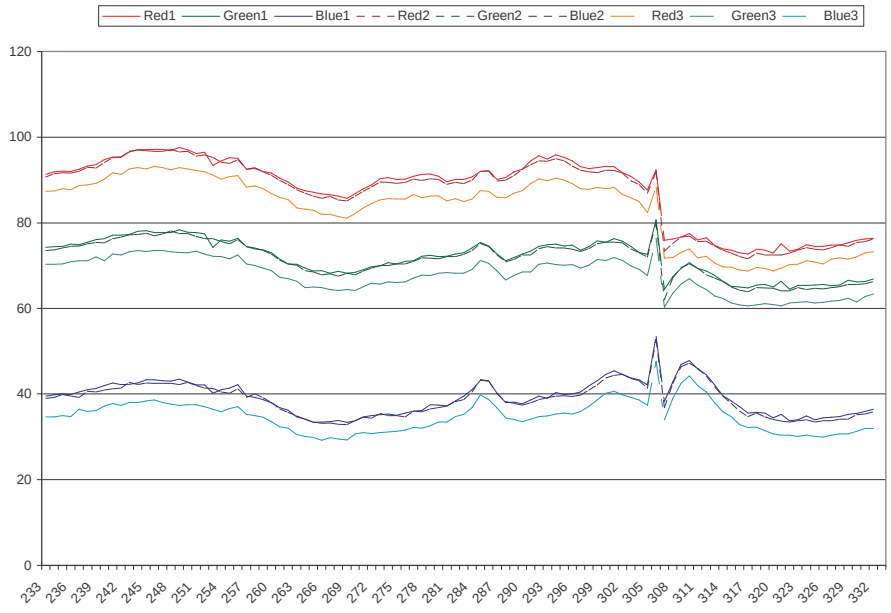

Figure 117 Mean of red, green, and blue components for 100 frames of a repeated clip

Direct application of this similarity scheme requires information about precise clip boundaries, so that average moment values may be computed correctly. Otherwise, normalization by an average value computed over a clip boundary would most likely lead to undesirable results. For example, consider adjacent clip pairs *A* and *B*, and *A'*

and *C*, in which *A'* is a repetition of clip *A*. If the average moment values were computed from the start of clip *A* to the end of clip *B*, and from the start of clip *A'* to the end of clip *C*, they will be influenced by the color moments of frames in clips *B* and *C*, which may be very different. Consequently, the normalized moment values for the frames in clips *A* and *A'* would be dissimilar. Therefore, in the absence of precise clip boundaries, the use of normalized moments is limited.

Moreover, if one is interested in detecting partial clip repetitions, moment normalization is not straightforward, even when clip boundaries are known. Consider clip *A* and its shorter version *A'*. If the sequence of frames of clip *A* that are absent from *A'* is considerably different in color composition than the frames present in *A'*, then the average moment values computed for *A* and *A'* will differ substantially. As a result, normalized moments for frames in clips *A* and *A'* will be quite dissimilar.

Therefore, in order to apply moment normalization in the context of partial clip repetition with uncertain clip boundaries, one needs to alter the metric's definition. Instead of computing average moment values over entire clips, one can impose a window of size $2w + 1$ around every frame in the video sequence, and determine the average moment values for that window. The normalized moments for a given frame can be obtained by dividing its moment values by the average moments for the surrounding window. This approach is illustrated in Figure 118, which shows normalized mean values for color components of three repetitions of the same clip. One can easily see that the shift in moment values due to the change in brightness has been eliminated.

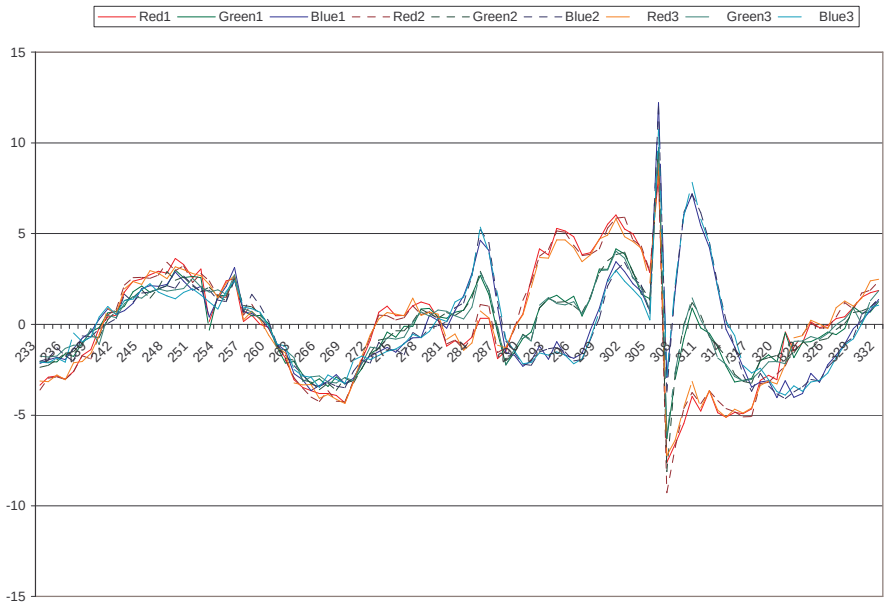

Figure 118 Normalized mean of red, green, and blue components for 100 frames of a repeated clip

This outcome indicates that the normalized moment difference metric – even in its modified form – may be suitable for detection of repeated video sequences. In order to compare the practical performance of the similarity metrics based on raw moment values and their normalized counterparts, we conducted another experiment in which repetitions of the same query clip were detected using the brute force method. The clip chosen for the experiment is 10 seconds long, and is shown in full or in part 5 times in the 24-hour video sequence. The start times of all occurrences are shown in Figure 119. With the exception of the 4th repetition, all matching sequences are at least 5 seconds long. The 4th sequence – although also 5 seconds in length – matches the query clip on only the last 2 seconds. The last sequence is slightly darker than the query clip. In addition, the query sequence is shown several times as picture-in-picture in a smaller window, often partially occluded by a view of an anchor person or an interviewee. Such occurrences are difficult to classify as repetitions of the query sequence, but they cannot be dismissed as false positives either.

In order to detect these repetitions, the brute force algorithm was run on the source sequence. The algorithm was configured to report all similar sequences for which the

number of matching frames was equal to at least 30 with the frame match threshold equal to 2.0. With these parameters, the raw-moment algorithm returned a reasonable number of matches. In contrast, the normalized moment difference metric indicated over 50,000 matching sequences. We adjusted the frame match threshold 0.5, and increased the minimal number of matching frames to 120. The results of repetition detection with these parameters are depicted in Figure 119 and Figure 120. While the raw moment similarity reported all five occurrences of the query sequence, the last detection must be regarded as accidental. The sequence at 22:54:36 actually matches the query clip on over 8 seconds (240 frames), but only 36 frames match according to the metric. The normalized moment metric indicated seven repetitions, three of which are the picture-in-picture versions of the query clip. The other four are the main screen repetitions of the query sequence. Finally, the normalized moment metric misses the 4th repeated sequence, which matches the query clip on only 2 seconds.

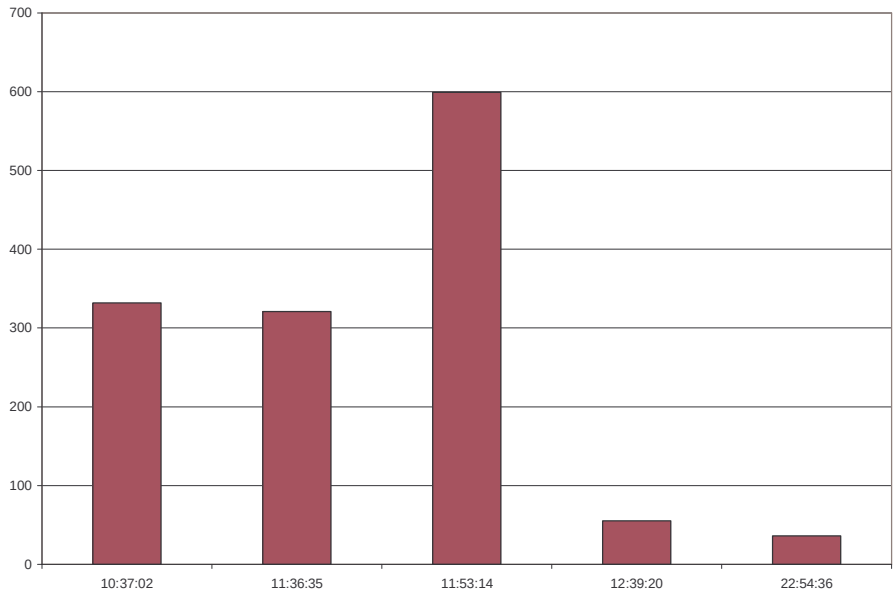

Figure 119 Sample clip repetitions detected using raw moment difference metric

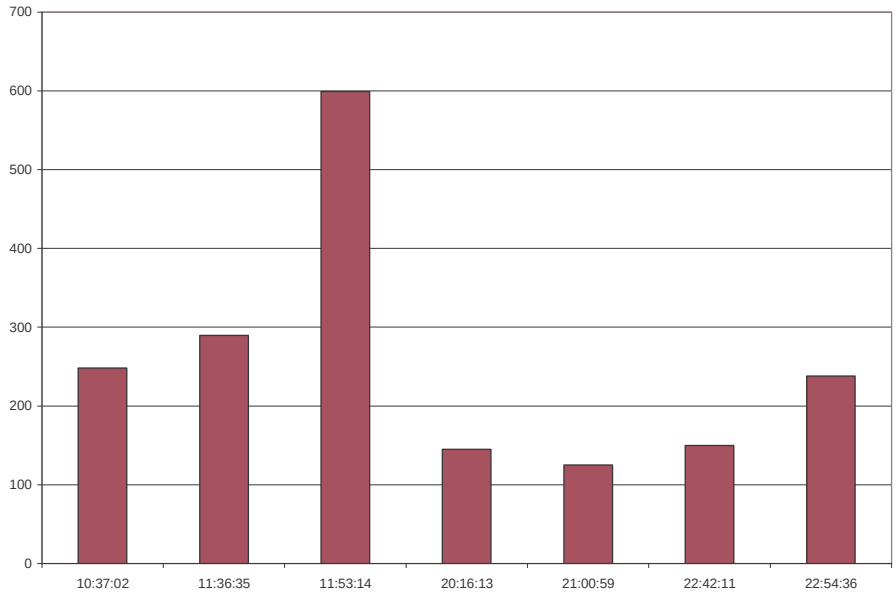

Figure 120 Sample clip repetitions detected using normalized moment difference metric

From these observations, we can derive a number of conclusions regarding the applicability of the two metrics for repeated sequence detection in news broadcasts:

1. Repetitions of video sequences with somewhat changed global characteristics, such as brightness, do occur. If the change is substantial, the raw moment metric will not detect a match. On the other hand, normalized moment metric handles these types of repetitions very well.

2. Normalized moment difference metric performs poorly if the overlap between the original clip and its repetition is small. Since the moment averages are calculated regardless of clip boundaries, their values may be determined by frames belonging to more than one clip. Specifically, if a frame *f* lies within *w* frames of a clip boundary, then its normalized moments will be affected by frames in the current clip, as well as the previous or next clip. Consequently, frames which belong to repeated clips but are close to clip boundaries may have different normalized moment values. However, assuming that *w* is small in comparison to the clip length, we conclude that for the majority of the

frames in the clip, the normalized moment values will be calculated correctly, and will yield the same values for repetitions of the same clip.

3. Because color composition of consecutive frames in a single clip usually changes in a gradual manner, the normalized moment values tend to be relatively small for a vast majority of frames in a video sequence. If the content of a sequence of frames does not change much over time, then the average color moments for this sequence are almost equal to the moment values of individual frames. As a result, the normalized moments are very close to zero, as shown in Figure 121. This in turn leads to difficulties in discerning between different clips, especially if they are relatively still. Even if the original moment values for frames in such clips may have been substantially different, their normalized values may be very small and quite similar.

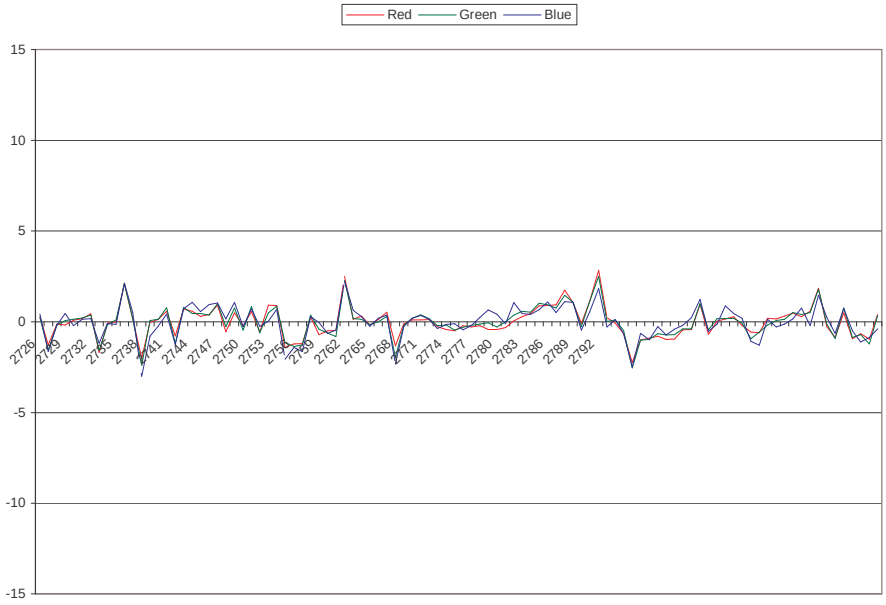

Figure 121 Normalized mean for red, green and blue components of a very still clip

4. The normalized method indicates repetitions in the picture-in-picture mode. This intriguing outcome is the result of the fact that the remainder of the

screen is still, and thus the differences in color moments are mainly due to changes in the clip.

Summarizing these conclusions, we note that the normalized moment difference metric outperforms the raw moment version in detection of somewhat altered sequences. However, it is not suitable for detection of still clips.

In section 3.2.2, we suggested another similarity metric invariant to global color changes in video, which relied on comparing first order differences of color moments. In this method, one calculates the difference in color moments between pairs of consecutive frames. The advantage of this approach is that it calculates correct values for all frames except those directly after a clip boundary. However, the color moments tend to differ very little between consecutive frames. As a consequence, the first order differences used in this method are even smaller than the normalized moment values, and thus the metric is even less useful for still clip comparison.

In view of the conclusions presented above, we can divide video sequences into two groups: static and dynamic. Static sequences contain little motion or other changes, and are characterized primarily by their color composition. On the contrary, the content of the dynamic clips changes considerably, and thus the clips are better represented by the change in color composition. The raw moment similarity metric is designed to measure color composition, and so is better suited for static sequences, whereas the normalized metric emphasizes similarity in the color composition change patterns.

Video news broadcasts certainly contain both dynamic and static clips. The former often appear in commercial sequences, as well as certain news content shots, while the latter are typical of anchor persons, interviews and studio settings. Therefore it is important that the similarity metric used was able to deal effectively with both types of shots. As we discussed above, the similarity metrics relying on changes in frame composition are ill-suited for still shots. Consequently, we chose to use the raw

moment metric for repeated sequence detection. In the next section, we examine the practical performance of this metric in detecting repeated sequences in video news.

Raw Moment Metric Performance on Video News Broadcasts

In order to assess the accuracy of the raw moment similarity metric we conducted a set of shot repetition detection experiments using a group of sample video clips. We divide the pool of clips into three categories: commercials, studio clips, and news content clips. Each of these three groups is different in nature and has a distinct pattern of repetitions in a typical news broadcast. Consequently, we performed separate tests for each of the groups. This section will describe in detail the results obtained for news content shots, as the accuracy of their detection is essential for effective story tracking. The evaluation for the other two groups will be discussed briefly.

For the detection of repeated news content shots a set of 50 representative shots was chosen from a 24-hour broadcast of CNN News. The shots varied in length from 60 frames to over 1000 frames, as well as differed in the amount of motion. Each of the selected shots was compared to every shot in the entire broadcast using the repeated shot detection algorithm described in section 3.5. The ground truth for the experiment was established manually, and the performance was measured using recall and precision, defined as follows.

$$recall_A = R_A = \frac{number\ of\ correctly\ reported\ shots\ matching\ A}{number\ of\ all\ shots\ matching\ A\ in\ the\ broadcast} \tag{43}$$

$$precision_A = P_A = \frac{number\ of\ correctly\ reported\ shots\ matching\ A}{number\ of\ all\ shots\ reported\ as\ matching\ A} \tag{44}$$

The two measures were combined into a single value of utility according to (9) with $\alpha = 0.5$. Since the accuracy of the metric depends on three parameters: significant length threshold, frame match threshold, and clip match threshold, we used different values of each parameter to establish the optimal settings. The results of the

experiment for different parameter values are presented in Table 8. One can see that the repetition detection is the most accurate if the metric is used with frame similarity threshold of 3.0, and clip similarity threshold set to 0.5. The values of recall and precision as the function of frame similarity threshold are depicted in Figure 122.

		Frame Similarity Threshold				
Clip Similarity Threshold		5.0	4.0	3.0	2.0	1.0
	0.25	48.50%	64.30%	79.28%	88.64%	78.44%
	0.50	60.28%	72.06%	**90.22%**	87.47%	77.47%
	0.75	66.53%	75.97%	89.41%	87.23%	74.60%

Table 8 Raw moment metric performance measured by the utility value

Subsequently, we ran the same set of tests using the significant length threshold L of 60 frames. The results we obtained were equivalent to the ones presented in Table 8. Therefore we chose $L = 30$ frames as the optimal significant length threshold, because it allows for detection of partially repeated shots of smaller length.

The outcome of these experiments can be summarized, in the following way. Two shots are considered partially matching if they overlap on at least 30 frames, out of which at least 50% differ in moments by at most 3.0.

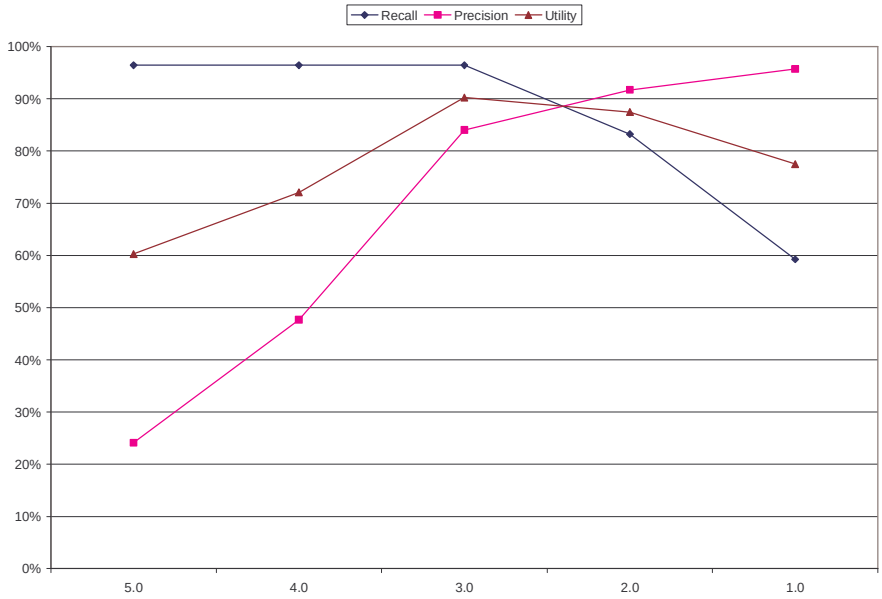

Figure 122 Raw moment metric performance for the clip match threshold of 0.5 as a function of the frame match threshold

A similar experiment was performed on a set of typical commercial shots. Because commercials are always shown on the full screen, the color moment values are not influenced by on screen captions. Consequently, moment values differ only slightly between repetitions and the detection attains high recall (over 90%) even with very small values of the frame match threshold. Precision values were found to be equally high (over 90%) for frame match threshold of 1.0 through 3.0. Therefore, the raw moment metric can be effectively used to detect repetitions of commercials with the same parameter values as for news content sequences.

While commercials and news content clips exhibit certain similarities in the repetition pattern, studio sequences are characterized by very different properties. Studio shots are always filmed live, and are virtually never repeated in the news broadcast. The exceptions to this rule are a few late night news shows which are usually re-broadcast a few times throughout the night, but these are of no particular interest for story tracking purposes. As a result, studio and anchor clips do not repeat in the strict sense provided in Definition 4. On the other hand, different clips of this type shown during the same news program are often visually very similar. Clearly, one can always

identify the original query clip in the source sequence, because the moment values of its frames are identical to the ones in the query clip. Thus recall of 100% is always attained. One can also obtain 100% precision by lowering the frame match threshold to zero, thus eliminating all other sequences except the original query clip. In practice, the optimal frame match threshold value determined above leads to substantially lower precision. This issue represents a weakness of the similarity metric and has an adverse effect on story tracking, which will be discussed in Chapter 4. In that chapter we will also propose a method which utilizes the locality of such pseudo-repetitions to alleviate this problem to a certain extent.

Summarizing, we can say that the raw moment metric can be used to accurately match both commercial sequences and news content clips. The best accuracy was achieved with significant length threshold of 30 frames, frame match threshold of 3.0, and clip match threshold of 0.5. The same parameters used for anchor and studio clip matching result in low precision, which is cause by visual similarity of this type of sequences.

3.7.2 Quantization and Hashing

In order to verify our assumptions regarding the properties of quantized color moments, we examined the statistical distribution of video frames among the hyper-cubes of the quantized moment space. The experiment we conducted demonstrates that the frame distribution is approximately uniform, with a vast majority of the cubes containing a very small number of frames. In the experiment we used the quantization step determined by (42) as twice the value of the frame match threshold. The best value of this threshold was established experimentally in section 3.7.1 as 3.0. Thus, the value of 6.0 was selected for the quantization step. We performed quantization of a 24-hour video sequence, which contained a total of 2,589,052 frames. Prior to quantization, all monochrome frames were removed from the sequence, leaving 2,563,990 frames to be quantized. Figure 123 depicts a histogram of frame distribution between the hyper-cubes of the quantized space. The histogram

shows that over 50% of all cubes contained only single video frame, while close to 90% held 7 frames or less. The average number of frame per hyper-cube was 6.65 with the standard deviation of 47.39.

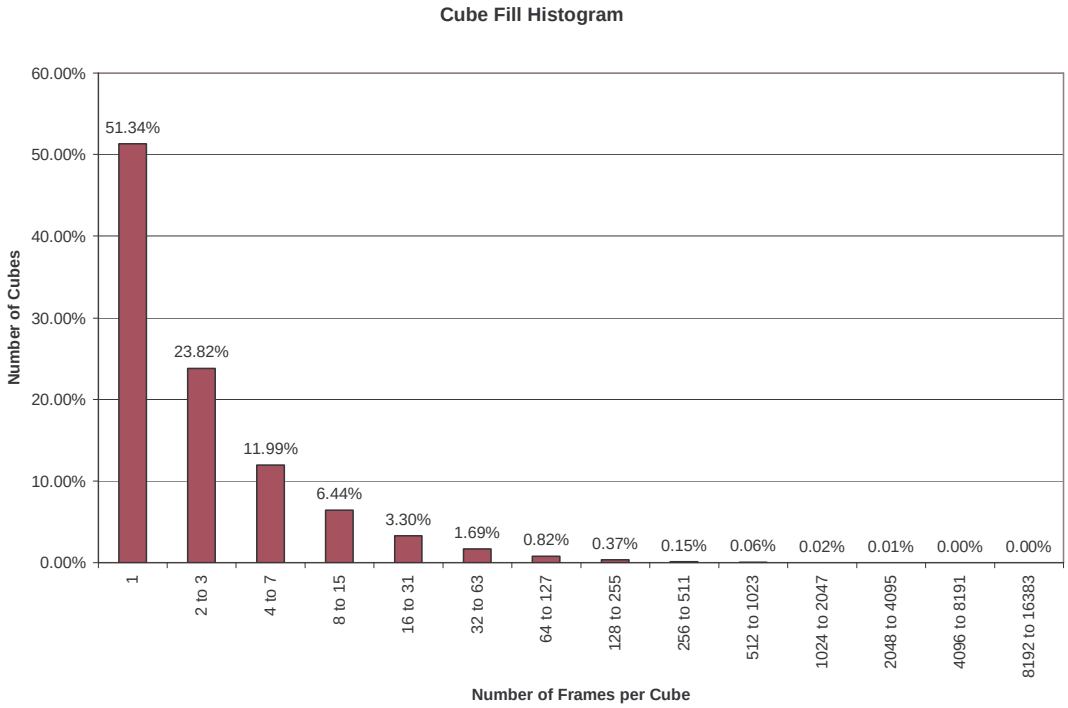

Figure 123 Histogram of frame distribution between hyper-cubes of the quantized space (step = 6.0)

The significance of this result lies in the fact that close to 90% of video frames share hyper-cubes with a very small number of other frames. This implies that most video frames are q-similar to very few other frames in the sequence, and as a result q-similarity allows us to effectively discriminate between video frames, as well as video sequences.

A similar experiment was performed to evaluate the quality of the color moment hashing function. We examined the distribution of video frames into hash table buckets with respect to the number of collisions. In this case a collision occurs if frames with different quantized color moments are placed in the same hash table bucket. If the average number of collisions is high, it may adversely influence the

hash table performance, and consequently slow down the shot matching process. In our algorithm we utilize a hash table with 10,000,001 entries. In this experiment we hashed the same 24-hour video sequence of 2,589,052 frames using the hash function described in section 3.6.3. We observed a total number of 278,507 collisions, which constitutes about 10% of the number of frames hashed. The average number of collisions per hash table bucket was 2.61 with the standard deviation of 6.59, and the maximum of 135 collisions was registered in a single bucket. The complete histogram of the number of collisions is depicted in Figure 124.

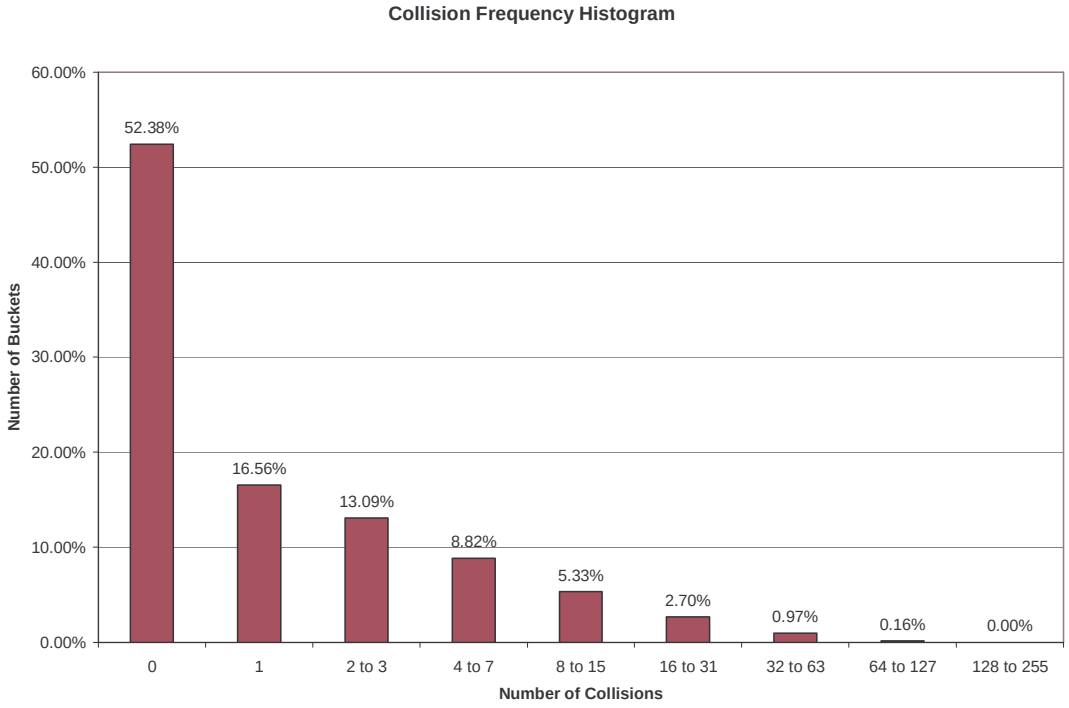

Figure 124 Histogram of the number of hash table collisions

This result shows that the hashing function chosen is not optimal, as it allows for a considerable number of collisions. Nonetheless, the distribution of collisions is such that the large majority – over 80% – of the hash table buckets contains frames from 3 or less hyper-cubes.

Summarizing, we can conclude that both the quantization scheme, as well as the hashing function have the properties desired for effective filtering of dissimilar video frames and sequences.

3.7.3 Execution Time

In order to quantitatively assess the reduction in execution time attained by the use of our filtering technique, we conducted an experiment in which exhaustive video sequence repetition detection was performed on video sequences of increasing length. Due to the overwhelming complexity of the raw video clip matching algorithm, this technique was not used for detection in this test. Instead, we compared the execution times of shot matching methods with and without filtering. The results are summarized in Table 9 and depicted graphically in Figure 125.

Length in Minutes	Frame Hashing	Frame Matching	Direct Shot Matching	Filtered Shot Matching
5	00:00:00.020	00:00:00.180	00:00:14.531	00:00:02.223
10	00:00:00.050	00:00:00.200	00:00:55.740	00:00:06.009
15	00:00:00.090	00:00:00.210	00:02:06.191	00:00:09.163
20	00:00:00.130	00:00:00.240	00:03:37.333	00:00:12.899
25	00:00:00.160	00:00:00.250	00:05:29.173	00:00:13.900
30	00:00:00.170	00:00:00.270	00:08:18.527	00:00:17.125

Table 9 Execution time of direct vs. filtered shot matching

The plots of execution time curves in Figure 125 confirm the theoretically derived complexity of the corresponding algorithms. The curve of the direct shot matching method clearly has a parabolic shape, while the duration of filtered shot matching grows linearly with the increase in video sequence length. As a result, direct shot matching is not a viable option for real-time repeated sequence detection for live news broadcasts. Conversely, identification of repeated shot using hashing and filtering can be performed in a fraction of the time needed to broadcast and capture the video sequence. This technique also scales well with the increase in the broadcast length.

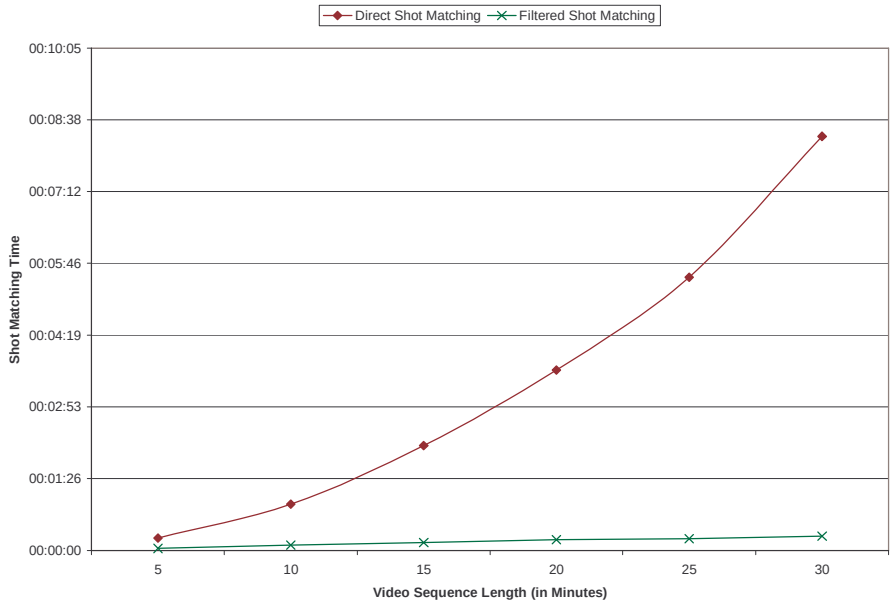

Figure 125 Execution time of direct vs. filtered shot matching

In addition, the results in Table 9 show that video frame hashing and subsequent collection prior to shot matching are performed in a small portion of the total execution time. Therefore, we can conclude that color moment hashing can be effectively used to substantially reduce the average complexity of repeated shot detection.

Naturally, the gain in execution time should not be realized at the expense of the shot matching accuracy. Hence, in the next section we will demonstrate that the hashing and filtering algorithm detects shot repetitions with recall and precision equivalent to those attained by the direct shot matching method.

3.7.4 Repeated Footage Detection Performance

The performance of our repeated footage detection method was evaluated on a group of 50 news content shots of different lengths. For each shot in the group, the ground truth was established by detecting repetitions using the repeated clip detection algorithm with a high frame match threshold. Thus detected repetitions were subsequently analyzed manually to remove all false detections, so that only true

repetitions were retained. In the first experiment, we compared the performance of the repeated shot detection algorithm with and without filtering. Both algorithms used the optimal parameter values established in section 3.7.1, i.e. frame match threshold equal to 3.0, significant length threshold of 30 frames, and clip match threshold of 0.50. The results of the direct shot repetition detection for the first 10 shots are shown in Table 10. The average performance for the entire set was 96% recall and 84% precision, yielding the utility value of 90%.

Shot No.	No. of Frames	True Matches	Detected Matches	True Positives	False Positives	False Negatives	Recall	Precision
11501	321	6	8	6	2	0	100%	75%
9534	167	4	4	4	0	0	100%	100%
10767	616	5	5	5	0	0	100%	100%
10662	333	12	12	12	0	0	100%	100%
7994	106	8	10	7	2	1	88%	80%
7996	100	7	7	7	0	0	100%	100%
7998	100	9	9	9	0	0	100%	100%
8004	66	9	10	9	1	0	100%	90%
7545	120	6	6	6	0	0	100%	100%
9860	370	4	7	4	3	0	100%	57%

Table 10 Recall and precision of repeated shot detection without filtering

The same test was conducted using repeated shot detection with filtering. The results for the same 10 sample shots are shown in Table 11, while the average recall and precision for all 50 shots were 86% and 91%, respectively. Hence, the overall utility was 88.5%. This outcome allows us to conclude that the utility value did not decrease substantially due to filtering. Therefore, in view of the great reduction of execution time, repeated shot detection with filtering is by far a superior method.

Shot No.	No. of Frames	True Matches	Detected Matches	True Positives	False Positives	False Negatives	Recall	Precision
11501	321	6	5	5	0	1	83%	100%
9534	167	4	4	4	0	0	100%	100%
10767	616	5	5	5	0	0	100%	100%
10662	333	12	11	11	0	1	92%	100%
7994	106	8	4	4	0	4	50%	100%
7996	100	7	7	7	0	0	100%	100%
7998	100	9	9	9	0	0	100%	100%
8004	66	9	9	9	0	0	100%	100%
7545	120	6	4	4	0	2	67%	100%
9860	370	4	5	4	1	0	100%	80%

Table 11 Recall and precision of repeated shot detection with filtering

In order to demonstrate the importance of partial shot repetition detection for story tracking in news broadcasts we conducted the same experiment, but required that repeated shots be of equal length. To account for imperfections in temporal video segmentation, which lead to imprecision in transition boundary detection, we allowed repeated shots to differ in length by up to 10%. This approach was used by Pua [Pua02]. The detection results using this method for the 10 sample shots are presented in Table 12, while the recall and precision for the whole set of shots were 45% and 100%, respectively.

This experiment confirms that video footage reused by television news stations is indeed frequently repeated only in part. Complete shot repetition detection methods fail to recognize such partial repetitions and consequently are not a good basis for story tracking in television news broadcasts. Conversely, the partial repetition detection technique introduced in this work is highly effective and attains recall of 86% and precision of 91%.

Shot No.	No. of Frames	True Matches	Detected Matches	True Positives	False Positives	False Negatives	Recall	Precision
11501	321	6	1	1	0	5	17%	100%
9534	167	4	1	1	0	3	25%	100%
10767	616	5	1	1	0	4	20%	100%
10662	333	12	8	8	0	4	67%	100%
7994	106	8	2	2	0	6	25%	100%
7996	100	7	1	1	0	6	14%	100%
7998	100	9	8	8	0	1	89%	100%
8004	66	9	5	5	0	4	56%	100%
7545	120	6	1	1	0	5	17%	100%
9860	370	4	4	4	0	0	100%	100%
Summary							45%	100%

Table 12 Recall and precision of the completely repeated shot detection with filtering

3.8 Conclusions

In this chapter, we introduced a video sequence repetition detection algorithm for live video news broadcasts. The method proposed works in real-time and can effectively deal with partial shot repetition, which is essential for effective story tracking. In order to create this algorithm, we first proposed a set of video sequence similarity metrics. We analyzed their properties in the context of video news, and found that the metric based on raw moments is most suitable for our purposes, although it is sensitive to global image changes, such as brightness adjustments. We demonstrated that repetition detection in the absence of shot detection results is extremely time consuming, and showed that detection of repeated shots can be performed substantially faster. We examined the impact of imperfections in temporal segmentation, and concluded that our method of partial shot matching can handle them relatively well. Finally, we described the hashing and filtering technique, which allowed us to reduce the execution time requirements to within real-time constraints. At the end of the chapter, we evaluated the performance of our repeated sequence

detection method, and showed that it attains high accuracy (86% recall, and 91% precision).

Chapter 4

Story Tracking

4.1 Introduction

In Chapter 1, we presented the problem of the lack of effective access to information contained in video. We also indicated that this issue is particularly pronounced in the domain of video news broadcasts. The contemporary world is awash with news from all over the globe coming to us continuously from a multitude of television news stations. In theory then, virtually anyone could have instantaneous access to the latest news at almost any time. Practically, however, gaining such access would require constant monitoring of all available news sources, which is humanly impossible. Moreover, perpetual viewing of even a single news channel would prove very time-inefficient, as a vast majority of information provided is of little or no interest to the user. Also, quite commonly the ratio of truly new information in the news is fairly small, and a lot of material is redundant. Thus, there is a need to create methods of effectively accessing news information which is of relevance to the user.

In the realm of video news broadcasts, which provide an overwhelming amount of information, it is important that the viewers be able to focus their attention only on the news of interest to them. In addition, they should be able to access new

information pertaining to their interest as it becomes available over time. This mode of accessing video news is facilitated by a technique called story tracking. This technique enables viewers to choose an interesting story in a news broadcast, follows the development of that story over time, and provides an effective means of viewing it entirely or in part. Thus, story tracking is essential for providing effective and intuitive access to video news broadcasts.

In this chapter, we present a story tracking method inspired by the observation that news stations frequently reuse video material when reporting the same story (see section 1.1). Based on this fact, we developed an algorithm which identifies repetitions in the video stream of the news broadcast, and then uses this information to track the development of news stories.

Before we provide details of our story tracking technique, we need to precisely define the problem of story tracking, as well as introduce the necessary terminology which will be used throughout this chapter. We have already referred to the concept of a news story. The following definition appeals to the intuitive understanding of the notion.

*Definition 17: A **news story** is the subject matter of news reporting, and consists of an event or a set of related events which occur in the real world.*

Television news stations exist to report news stories. As the news story is related verbally, it is accompanied by video footage which may provide visual clues regarding the news story. The video footage may be segmented into a sequence of video shots, as it was described in Chapter 2. If a video news broadcast is temporally segmented, each of the resulting shots may be associated with one or more news stories, as in the following definition:

*Definition 18: A shot s in the video news broadcast is **relevant** to a given news story Φ if it is displayed while the news story is reported.*

The definition above makes no restrictions on the number of shots which may be relevant to any given news story. A news story may have a single relevant shot, or it

may be associated with multiple relevant shots. A shot may also be relevant to more than one news story. For instance, an anchor person may finish reporting one news story and switch to another in a single video shot.

Having described the concept of shot relevance to a *news story*, we can now define the notion of *story* in a video news broadcast, which from now on will be referred to as a *story* for simplicity.

Definition 19: Given a news video sequence V consisting of N shots $V = \langle s_1, s_{2,} \ldots, s_N \rangle$, *a* ***story*** $S_\Phi = \{s_1^\Phi, s_2^\Phi \ldots, s_l^\Phi\}$ *in V is the set of all shots in V relevant to a single news story* Φ.

Although it is theoretically possible that the entire story may consist of only a single shot, such stories rarely occur in practice. Usually, news story reports take at least a few minutes and, therefore, their corresponding stories span several shots. In addition, multiple reports on the same story may take place at different times during the day, and be separated by other reports or commercials. Thus, it is logical to divide a story into a number of disjoint segments, each corresponding to a single report on the story during the broadcast. Each story segment consists of shots relevant to the news story, which are consecutive in V.

Definition 20: Given a news video sequence V and a story S_Φ, *a* ***story segment*** *is a set of shots in* S_Φ, *which are consecutive in V, i.e.*

$$E_\Phi = \{s_k^\Phi, s_{k+1}^\Phi \ldots, s_{k+n}^\Phi\}, such\ that\ \forall i = k,...,k+n : s_i \in S_\Phi \wedge s_{k-1} \notin S_\Phi \wedge s_{k+n+1} \notin S_\Phi$$

The set of concepts introduced above allows us to precisely describe the task of story tracking in video news broadcasts, which is the focus of this chapter.

Definition 21: The ***problem of story tracking*** *is as follows. Given a news video sequence V consisting of N shots* $V = \langle s_1, s_{2,} \ldots, s_N \rangle$, *and a nonempty query set* $Q = \{q_1, q_{2,} \ldots, q_M\}$ *consisting of shots relevant to the news story* Φ, *determine the story* S_Φ *in V.*

A story, as defined above, does not imply any particular ordering of the relevant shots. It also, by definition, includes all repeated relevant shots. For purposes of presenting the story to the user, the shots need to be arranged into certain order, and not all of them need to be shown. The choice of the subset of shots to be shown, as well as their ordering is determined by the purposes of performing story tracking, or simply user preferences. A subset of story shots, along with their order, determines a story view. In other words, a ***story view*** is a subset of shots in S_Φ which has been arranged for presentation, which may be formally defined as follows:

Definition 22: ***Story view*** *is a subset W of shots in* S_Φ *along with a partial order* $\leq$ *defined on W, i.e.* $\langle W, \leq \rangle$.

If all shots are arranged in a sequence, in which case $\leq$ becomes a total order, the story view is called ***linear***. The task of creating the story view for a given story is the subject of story presentation.

In this chapter, we introduce a solution to the problem of story tracking in video news broadcasts based on redundancies in video material. We rely on video sequence repetition detection methods developed in Chapter 3 to identify story segments, and combine them into a single cohesive story. We evaluate the performance of our story tracking methods using the standard information retrieval parameters of recall and precision. In addition, we explore techniques of automatically classifying news shots based on their repetition patterns, as well as examine performance improvements in story tracking which may be achieved using this classification. Finally, we propose two different methods of story presentation and discuss their advantages and disadvantages with different story tracking approaches.

The remainder of this chapter is organized as follows. Section 4.2 describes related research. Section 4.3 presents our general approach to story tracking and explains story representation used by our algorithms. In section 4.4, we present our story tracking algorithm based on repeated shot detection. We evaluate the algorithm's performance on real-world news broadcast and discuss potential problems discovered

in the course of the experiments. Section 4.5 discusses improvements which can be introduced into our method by using news shot classification. We show how the algorithm can be adapted to incorporate this additional information, and propose automated methods of shot classification. Section 4.6 presents a conceptual overview of different methods of story presentation. Finally, the chapter closes with section 4.7, which contains a summary and conclusions.

4.2 Related Work

The problem of story tracking is relatively new, and was first posed as part of the Topic Detection and Tracking (TDT) initiative in 1997. Research in the field can be classified into two broad categories: techniques that focus on textual information (written and spoken), and methods that utilize visual information (images and video). In both cases, domain specific information can be exploited to detect and track stories.

4.2.1 Textual topic detection and tracking

The Topic Detection and Tracking (TDT) initiative started as a joint effort between *DARPA*, the *University of Massachusetts' Center for Intelligent Information Retrieval*, *Carnegie Mellon's Language Technology Institute*, and *Dragon Systems*[All98a, All98b]. TDT is a research program investigating methods for automatically organizing news stories by the events that they discuss. TDT includes several evaluation tasks, each of which explores one aspect of that organization – i.e., splitting a continuous stream of news into stories that are about a single topic (“segmentation"), gathering stories into groups that each discuss a single topic (“detection"), identifying the onset of a new topic in the news (“first story detection"), and exploiting user feedback to monitor a stream of news for additional stories on a specified topic (“tracking”). The domain of TDT's interest is all forms of written or spoken broadcast news, but does not include video.

A year-long pilot study was undertaken to define the problem clearly, develop a test bed for research, and evaluate the ability of current technologies to address the problem. Results of that study were reported at a workshop in October of 1997, and a final report was made at a related workshop. The groups involved in the tasks found that current methods are capable of providing adequate performance for detection and tracking of events, but that there is a high enough failure rate to warrant significant research into how the state of the art can be advanced.

The name "Topic Detection and Tracking" suggests that TDT deals with *topics*, whereas the description cited above defines TDT using the term *event*. This implies that the terms are synonymous, which is not the case. Therefore, in order to avoid confusion in the future, a distinction between the two needs to be made.

Let us consider an *event* that occurred in the real world at a certain time and place. That event may later be reported in the news broadcast in the form of a *story* (strictly *episode* of a story). Later, as additional facts surface, new reports may be broadcast, and thus become new *episodes* of the same *story*. Moreover, new *events* related to the original event may happen and be reported in the news, thus adding new *episodes* to the same story, or if the *event* is significant enough in itself, starting a new *story*. So, *stories* evolve over time to include new *episodes* describing or related to the original event, but *events* themselves remain static, situated in their time and place. Finally, if we collect all the *stories* describing or related to a certain type of *events*, that collection becomes a *topic*.

Hence, we propose the following taxonomy:

Event: An event corresponds to something that occurred in the real world at a certain time and place, e.g., President Bush's trip to the Middle East in May 2001.

Episode: An episode is a temporally continuous description of an event, such as a relation of the President's visit to the Middle East in the morning news on a certain day.

Story: A story is a sequence of episodes describing or related to a certain event. The entire coverage of the President's visit in the news over a period of time is an example of a story.

Topic: A topic refers to a general notion of some potential subject of interest, such as politics, natural disasters, movies, etc. As such, it can be viewed as a collection of stories sharing a certain theme, for example, U.S. foreign politics, or East Coast hurricanes.

Because TDT deals primarily with events, it is perhaps more appropriately called Event Tracking [All98b]. In fact, to be precise, we should say that since events occur once and remain immutable over time, they can be detected but not tracked. Stories, on the other hand, do not occur, strictly speaking, and are only a reflection of the events in the news, but they do evolve over time. Therefore, their development over time can be tracked. Hence, we will use the name Event Detection and Story Tracking (EDST) in place of TDT throughout the rest of this work.

Event detection and story tracking

The problem of EDST can be divided into two subtasks: new event detection and story tracking. These tasks were extensively analyzed by Allan *et al.* [All98a, All98b] and described as follows.

"The goal of those tasks is to monitor a stream of broadcast news stories so as to determine the relationships between the stories based on the real world events that they describe. New event detection requires identifying those news stories that discuss an event that has not already been reported in earlier stories. Event tracking means starting from a few sample stories and finding all subsequent stories that discuss the same event." In this case the phrase "broadcast news stories" refers to textual news services, and transcripts of TV news broadcasts, and does not include video.

As described above, EDST is closely related to the Information Retrieval (IR) problems of Information Filtering and Routing. However, there are two important differences:

First, EDST is restricted to the domain of news, which is an ideal medium for finding coverage of events. Filtering and Routing, on the other hand, apply to unrestricted corpora covering arbitrary topics.

Second, the "query" in EDST refers to an event, and is specified using a few examples of stories about this event. By contrast, IR queries are generally at the broader level of topic, and are described directly by the user.

Hence, the field of EDST is defined more narrowly than Filtering and Routing, and allows for the evaluation measures to be more easily agreed upon.

Event detection methods

Allan *et al.* [All98b] created an algorithm which detects new events based on commonly known word statistics in Information Retrieval. They represent episodes and queries as vectors of weighted features, which were chosen to be all nouns, verbs, adjectives, and numbers. The algorithm builds a query from one training episode using the *n* most frequently occurring features. It also computes an initial similarity threshold. All subsequent episodes are compared to the query by computing the similarity measure. If its value is above the similarity threshold, then the episode is pronounced to describe the same event. Otherwise, a new event is detected.

The authors observe that the likelihood that the episode describes the same event as some earlier episode decreases with time. Therefore, they increase the threshold value with the temporal difference between compared episodes.

Story tracking methods

Allan *et al.* [All98b] propose an event tracking algorithm based primarily on Information Filtering. The algorithm uses a certain number of stories (N_t) to form a

query and a threshold for matching episodes to the query. All subsequent stories are then compared to the query and, if the match is above the threshold, the episode is considered to be about the same event. A query is composed of a certain number (n) of features most commonly occurring in the N_t training episodes. The authors show that the performance of their method is stable across a range of n and appears to be optimal for 10-20 features. They also demonstrate that, although increasing the number of training stories (N_t) improves performance, raising the value of N_t above four provides little help.

Allan *et al.* admit that this approach alone does not perform very well due to the evolution of event coverage over time. Namely, in the initial phase of reporting usually not much information is available, but as additional facts are discovered they are included in the coverage and often become the focus of the story. Therefore the word-based feature model of the story changes over time. To account for this phenomenon, they introduce an adaptive tracking algorithm that amends the original method by incorporating features of the detected episodes of the same story into the query for further story tracking. This way, the query evolves along with the model of the story.

Papka [Pap99] provides a brief description of other event tracking methods presented for evaluation on TDT2. They demonstrate that all of the techniques shown perform similarly according to the NIST evaluation. Considering that textual event tracking is not a primary focus of our research, we will limit ourselves here to only briefly mentioning other textual story tracking systems.

BBN The *BBN* tracking system is based on formulating a mixture of classifiers from three models: Topic Spotting (TS), Information Retrieval (IR), and Relevance Feedback (RF). The first two approaches are based on a probabilistic approach to word occurrence distributions. The TS model assumes that words in the test story are generated by the model from the training stories; the IR model assumes that the training stories are generated by the model from the test story; and the RF approach

used frequently occurring terms in the training stories. In their report, they also show improvements using an adaptive query formulation approach. [Jin99]

CMU The group from *Carnegie Mellon University* tested Decision Trees (DT) and a K Nearest Neighbors (KNN) approach to tracking. In their DT approach, they used features in addition to word cooccurrence statistics including the location of a word relative to the beginning of the story, whether the root of the word appeared in the story, and an adaptive time window approach. The KNN 65 approach used a tf · idf document representation. Their analysis suggests that the KNN approach appeared slightly more effective than the DT approach. [Car99]

DRAGON *Dragon Systems* uses statistical approaches based on a beta-binomial model and a unigram language model. Their data suggest that a mixture of their approaches leads to improved tracking effectiveness. They also apply background models that are constructed from an auxiliary corpus. A document is considered relevant to an event if it is more similar to the model resulting from the training documents than to one resulting from a background model. [Yam99]

UPENN The system from the *University of Pennsylvania* is based on a similar representation to the one used in [Pap99]. They used a tf · idf representation for classifiers and documents, where incremental idf was seeded with the document frequencies from the TDT1 corpus. A cosine similarity function was used to compare classifier and document vectors. [Sch99]

4.2.2 Multimodal techniques of video news organization

The concept of topic detection and tracking has been extended to include non-textual sources of information. Although text-based techniques can be applied to the textual transcripts of video news broadcasts, its methods completely ignore the rich layer of visual information present in the video stream. Therefore, in recent years initial research has been done on incorporating audio and visual characteristics of video into news organization techniques.

Ide *et al.* [Ide00] attempt to identify news scenes by comparing background images using simple visual features. For this purpose, they manually remove the region corresponding to the person in front of the camera, such as an anchor or reporter, and compute basic color histogram and correlogram of the resulting image. These features are then compared against a database of features calculated for some known locations. The method was tested on a very small database of only 3 different locations: parliament, cabinet meeting, and a news briefing. Although the performance of location detection on a video sequence of about 3 hours was reported to be very high, this approach is unlikely to scale well to a large database with significant number of different locations.

A few papers have been published on the subject of classification of shots in news broadcasts. Some simple commercial detection techniques use black frames to detect commercials [Lie97, Hau98]. However, such simple approaches must fail for television channels that do not use black frames to flag commercial breaks. Also, black frames used in other parts of the broadcast will cause false alarms. Furthermore, progress in digital technology obviates the need to insert black frames before commercials during production. An alternative makes use of shorter average shot lengths as in [Mar01]. However, this approach depends strongly on the 'high activity' rate which may not always distinguish commercials from regular broadcasts.

In more recent research, face recognition is used to identify anchor person shots [Ide99b, Hau03]. Ide *et al.* [Ide99b] use face recognition to identify three additional shot classes: speech, walking person, and gathering. They also distinguish graphics shots by detecting large numbers of still frames in the shot. Duygulu *et al.* [Duy04a] propose a commercial classification technique which relies on unique characteristics of commercial sequences. First, they observe that commercials are frequently repeated throughout news broadcasts. They also utilize visual and audio characteristics which distinguish commercials from the rest of the broadcast. They combine the results of detection of repetitions with the outcome of the video and audio classifier to identify commercials. In their method, video shots are represented

by keyframes, and this representation is used for both detection of repetitions, and derivation of visual features. A number of image features are used to recognize repetitions of keyframes. Duygulu *et al.* report very high accuracy of their combined method (recall and precision around 90%).

In another article, Duygulu *et al.* [Duy04b] use shot classification based on visual features to improve association of video shots with words for news videos. They classify news video shots as: anchors, commercials, graphics and studio settings, and other. They detect commercials using the method developed in [Duy04], and identify anchor person shots using a classifier proposed in [Hau03]. In addition, they manually detect graphics and studio shots by first clustering all remaining shots by their color composition, and then visually inspecting representatives of the clusters. After all these types of shots have been recognized, the authors divide the news broadcast into segments by applying a heuristic based on the shot classification. Their segmentation algorithm is as follows:

1. Start a new segment after a graphics shot or a commercial
2. End the current segment if the next shot is a commercial or a graphics shot
3. Start a new segment on an anchor shot which follows a non-anchor shot
4. End the current segment on an anchor shot followed by a non-anchor shot

Their method was tested on a relatively small data set of 114 story segments, and a relatively low accuracy of approximately 60% was reported.

The story tracking technique we propose in this work relies solely on visual characteristics of the television news broadcasts. It is, therefore, complementary to textual story tracking and could be used in conjunction with it to discover correlations between story segments that do not have textual similarity. For example, it may be used when news is reported in different languages or when closed captions are not available. Our method could also help distinguish between two story segments related to the same person or place, but reporting different stories. On the other hand,

textual story tracking could provide additional clues for linking story segments that do not contain common video footage.

4.3 Method Overview

In this section, we describe the general concept of our story tracking method. We present the input parameters and internal story representation used by our algorithm. We also discuss evaluation strategy.

Our story tracking method addresses the following usage scenario. A person watches a live news video channel and finds certain news story interesting. He or she selects the interesting portion of the broadcast and provides it as the input for our algorithm. The algorithm analyzes the query and begins tracking the story by means of detecting repetitions of video shots contained in the query. At any later point in time, the viewer may request to view the story of interest. In response, our algorithm returns a desired view of the story.

4.3.1 Inputs

In addition to the query set Q described earlier, the story tracking algorithm takes as a supporting parameter a partition of the source video sequence induced by a shot match relation. The significance of this parameter is explained below.

The partition of the source video sequence V is determined by the *shot match relation*, and groups together matching shots. A shot match relation is an equivalence relation, which can be inferred from a video sequence similarity metric introduced in Chapter 3. In that chapter, we also showed that given similarity metric α, and a threshold on the value of this metric, we could determine all pairs of matching shots in a video sequence. If only complete shot matching is required, then the set of all such pairs along with all the shots which did not match any other shots, forms an equivalence relation in V. The three necessary properties of an equivalence relation: reflexivity, symmetry, and transitivity follow directly from the definition of the

similarity metric. Thus, for every shot s_i we can determine an equivalence class, as the set of all shots s_j which match s_i, i.e. $\varepsilon(i) = \{s_j : s_i \text{ matches } s_j\}$

On the other hand, if partial shot matching is allowed, the relation inferred may not be transitive. This issue is best illustrated by a diagram. Suppose we have three shots *A*, *B*, and *C* shown in Figure 126, which contain identical sequences of frames indicated by the overlap between the shots. In this example, shot *A* partially matches *B*, and *B* partially matches *C*. However, shots *A* and *C* do not share any sequence of frames, and therefore do not match even partially. Consequently, pairs (*A*,*B*) and (*B*,*C*) are in the relation, while (*A*,*C*) is not, and so the relation is not transitive.

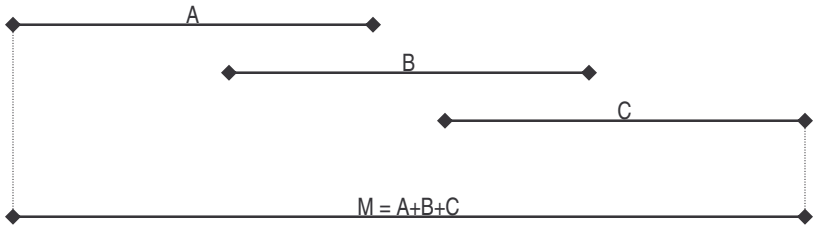

Figure 126 An example of a non-transitive shot matching relation

In order to ensure transitivity of the shot match relation, we need to take a transitive closure of the inferred relation. Thus, if any two pairs of shots (*X*,*Y*) and (*Y*,*Z*) match, then the pair (*X*,*Z*) is automatically considered matching. This corresponds to the example in Figure 126. Here shots *A*, *B*, and *C* must, in fact, be a shortened version of some shot *M* which is the union of all three. Taking a transitive closure of the match relation is equivalent to comparing each shot to the combined shot *M*. Consequently, we can formally define the equivalence class for any shot s_i as:

$$\varepsilon(i) = \{s_i\} \cup \{s_j : \exists s_k \in \varepsilon(i) : s_k \text{ matches } s_j\}$$

A set of all matching shot equivalence classes defines a partition *P(V)* on *V*.

The two input parameters are assumed to be available at the start of the algorithm. However, in practice only the query *Q* is presumed entirely accurate, that is containing only shots relevant to the news story in question. The partition *P(V)* may

contain errors if the temporal segmentation or repeated shot detection are imperfect. For instance, some pairs of shots may be determined matching, while in reality they were not.

At the start of the algorithm, the input parameters are inserted into an internal representation of the story, which will be described in the next section.

4.3.2 Internal Story Representation

The main task of the algorithm is to build the story *S*, i.e. create the set of shots relevant to the news story, from the given query *Q*. The story is built gradually as the video content becomes available using an iterative algorithm. In order to facilitate the process of story building, the algorithm uses a story board – an internal structure representing the story at a given time. The story board is a tuple $SB_\Phi = \langle \Sigma, \Omega, \mathrm{P}(\Sigma), \delta \rangle$, in which Σ is the story as detected by the algorithm, Ω is the story core, $P(\Sigma)$ is a partition of Σ, and δ is the co-occurrence function.

The story core Ω is a subset of the story Σ and comprises all shots whose occurrences are tracked by the algorithm. $P(\Sigma)$ is a partition induced by a shot match relation, and is provided as a supporting input parameter. Co-occurrence function $\delta : V \times V \to N \cup \{0\}$ assigns a non-negative value to every pair of shots in *V*. The value of the function is zero if the two shots do not co-occur in the same story segment. If they do co-occur, the function δ assumes a value equal to the temporal distance between the two shots.

A story board can be represented as a non-directed graph whose vertices are all the shots in Σ. The graph contains two types of edges: co-occurrence edges and shot match edges. Co-occurrence edges connect pairs of shots for which the co-occurrence function δ is non-zero. Each edge of this type has a weight equal to the value of the function δ. Shot match edges join matching shots, and have no weight assigned to them. In addition, we distinguish the core of the story as a sub-graph of the story graph. The core contains the shots whose repetitions we want to track.

Figure 127 depicts a sample story board graph consisting of two segments: $\langle A1,B,C1,D \rangle$ and $\langle E,A2,F,C2 \rangle$. Each segment is depicted on a separate line. In the graph, solid lines represent shot match edges, while dashed lines show the co-occurrence edges. In addition, shots whose names are italicized belong to the story core. In this story board, we have six shot equivalence classes: {*A1*, *A2*}, {*B*}, {*C1*, *C2*}, {*D*}, {*E*}, and {*F*}, and the story core consists of {*A1*, *A2*, *C1*, *C2*}.

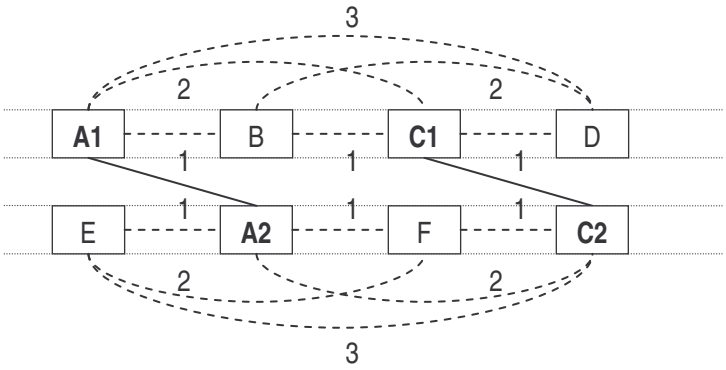

Figure 127 Sample story graph

The co-occurrence edges may be shown explicitly as in the diagram above. However, thanks to the rule of depicting all shots in one episode on a single line, and separate episodes on separate lines, these edges may be inferred to fully connect all shots in the same line. This simplifies the graph diagram, and prevents the co-occurrence edges from obscuring other details. As an example, Figure 128 depicts the same graph in its simplified form. In the graph, the time runs left to right and top to bottom, that is all shots in the same row were broadcast in their left to right order, and all shots in row *i* precede all shots in row *i+1*. In the rest of this chapter, we will use the simplified graph representation.

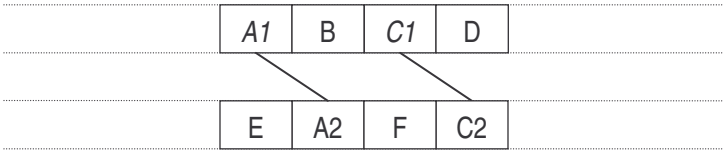

Figure 128 Simplified sample story graph

The story tracking algorithms which will be presented in this chapter build a story board in an iterative fashion starting from the set of query shots. In order to present a story board, as it is created, in the broader context of the surrounding shots from V, we adopt the following graphical notation.

Figure 129 Graphical notation for story graphs

Figure 130 depicts a graph representation of a sample story consisting of three episodes: $\langle A1,B,C1,D \rangle$, $\langle E1,A2,F,C2,G1,H \rangle$, and $\langle E2,G2 \rangle$. Although the diagram shows additional shots, only the shaded shots belong to the story graph. In the graph, shots *A1*, *A2*, *C1*, *C2*, *E1*, *E2*, *G1*, and *G2* belong to the story core. Shot *C1* was given as the query, and all shots *X1* through *X8* are irrelevant to the story. The graph also shows matches between shots (*A1*, *A2*), (*C1*, *C2*), (*E1*, *E2*), and (*G1*, *G2*). The temporal distance edges were omitted for simplicity and are implied to link all pairs of shots in the same row. Each of them has an associated weight equal to the number of shots separating the beginnings of the two shots.

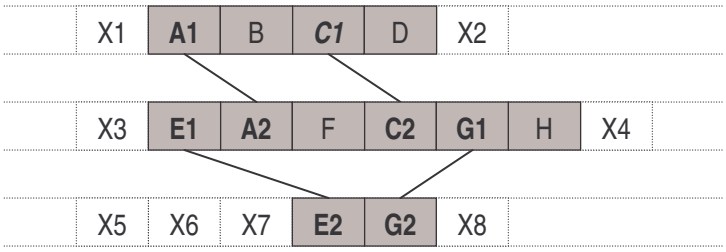

Figure 130 Sample story graph

4.3.3 General Algorithm

Given the input query, our story tracking algorithm begins to build the story in a gradual fashion, iteratively adding shots until no more shots can be added. This is

shown on the block diagram in Figure 131. First, all the query shots are placed in the story core. Then, the algorithm operates in three main steps which are repeated cyclically. For every shot in the core, the algorithm selects the next repetition of that shot from the corresponding equivalence class. Subsequently, a story segment is built around the newly discovered matching shot. All shots in this segment are added to the story Σ. In addition, some of the shots in this segment may be added to the core depending on whether they satisfy certain criteria. Once this is done, another matching shot is chosen for the current core shot, and all steps are repeated. If there are no more matching shots for the current core shot, the next core shot is chosen. When there are no more shots can be added to the core, the algorithm terminates and the story Σ is returned as the result.

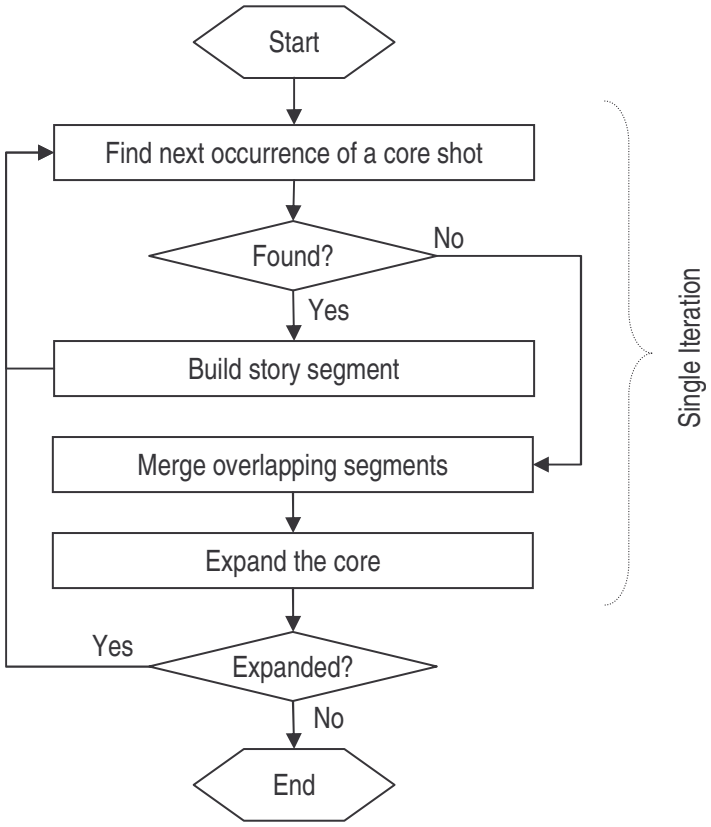

Figure 131 Block diagram of the general story tracking algorithm

In the following section, we present and evaluate an automatic story tracking method. A number of variations of the technique are considered which differ primarily in the strategies used to build story segments and expand the story core. Depending on the strategy used, the story tracking algorithm may achieve very different performance, which will be the subject of performance evaluation.

4.3.4 Evaluation Approach

In order to examine the accuracy of our story tracking methods on real world data, we recorded a 24-hour video broadcast of a typical day of CNN News channel. The video was captured using Windows Media Encoder 9.0 and stored in the Windows Media format at a frame rate of 30 frames per second with frame size of 160 by 120 pixels. The resulting video file is 40 GB in size, and represents a reasonable compromise between video quality and storage requirements.

The news broadcast contains several stories, some of which comprise only one segment, while others contain several. For purposes of story tracking evaluation, we chose one story which consists of a number of episodes. The story regards the arrest of Michael Jackson in connection with alleged child abuse charges. All segments of the story were manually annotated to establish the ground truth for the experiments.

The overall performance of a story tracking method may be evaluated using the standard information retrieval measures of recall and precision. Both concepts were introduced in section 2.4 and can be applied to story tracking results in the following manner. By definition, a story is a set of shots, and so the tracking performance may be viewed as the accuracy of story shot detection, and will be evaluated in terms of shot recall and precision. Shot recall is the ratio of the number of correctly detected story shots to the number of all shots in the actual story. Shot precision is equal to the ratio of the number of correctly detected story shots to the total number of story shots reported by the algorithm.

$$R_{shot} = \frac{number\ of\ \ correctly\ reported\ story\ shots}{number\ of\ \ all\ shots\ in\ the\ story}$$

$$P_{shot} = \frac{number\ of\ \ correctly\ reported\ story\ shots}{number\ of\ \ all\ reported\ shots}$$

The story tracking technique discussed in the following sections will be evaluated using these two performance measures.

4.4 Implementation and Evaluation

In this section, we introduce our story tracking method, which relies on the shot match information and shot co-occurrence function. We begin by presenting an algorithm which tracks stories by simply detecting shot repetitions. We then discuss our general strategies of building story segments around repeated shots, which is followed by a presentation of two different story core expansion schemes. The accuracy or our story tracking technique is evaluated on the experimental data described above. At the end of this section, we summarize the results and examine the main challenges.

4.4.1 Segment Building Strategies

One can conceive of a story tracking algorithm which does little more than detect repetitions of the query shot or shots. In such a method, all query shots are placed in the core, and the detection process starts. When a repetition is identified, it is added to the story, as in the following example.

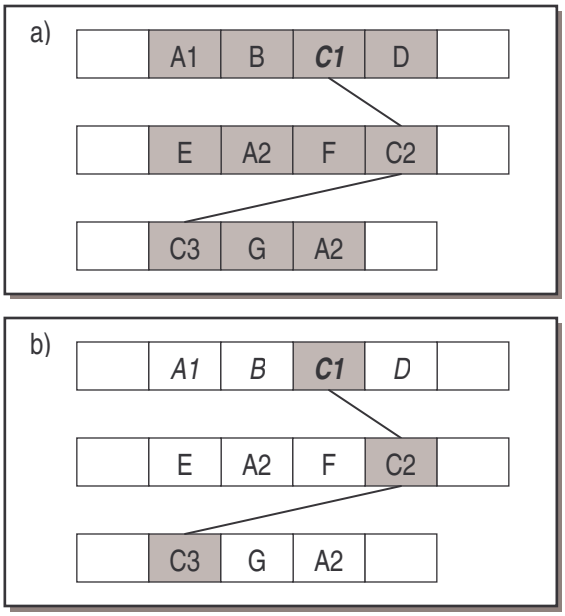

Figure 132 Story graph a) of the actual story, b) as detected by the basic tracking method

The example shows that although the actual story consists of several shots in three episodes, the algorithm recognizes only repetitions of the query clip *C1* as belonging to the story. Thus, the story detected by this technique is uninteresting. If the user has already viewed shot *C1* and selected it as the query, then viewing repetitions of *C1* will add little to his knowledge of the news story.

Clearly, the story detected by any tracking method should be extended beyond the original query shot. We need a method of building an episode around every repetition of the query shot. For this purpose, we need to determine the episode boundaries. One approach is to assume that the every occurrence of a query shot is at the center of the corresponding episode. Hence, we can examine a symmetric neighborhood centered on the query shot, and consider all shots in the neighborhood to belong to the episode. Once an occurrence of a query shot is detected, all shots in the neighborhood may be added to the story. Let us assume the neighborhood size was set to two shots. In the same example with shot *C1* given as the query, this extension would result in the following story (see Figure 133).

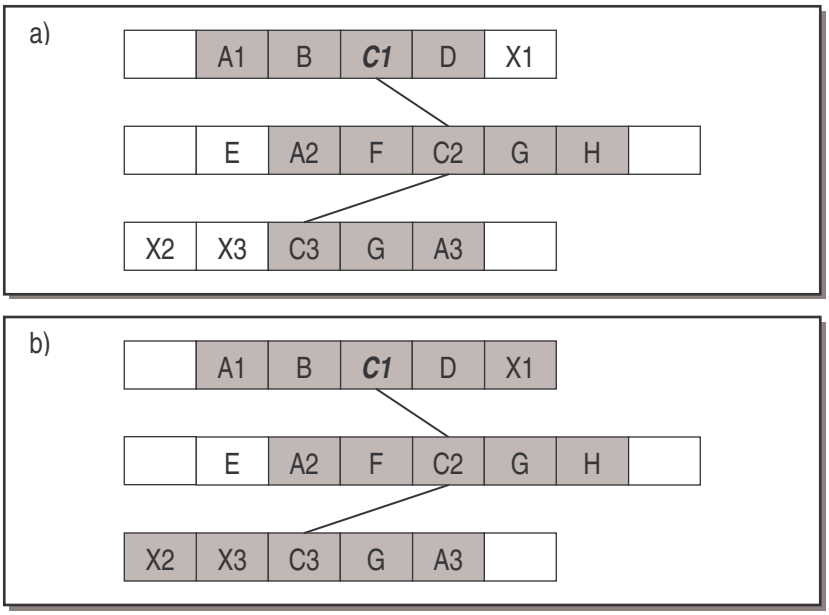

Figure 133 Story graph a) of the actual story, b) as detected by the basic tracking method with liberal segment extension

This example demonstrates that liberal segment extension offers much better coverage of the actual story. Of all actual story shots, only one (*E*) was not included as part of the story by this technique. On the other hand, this method of segment extension may introduce some irrelevant shots into the story, especially if the query shot appeared closer to the episode boundary. In the example above, shots *X1*, *X2*, and *X3* are not relevant to the story, but were included by the algorithm. Therefore, this segment building method may be called *liberal*.

The accuracy of this segment building strategy depends on the chosen neighborhood size. If the size is set to a large value, then the segments built by the algorithm will be long and may include irrelevant shots. On the other hand, small neighborhood size may lead to omission of story shots. The optimal value of this parameter could be determined experimentally, as the average story segment length in a given video domain.

To alleviate the shortcomings of the liberal method, one can devise a *conservative* segment building technique. This method relies on a larger number of query shots.

In our example, if both *A1* and *C1* were given as the query, then the algorithm could detect shots *A2* and *A3* as repetitions of *A1*. If repetitions of both *A1* and *C1* are detected in close proximity to each other, one can assume that the shots between them belong to the episode of the story. Thus, the conservative segment building technique adds such shots to the story, as shown in Figure 134. The story shots lying outside of the shot span defined by the query shots are not added to the story. Naturally, if only one core shot is found in a new episode, that episode is not expanded.

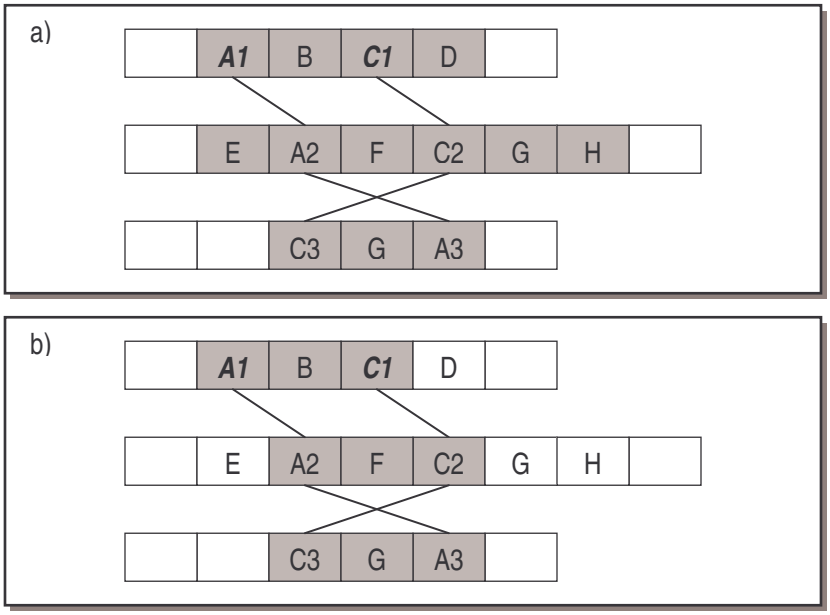

Figure 134 Story graph a) of the actual story, b) as detected by the basic tracking method with conservative segment extension

The two segment-building strategies may be combined into a single algorithm. When an occurrence of a core shot is identified, the algorithm searches for occurrences of other core shots in a neighborhood of a certain size, called a *search window*. The size of the search window should be determined experimentally based on the average length of a story segment in a given new source. If another core shot is found, then the search continues also around that shot. Once all core shots in the segment have been found, the algorithm marks the new segment as stretching from the earliest to the latest core shot. In addition, the segment is extended by a small

number of shots beyond the bounding core shots. This algorithm combines the advantages of both earlier techniques, as it allows for the segment to become quite large if multiple core shots are discovered. At the same time, if only one core shot is found, the segment built around it is relatively small, but contains more than just the core shot.

In all methods presented so far, the story core remained the same, and contained only the query shots. These methods may, therefore, be classified as *static core methods*, and all share one weakness. As the story develops over time, new video footage becomes available, which may be more relevant to the latest events. Thus, old footage may be gradually phased out, while the new footage is introduced. After a few story segments, the original query shots may not appear any more. As a result, static core methods would be unable to continue tracking the story past the point of the last occurrence of any of the original query shots. On the other hand, footage introduced later is likely to be reused in later segments. Therefore, if such new footage is recognized, the tracking may continue. This issue is addressed by the *dynamic core techniques*, which are presented in the next section.

4.4.2 Dynamic Core Expansion

In this section, we discuss our dynamic core story tracking approach, which deals with the problem of evolution of the video footage used in story segments over time. To this end, our dynamic core technique expands the core to include certain shots found in the new segments of the story. When the story tracking algorithm identifies a new segment and determines its boundaries by the *liberal* or *conservative* building strategy, some or all of the shots in the new segment may be added to the core, according to the core expansion scheme.

In the *optimistic expansion scheme*, all shots in the new segment are added to the core. Using this approach, we assume that all shots in the segment are, in fact, relevant to the news story. If this is the case, the optimistic scheme maximizes our chances of detecting new segments of the story. However, if some of the shots

belong to a different story (which is possible because of the imperfections in segment building), our algorithm will begin to track two stories simultaneously, and the resulting story will likely be a combination of the two. To illustrate this point, we can examine the example shown in Figure 135. The diagram depicts segments of two different stories, one consisting of shots *A* through *H*, and the other comprising shots *M* through *P*. In part b) the liberal segment building strategy was used, and consequently shot *M1* was included in the segment. If the core is expanded optimistically, shot *M1* becomes part of the core. As a result, the next segment of the story *M-P* is considered part of the story *A-H*. This incorrect expansion adversely affects the precision of story tracking.

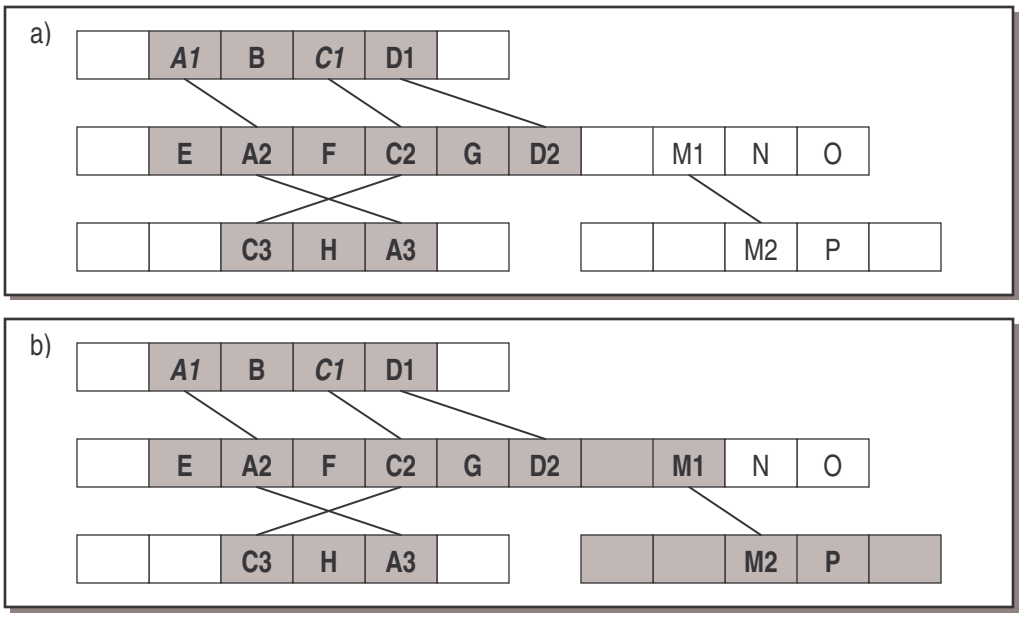

Figure 135 Story graph a) of the actual story, b) as detected by the tracking method with optimistic core extension scheme

This issue may lead to multiple stories combined into a single story board. In the extreme case, the entire source video sequence may be returned in response to the given query, which is certainly undesirable.

In order to alleviate this problem, one can use the *pessimistic expansion scheme*. In this scheme, only shots that occur in other segments of the story are added to the core.

When a new story segment is detected, all shots in the segment are examined with respect to their matching shots. The shots which match at least one shot found in some previously discovered segment are added to the core. Although in this scheme much fewer shots are added to the core, the ones which are added are less likely to be irrelevant to the news story. Intuitively, if a shot is used more than once in the story, it is likely to be a part of the recorded footage used by the news station. In addition, the potential for inclusions of shots from other stories is considerably diminished. If this were to happen, two segments of two different stories would have to be aired adjacent to each other at least twice.

The following example (Figure 136) demonstrates the advantage of the pessimistic core expansion. The story graphs show that even though the greedy segment building strategy erroneously classified *M1* as part of the story *A-H*, the shot was not included in the core because no occurrence thereof was found in other segments of the story. Thanks to that, the following segment of the story *M-P* was not misclassified as belonging to story *A-H*. At the same time, shot *D1*, whose repetition was found in the new segment, was incorporated into the core. Hence, subsequent segments may be detected based on the repetition of that shot.

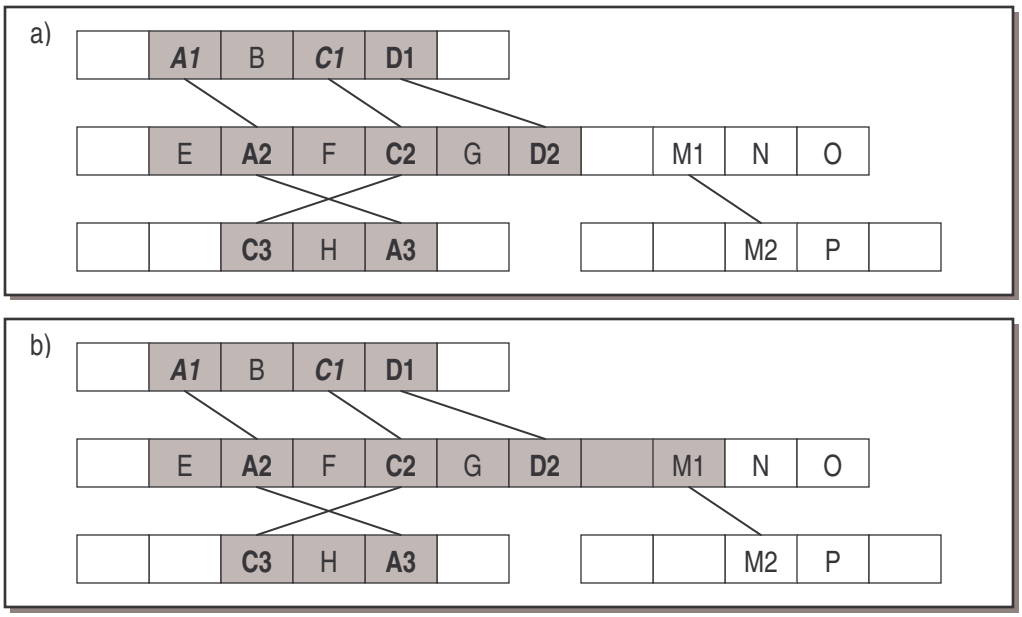

Figure 136 Story graph a) of the actual story, b) as detected by the tracking method with pessimistic core extension scheme

In summary, the pessimistic core expansion scheme may, at times, exclude shots from the core that could be valuable in detection of subsequent story episodes. However, it is generally more accurate than the optimistic scheme, and will be used in our algorithms.

4.4.3 Evaluation

This section discusses the performance of out story tracking technique on a typical news broadcast. For this purpose, we selected an 18-hour video sequence of CNN News channel, and chose a story regarding Michael Jackson's arrest for evaluation purposes. This story was deemed the most interesting during the whole broadcasts, as it is reported over many hours and contains some new developments, as well as discusses different aspects of the events. The story consists of 16 segments of various lengths. The shortest of them is only 30 seconds long, and consists of merely 3 shots, while the longest lasts almost 10 minutes, and contains a large number of shots. The entire story contains 17 repeating shots, some of which occur as many as 8 times in

different episodes, while others are shown only twice. The whole 18-hour broadcast was viewed by a human observer, and all segments of the story were manually detected to establish the ground truth for the experiment. The resulting story graph is depicted in Figure 137, where story segments are represented as schematic film strips, and repeated shots are shown as keyframes.

Figure 137 Graph of the entire story used in the experiment

To test our algorithm we chose a set of three different queries, each corresponding to a single segment of the story (see Table 13).

Segment	Segment Duration	Query Size
3	0:35	1
5	0:21	3
6	4:22	6

Table 13 Experimental queries

Each query was used as an input for the story tracking algorithm with liberal segment building and pessimistic core expansion. The algorithm, as described earlier, is controlled by two parameters:

1. Neighborhood size, which determines the size of a story segment build around each repetition of a core shots.
2. Co-occurrence threshold, which governs the process of core expansion.

We conducted several experiments using different values for these two parameters. Since the algorithm executes iteratively by first detecting new story segments and then expanding the core, we ran it until ten iteration steps were performed, or until the story converged, and no more episodes could be added.

We first observed that for the vast majority of the input parameter settings, the story tracking process did not cease until the entire video sequence was included in the story, thus yielding 100% recall, and unacceptably low precision. Therefore, we decided to examine the values of recall and precision after different number of iterations. Figure 138 and Figure 139 depict recall and precision achieved by the algorithm starting with query 3. In the graphs, iteration 0 corresponds to the initial query, and each subsequent value represents the recall and precision for the story detected by the algorithm after *n* core expansions.

The graphs indicate that recall grows gradually with the number of iterations as the algorithm discovers new episodes of the story and extends the existing ones. Conversely, precision decreases in the same manner due to inclusion of irrelevant

shots. For a given neighborhood size, the rate of the increase in recall and decrease in precision depends on the co-occurrence threshold. A similar pattern was observed for the other queries (5 and 6).

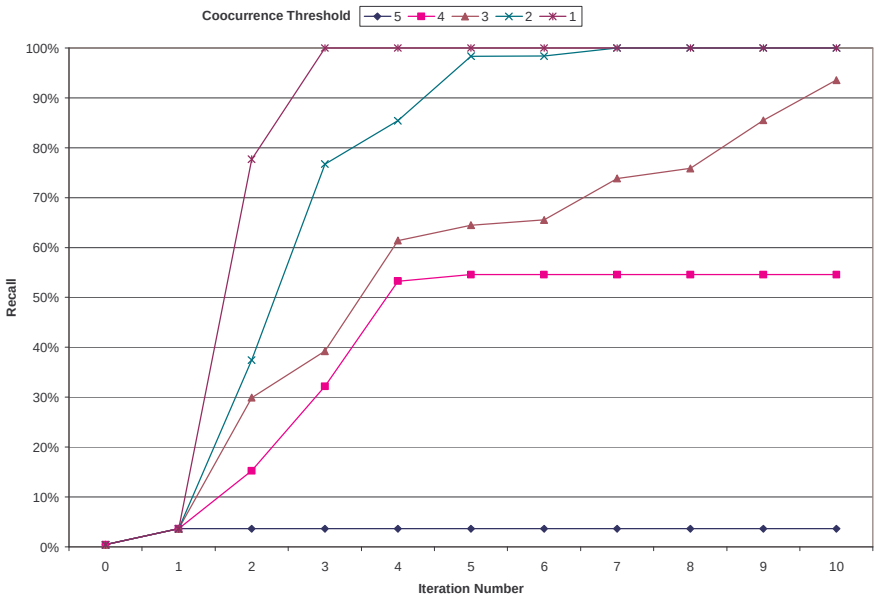

Figure 138 Story tracking recall after different number of iteration for query 3 with neighborhood size of 2.0 minutes

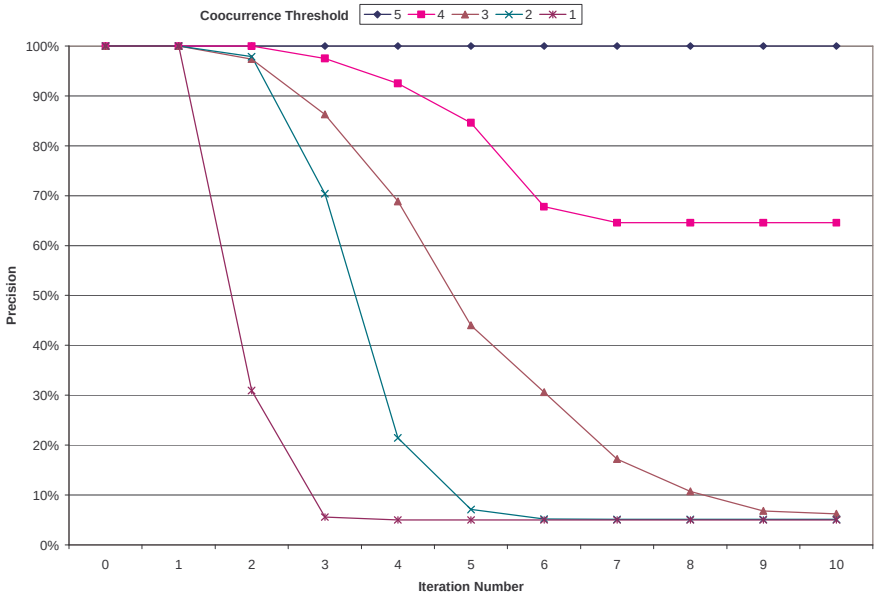

Figure 139 Story tracking precision after different number of iteration for query 3 with neighborhood size of 2.0 minutes

These observations have the following significance. Although the algorithm usually does not stop on its own accord until the entire broadcast is contained in the story, it is possible to obtain the desired level of accuracy by limiting the number of iterations the algorithm executes. For instance, the story tracking process yields 75% recall and 70% precision after 3 iterations with neighborhood size of 2 minutes, and co-occurrence threshold of 2.

In order to establish the optimal set of parameters for the algorithm, we aggregate recall and precision into a single performance estimator – utility function. The utility function – first introduced in Chapter 2 - is a weighted sum of recall and precision as presented in (45), where α is a coefficient regulating the relative importance of the two measures.

$$utility = \alpha \cdot recall + (1 - \alpha) \cdot precision \tag{45}$$

For this evaluation we chose $\alpha = 0.5$, thus assigning equal importance to recall and precision. Figure 140 presents a graph of the utility function for different number of iterations of story tracking with query 3 using neighborhood size of 2 minutes. The function exhibits a characteristic pattern. Its values first increase to reach a maximum at a certain small number of iterations. Subsequently, the values diminish and stabilize around the starting level. The same pattern is also present for detection starting with queries 5 and 6, as well as for different values of the neighborhood size. This allows us to conclude that the story tracking algorithm attains highest accuracy after a small number of iterations, during which the gain in recall outweighs the loss of precision. Afterwards, precision drops quickly decreasing the overall performance.

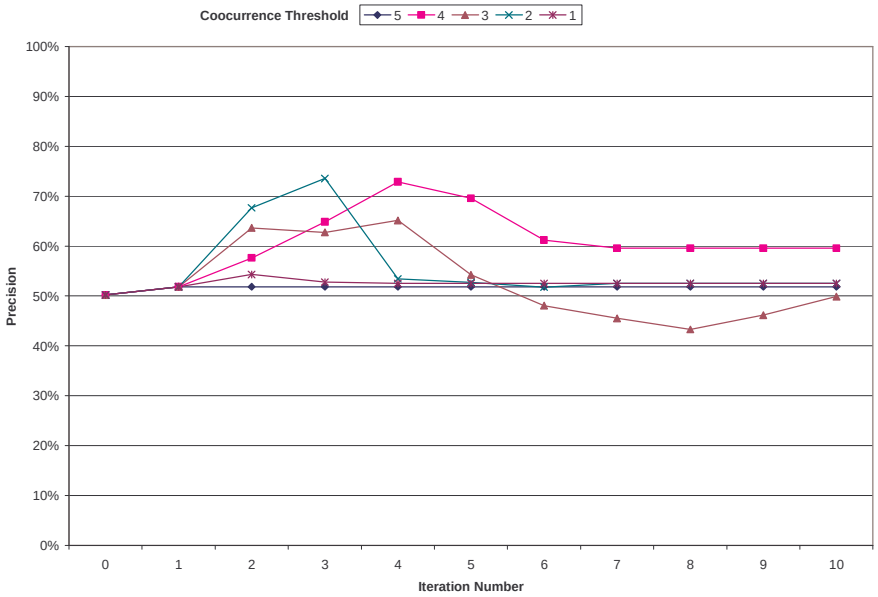

Figure 140 Story tracking utility after different number of iteration for query 3 with neighborhood size of 2.0 minutes

Analysis of the utility function for all queries over the whole range of parameters shows that the algorithm achieves the best performance using a neighborhood of 2.0 minutes. The optimal value of the co-occurrence threshold was 2 for query 3, and 3 for queries 5 and 6. The graphs of recall, precision and utility function for all three queries with their respective optimal parameters are shown in Figure 141 and Figure 142.

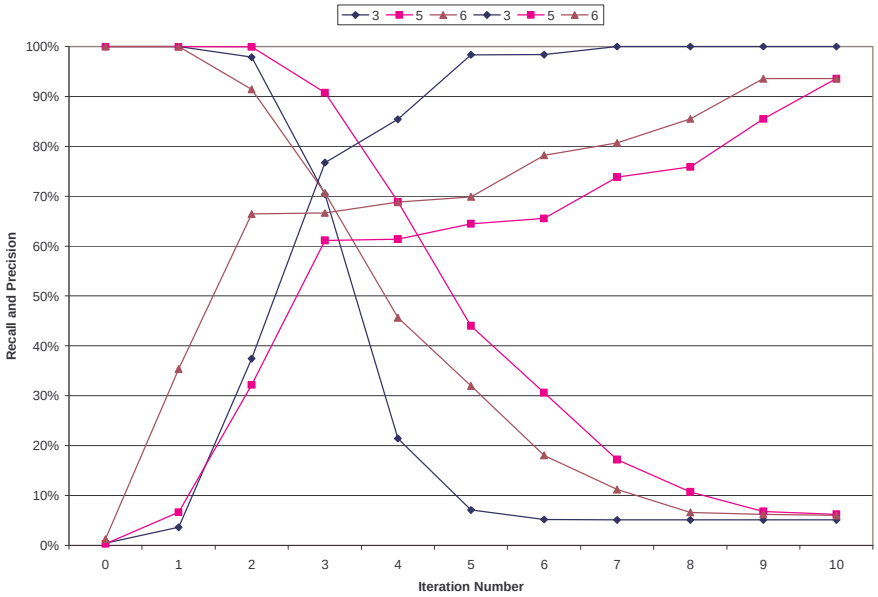

Figure 141 Recall and precision curves for all three queries with optimal parameters

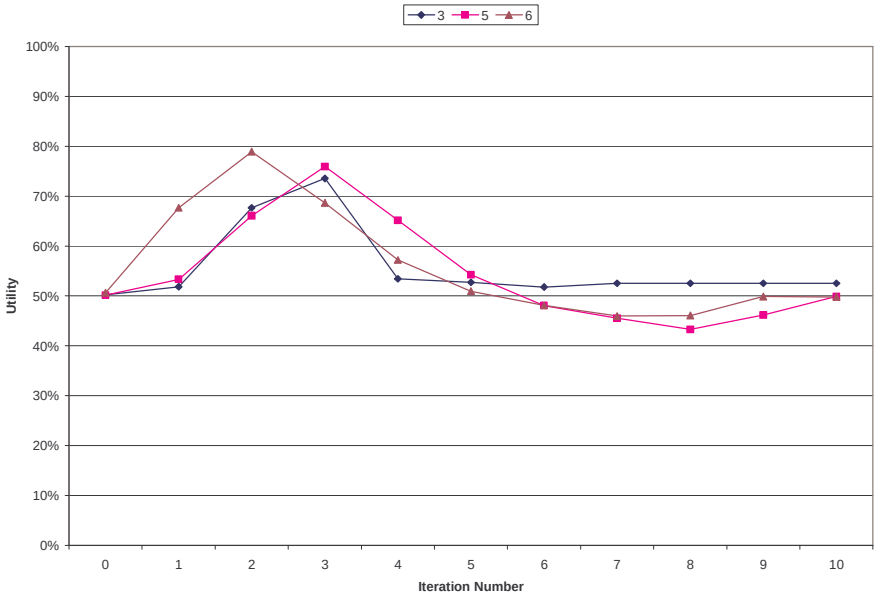

Figure 142 Utility function for all three queries with optimal parameters

The graphs in Figure 141 and Figure 142 demonstrate that our story tracking algorithm described in this section is capable of achieving recall and precision in the range of 70% to 80%, which is an excellent result compared to the values of recall and precision for the original query.

4.4.4 Summary and Discussion

In this section, we presented a novel story tracking technique which relies on shot repetition patterns to identify story segments related to the user's query. We showed that repeated shot detection combined with episode creation and story core expansion may be used to track stories with very good accuracy. The experiments we performed show that the algorithm achieves the best performance after a small number of iterations. With the increase in the number of iterations, the algorithm detects larger portions of the story and increasing recall, while simultaneously including some irrelevant shots, which causes a gradual decrease in precision. Consequently, the number of iterations could be used as a parameter which controls the trade-off between recall and precision. This would allow the user to adjust the algorithm performance to their individual preferences.

Throughout this section, we also identified a number of challenges in our automated story tracking method, which were confirmed by the experiments and will now be discussed in more detail. We showed earlier that the set of all repetitions of the query shots is an uninteresting result of a story tracking algorithm. Thus, we proposed two strategies (liberal and conservative) of discovering story segments around the repeated shots. Both approaches have potential weaknesses. The liberal strategy builds an episode of all shots within a certain time span, whose size must be determined *a priori*. If the time span is too small, very few shots are incorporated into the story. On the contrary, if it is large, the algorithm may include irrelevant shots (from commercials, or other stories) into the story being built. Irrelevant shots may also be included if the core shot is not centered in the episode. The conservative strategy attempts to alleviate this problem by adding to the story only the shots which lie between two or more repeated core shots. This approach eliminates the need for the *a priori* parameter determining the episode length. However, it requires that at least two core shots be repeated in every segment of the story in order to effectively detect segment boundaries.

In the experiments described in section 4.4.3, the liberal story building strategy was used. Even though it was shown that the optimal performance was achieved with neighborhood size of two minutes, this neighborhood substantially exceeds the size of several segments of the actual story. For instance, segments 3 and 5 are only about 30 seconds in length. Consequently, when either of the shots was detected, the story was extended to include several irrelevant shots. We also observed that a few of the story segments were surrounded by commercial blocks. Although it is clear that commercials are not part of the story being tracked, the algorithm proceeds to include them as long as they are within the neighborhood size of some core shot.

Earlier in this section, we demonstrated that in order to effectively track stories over time, we must allow the story core to evolve and include additional shots as they become available. This proposition was confirmed by the experimental results. When we attempted to track the story using co-occurrence threshold of 5.0 (see Figure 138), no shots were added to the core. As a result, the story did not expand past the repetitions of the original query shot, and the recall remained at a mere 5%.

To alleviate this problem, we devised two story core expansion schemes. In the first, we optimistically add all shots in the episode to the core. This scheme would be optimal if the segment building strategies were perfect. Because they are not, some number of irrelevant shots from other stories may become part of the story segment. If such shots are also added to the core, the tracking method will inadvertently combine multiple stories into one. This problem is rectified to a large extent by the pessimistic expansion scheme, which requires that only shots repeated in other segments of the story be added to the core. However, it is still possible for the same commercial to be aired next to two or more different segments of the same story. If this occurs, the shots belonging to the commercial may be added to the story and the core. Since commercials repeat throughout the news broadcast independently of any particular story, including any of the commercial shots in the core may lead to the story growing far beyond its true boundaries, and even include the entire broadcast. This occurred in most of the experiments conducted in section 4.4.3. For example, in

the second iteration of story tracking with query 3, neighborhood of two minutes, and co-occurrence threshold equal to 2, several shots from a BWM commercial were added to the core. In the subsequent iteration a number of new "story segments" were detected around the repetitions of the same commercial.

All of the deficiencies discussed above may manifest themselves, even if the shot match relation given as the input to the algorithm is perfectly accurate. In the presence of imperfections in shot matching, some of the problems become more acute. Specifically, the shot matching techniques introduced in Chapter 3 do not distinguish well between different anchor or studio shots. Shots of these types tend to be visually very similar, and hence are often regarded matching, even if they are not strictly repetitions of one another. This poses a problem because usually during a single news program, one anchor person reports several different stories, and thus anchor shots are present in story segments of multiple unrelated stories. If any of them make their way to the story core, then the tracking algorithm will quickly incorporate all other segments reported by the same anchor into a single story. As a result, the story may grow out of control and is very likely to include the entire news broadcast. This problem was exemplified by the second iteration of the same experiment with query 3. In this iteration a studio shot, which accidentally matched two another studio shot already in the story, was included in the core. Consequently, in the next iteration the algorithm added two unrelated segments, which in turn contributed to the uncontrollable growth of the story.

Clearly, the issues discussed in this section hamper the story tracking performance. In particular, their impact prevents the algorithm from converging before the story comprises the entire broadcast. In the course of working with the story tracking algorithm, we discovered that some of these problems could be alleviated if the algorithm had information about classes of news shots involved in the story building. In the next section, we will consider a number of improvements that could be made by classifying news shots.

4.5 Shot Classification

In the previous section, we introduced a story tracking algorithm which relied solely on matching shot information. We also demonstrated that the algorithm performs reasonably well, but pointed out certain shortcomings, and suggested that some of them may be addressed by making use of news shot classification. Most of the shots in television news broadcasts can be categorized into a small number of classes, such as anchor persons, studio shots, commercials, news content, etc. Thus, we could introduce a *shot classification function* γ, which provides a label for each shot in the original video sequence. The function $\gamma : V \to \Psi$ assigns a shot class from the set Ψ to every shot in *V*. The function γ could be used as an additional input parameter of the story tracking algorithm.

In this section, we explore improvements which can be made to the story tracking algorithm by utilizing the category labels assigned to every shot. We consider a classification scheme which groups shots into four categories: anchor person or studio, commercial or promo, logo or graphics, and news content. Hence, the shot class set Ψ may be defined as follows:

$$\Psi = \{anchor, commercial, graphics, news\}$$

We also suggest methods of automatic shot classification for the classes *anchor* and *commercial* based on shot repetition pattern.

4.5.1 Improvements from Shot Classification

The problems with the basic story tracking algorithm concerned two phases of the tracking process: segment building and core expansion. Due to imprecision of these two steps, irrelevant shots could be included in the story or even incorporated into the story core. In the following two sections, we discuss ways of rectifying these problems by shot classification.

Segment Boundary Detection

The main deficiency of the segment building used by our algorithm the assumption that all segments are of approximately the same size and centered around the repeated shots. Clearly, this simple approach cannot fit all types of segments and all locations of repeated shots within the segments. As a result, some relevant shots are not included in the story, while certain irrelevant shots may be. Although this problem cannot be entirely eliminated by using shot classification, some improvement can be made.

A closer examination of the experimental results obtained in section 4.4.3 indicates that news segments regarding one story very rarely span a commercial block. Most often, the anchor person finishes reporting the story, and only then the station goes to a commercial break. Therefore, detection of a commercial shot in the neighborhood of a repeated shot is a very good indication that the story segment ends before the commercial.

Hence, the following algorithm may be used to build story segments. Start with the last shot preceding the repeated shot. For every preceding shot within the experimentally determined distance of the repeated shot check the shot class. If the shot is labeled as a commercial, do not add it to the story and stop building the segment in this direction. Repeat the process for the shots succeeding the repeated shot. This process is depicted on the block diagram in Figure 131.

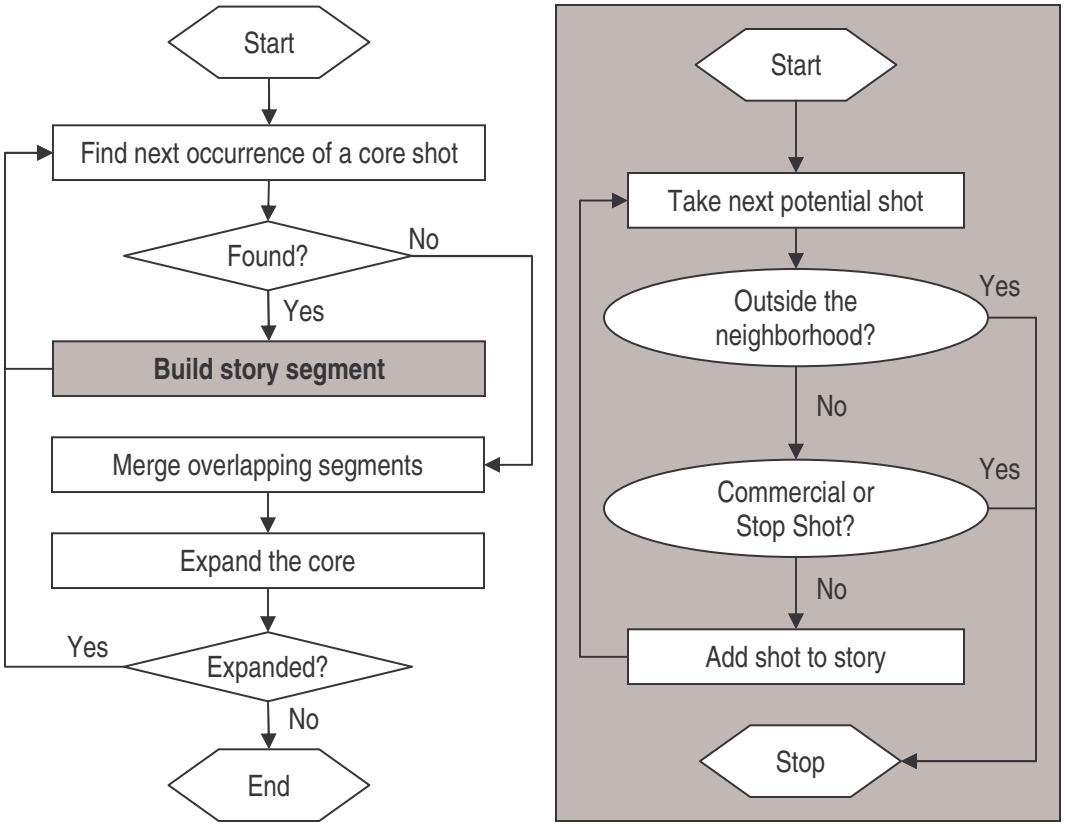

Figure 143 Block diagram of the segment building algorithm

Core Expansion

Errors in core expansion have more severe consequences than imperfections in segment building. As discussed earlier, inclusion of irrelevant shots in the query may result in an uncontrollable story growth.

If every shot in the news segment is labeled according to its class, we may elect to include in the core only the news content shots. This way we eliminate the issue of placing commercials in the core and finding their repetitions at random places in the broadcast. Simultaneously, we also filter out anchor shots, which may be matched to other shots by an imperfect shot matching technique.

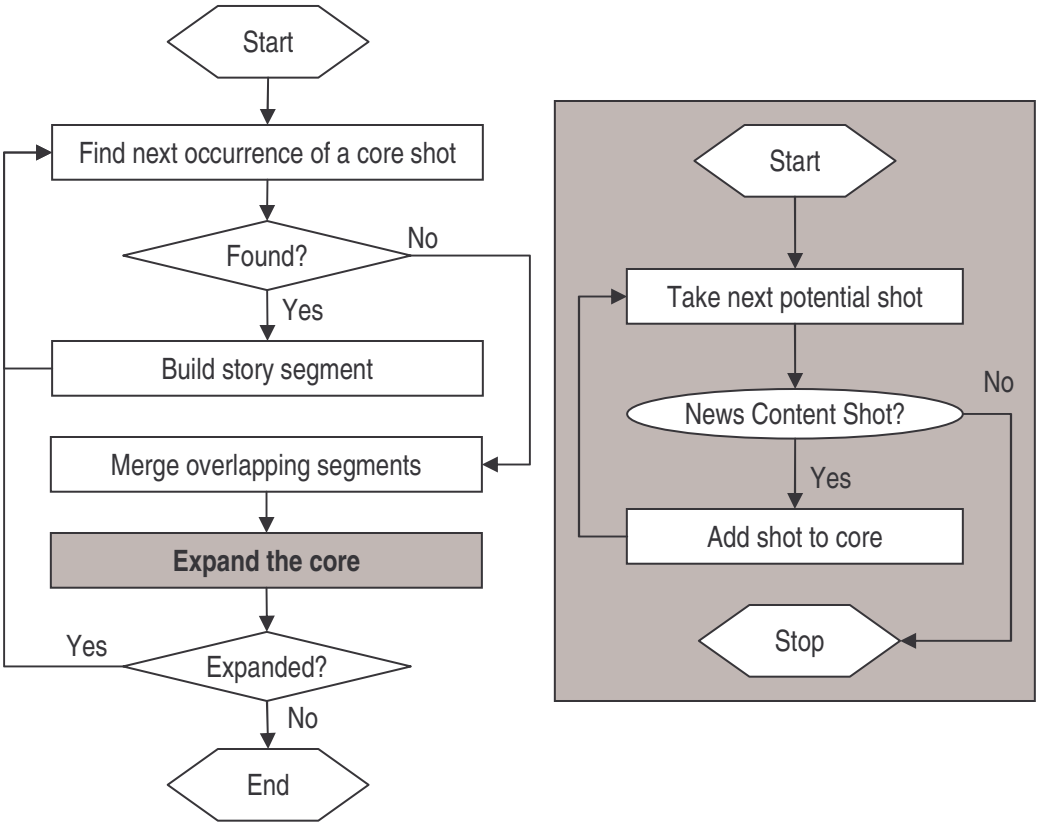

Figure 144 Block diagram of the core expansion algorithm

If we combine this method with the pessimistic core expansion scheme, we could virtually eradicate the possibility of including any irrelevant shots in the core.

4.5.2 Automatic News Shot Classification

In the previous section, we proposed a method by which correct shot classification can be used to improve the results of our story tracking algorithm. In this section, we suggest techniques of automatic shot classification based primarily on repetition patterns typical for different types of shots. We propose algorithms for automatic recognition of three classes of shots common in video news: commercials and promos, anchor and studio shots, and news content shots. In addition, we identify the most commonly repeating shots, named *stop shots*.

Stop Shots

In the field of textual information retrieval, words that occur very frequently contain little semantic information and are called *stop words*. Similarly, in video news broadcasts certain shots occur so often that they provide nothing relevant to news stories, and could analogically be called *stop shots*. Examples of such shots may be sequences of black frames between fade-outs and fade-ins, or television station logos which appear between news segments. Therefore, it is beneficial to identify stop shots and make sure they are never added to a story.

The classification technique for this type of shots is quite straightforward. Given a shot match partition on the source video sequence V, one can simply label a certain percentage of the most frequently repeating shots as stop shots. The specific percentage value may be chosen experimentally depending on the video source.

In order to verify this proposition, we calculated a histogram of shot repetition in the 18-hour CNN News broadcast used in other experiments in this chapter. The resulting histogram is depicted in Figure 145. The graph shows that indeed the broadcast contains a small percentage of shots which repeat frequently, while the majority of shots occur only a few times. More detailed investigation revealed that the most frequently occurring shots were anchor persons and certain promos. However, we also observed that a large number of anchor person shots, commercials, and studio shots, appeared were matched to less than 10 other shots. Considering that certain news content shots belonging to our story were repeated a similar number of times, we found it difficult to establish an optimal frequency threshold separating important shots from potential stop shots.

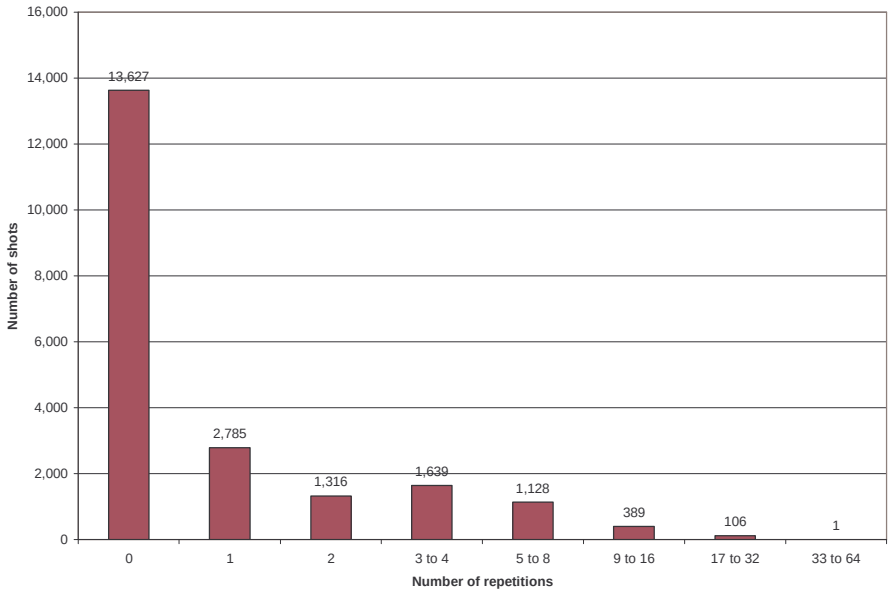

Figure 145 Shot repetition histogram in an 18-hour CNN News broadcast

Nonetheless, we chose to perform a story tracking experiment in which stop shots were excluded from the story core expansion. Having precise knowledge of the story shot repetition patterns – none of the shots repeated more than 10 times – we selected the stop-shot frequency threshold equal to 10. Contrary to our initial expectations, the removal of the most frequent shots did not improve the story tracking performance. This result may be better understood, if one remembers that the majority of anchor person and commercial shots were not eliminated by this frequency threshold. Apparently their presence in close proximity to the actual story segments was sufficient to introduce a large number of false segments into the story.

This experiment shows that in order to substantially improve the story tracking accuracy, frequency-based classification is insufficient and classification methods for the specific types of news shots are needed. In the next sections we suggest such automated methods for detection of commercials and promos, as well as anchor person and studio shots.

Commercials and Promos

Commercials and promos have a number of characteristics that help distinguish them from the rest of the news broadcast. First, they repeat regularly at various times during the broadcast (see Figure 146). They tend to consist of several shots, and usually are either 30 or 60 seconds long. In addition, they occur in blocks of several commercials at a time. These characteristics could be used to create a classification method which could use multiple criteria to correctly identify commercials and promos. Such a technique could detect sequences of multiple shots, which appear several times during the broadcast, especially in close proximity to other sequences of shots of similar repetition pattern. Commercial classification is beyond the scope of our work, but shows potential for improvement of story tracking performance.

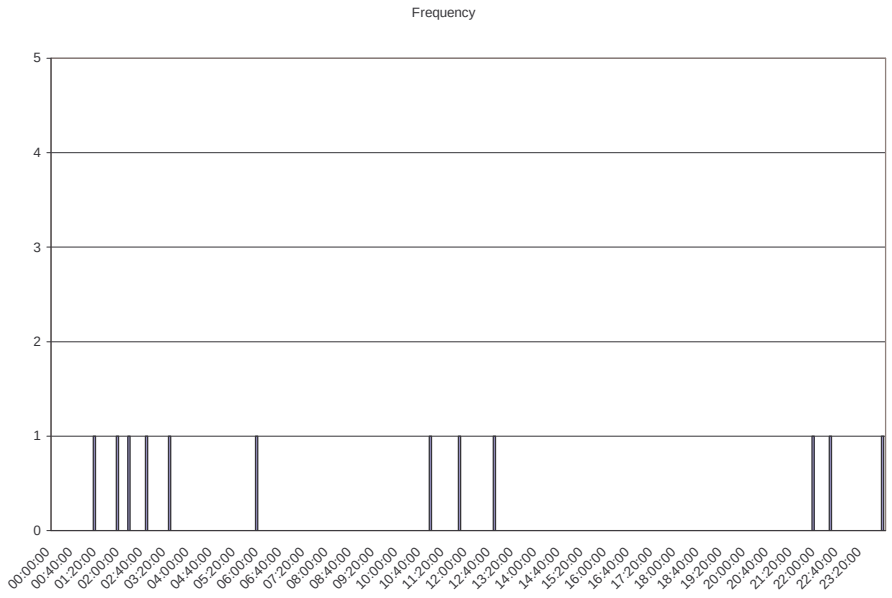

Figure 146 Typical repetition pattern of a commercial shot

Anchors and Studio Shots

In this section, we propose an automatic method of identifying anchor and studio shots. By analyzing typical news shows, we observed that during a single show, anchor persons appear multiple times to relate different stories. Although in the strict sense those shots are not repetitions of one another, they are visually very similar, and

are frequently reported as matching by an automatic shot matching algorithm. This shortcoming of the shot matching mechanism may be used to our advantage.

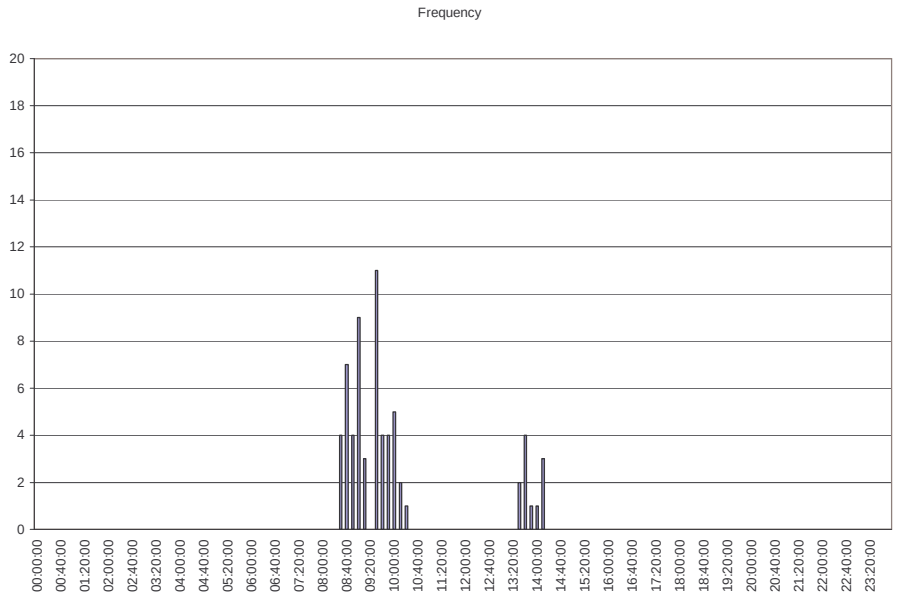

Figure 147 Typical repetition patter for an anchor person shot

Figure 147 shows a typical distribution of shots matching a sample anchor person. In this experiment, a 60-frame portion of an anchor person shot was selected and all its repetitions were detected and grouped within 10-minute time spans. It is clear that in a time window of 08:30:00 to 10:20:00, a large number of occurrences is found. This corresponds to a single news show lasting about 2 hours. On the other hand, no matching sequences are found any other time during the day, except for some accidental matches to different anchor person shots in another news show. Thus, unlike commercials and promos, which occur frequently during the entire broadcast, anchor and studio shots tend to appear in close proximity to one another.

Therefore, frequent local repetition may be used as good anchor classifier. If a certain shot matches a large number of other shots within a certain distance corresponding to the length of a typical news show, but does not match many shots outside of this range, it may be labeled as an anchor shot. The characteristic length of a news show may be established experimentally depending on the news source.

In addition, anchor shots usually contain very little motion. As a result, the video features of their individual frames remain very stable across the whole shot. This fact may be used to augment anchor classification. To this end, one can measure the auto-similarity of every shot. Naturally, if one compares any video sequence to itself by measuring the difference in features of the corresponding frames, one will find all sequences matching. This is not very helpful in establishing auto-similarity. On the other hand, if one can consider the number of non-corresponding frames that match, one will notice that relatively still shots contain a large number of them, while sequences with substantial motion do not. Thus, the total number of matching frames could be calculated by comparing all pairs of frames in the shot. This process, however, would be very time consuming. Alternatively, we could use quantized color moments (see section 3.6.2). The auto-similarity of a given shot may be quickly assessed by counting the total number of frames which share a single hyper-cube of the quantized color moment space.

The combination of the two criteria introduced above, i.e. high frequency of local repetition and high auto-similarity, shows promise as an anchor shot classification technique. Implementation and evaluation of this approach are beyond the scope of this research, and could constitute an interesting extension of our work.

News Content Shots

Due to a great variety of video footage used by news stations as visual clues for reported stories, it is difficult to provide a direct classification technique for news content shots. Instead, we will assume that all shots not categorized into other shot classes will be considered news content shots. This approach is not perfect, but it substantially limits the number of shots used for story core expansion, and thus reduces the odds of uncontrollable story growth.

4.5.3 Summary

In this section, we proposed news shot classification as a source of improvement for the story tracking algorithm. We explained how shot classification could be used to address problems in our technique discovered in earlier sections, and suggested how this information can be incorporated into the algorithm. Finally, we proposed methods of automatic shot classification based on shot repetition patterns and basic visual features of typical news shots.

4.6 Story Presentation

The problem of story presentation may be regarded very broadly, as an effort to devise optimal ways of presenting news stories to the viewers. Traditional video news media, i.e. television stations, are limited by their nature, and can report news only in a linear fashion. With the advent of the Internet, as well as proliferation of various electronic multimedia devices several other models of story presentation are possible. As these new media become more dominant sources of video news, the domain of story presentation is bound to gain more importance. Detailed analysis of the various alternatives of news story presentation using modern media is certainly beyond the scope of this work. Hence, in this section we do not attempt to provide definitive solutions to the general problem of story presentation. Instead, we focus on the issue of presenting the stories resulting from story tracking in television news broadcasts.

As discussed in section 4.1, the problem of story tracking is intricately connected with the issue of story presentation. Once the story has been identified by the story tracking algorithm it must be presented to the user in a manner which facilitates intuitive browsing and viewing. To this end, we propose to create a story view, which comprises a subset of all story shots, and arranges them according to a certain order. In this section we discuss a few alternative story views which follow naturally

from our story tracking technique. A brief exposition of graphical presentation of the story views is also given.

4.6.1 Complete Story View

The story tracking algorithm developed in this chapter returns a story as a set of video shots. A straightforward way of presenting the story to the user is to simply show all the shots in their chronological order. If, in addition, the associated audio is played while the shots are shown, such presentation constitutes the most comprehensive coverage of the story. On the other hand, it may contain a considerable amount of visually redundant information. This could be eliminated by retaining only the first occurrence of any story shot, and removing all of its repetitions. In this case, the information in the audio signal may be difficult to understand due to the removal of the audio associated with the repeated shots.

4.6.2 Visual Content View

Our tracking algorithm detects stories using visual features of the broadcast. It is, therefore, natural and more interesting to focus on the visual aspects of the story. It turns out the story board used by the algorithm lends itself to creation of a story view which focuses on visual aspect of the story.

This approach to story presentation emphasizes the visual content of the story. While video shots of anchor persons and reporters convey important portions of the story in the associated audio, they are of little importance visually. If the user is interested primarily in the visual aspects of the story, it may be beneficial to present to him only the news content shots. To this end we propose to use the story core created by our story tracking algorithm. Thus, we select a story view (see section 4.1) which consists of the set of all shots comprising the core along with a partial or total order on these shots. In this section we discuss two possible shot orderings, which result in two different story core views.

One story view may be constructed by arranging all core shots in the order of their *chronological appearance*. This simple arrangement captures an important aspect of story development. As the news story evolves over time and comprises new events, television news stations obtain new video material and show it to the viewers. Thus, by viewing only the core shots in the order they were shown by the station, the user may deduce the temporal development of the story.

Perhaps the main disadvantage of presenting the story in this fashion is the significant proportion of redundant visual information. The story core, as constructed by our algorithm, contains every repetition of every news content shot in the story. Thus, viewing the entire story core the user would be shown the same shots multiple times. This issue can be rectified by reducing the view to only the unique shots in the core, which may be accomplished by using the equivalence classes in the story board. For every equivalence class a single representative shot may be selected and placed in the view. The choice of the representative shot allows for three alternatives. First, one can simply take the chronologically earliest shot in the class. However, since shots in the class often vary in length and may match only partially, the earliest shot may be very short, and contain little information. A better choice is to use the longest shot in the class. Finally, in order to maximize the visual content, one can construct a video sequence constituting the union of all shots in the class, as depicted in Figure 148.

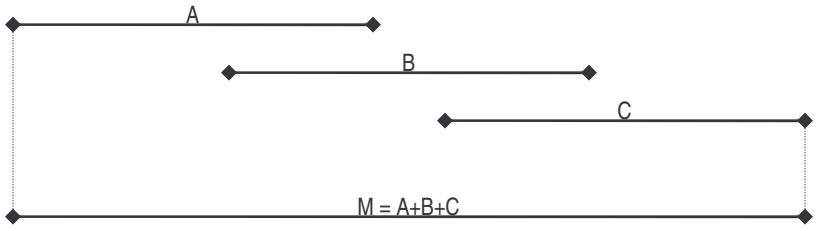

Figure 148 Example of shot merging to obtain maximum of visual content

In the end, after every equivalence class has been reduced to a single shot, the *chronological story view* becomes a linear sequence of unique shots presenting the maximal portion of the visual content of the story in the shortest amount of time.

The chronological story view utilizes temporal relationships between core shots. Certainly other relationships exist and could be exploited for story view creation. Here, we will consider one alternative, which explores the core shot organization within story segments. The news content shots in the story segments are arranged by the director of the news program during the production process. This arrangement may correspond to some logical connections between the shots, and thus presenting the story in a manner most accurately reflecting these connections may be beneficial. We can utilize the story board created by our algorithm to construct such a story view.

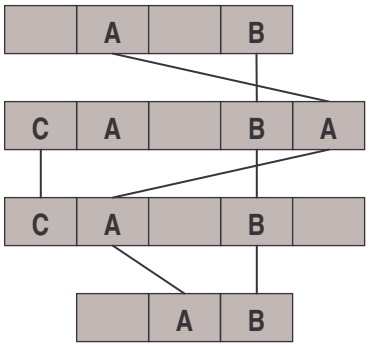

Figure 149 Sample story

To this end we form a co-occurrence matrix *C[M,M]*, where *M* is the number of equivalence classes in the core. Every entry c_{ij} of this matrix is the total number of times a shot from class *i* preceded a shot from class *j* in any segment of the story. Consider the story depicted in Figure 149, which has a core whose shots belong to three equivalence classes: *A*, *B*, and *C*. The co-occurrence matrix for this story is shown in Table 14.

	A	B	C	Total
A	0	4	0	4
B	1	0	0	1
C	3	2	0	5

Table 14 Sample co-occurrence matrix

Given the co-occurrence matrix, one can construct a partial order on the core shots in the following way. For every shot *x* a co-occurrence score *cs(x)* can be calculated as the sum of all entries in the corresponding row of the matrix. The partial is the set of all pairs of shots (*x,y*) with different scores, such that *cs(x) > cs(y)*. In the example above, the resulting partial order becomes $\{\langle C, A\rangle, \langle C, B\rangle, \langle A, B\rangle\}$.

In this simple example the partial order is also a total order, but in general several shots may have the same co-occurrence score, and therefore cannot be ordered with respect to each other. However, a total order may always be constructed by considering the chronology of the shots in question. If multiple shots share the same score, they can be ordered according to the time of their first occurrence in the news broadcast.

In summary, the story core, which comprises the visual content of the story, may be presented in either a chronological order, or arranged according to the order of shot appearance in the story segments.

4.6.3 Graphical Story Presentation

Regardless of the choice of story view and the method used to create it, the story view must be presented to the user in a form that facilitates intuitive browsing and viewing of the story. In this section we briefly describe a concept of a graphical story presentation interface which serves this purpose.

In order to graphically display the complete view of the story one can use a representation visually similar to the story graph used throughout this chapter. All segments of the story should be arranged chronologically, and displayed on separate lines, as shown in Figure 150. The shots within individual segments can be depicted by icons showing corresponding keyframes. To avoid displaying a very large number of icons, especially for long segments, only the keyframes representing the core shots could be used.

Figure 150 Sample user interface for a complete story view with two segments

Presented with this interface, the user could click on the displayed icons to play the corresponding core shots. Alternatively, the entire segment could be played by clicking on the area between the keyframes.

The visual story content view is not inherently divided into segments. Rather, the view presents the story core in either a linear or non-linear fashion. For the linear view, an interface resembling a film strip could be adopted, as shown in Figure 151. The user's interaction with this interface could be analogical to the one used for complete story view.

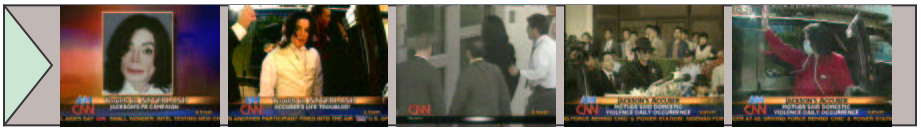

Figure 151 Sample user interface for a linear story core view

The non-linear view of the story core requires a different representation. Since only certain pairs of core shots are strictly ordered, the story view could be shown as a graph, in which vertices correspond to shots and edges represent ordering, as in Figure 152. This interface requires the user to select individual shots for playback one at a time. Alternatively, the user could mark a path in the graph and have all shots in this path played sequentially.

Figure 152 Sample user interface for a story with two segments

All of the user interfaces presented could be used to efficiently and intuitively view the results of story tracking. However, since the development of a graphical user interface for story viewing was not the focus of this work, we restricted ourselves to implementing a simple linear view of either the complete story or the story core. The view is shown to the user in the form of a list of video clips described by their start and end time, as shown in Figure 153. Each such list item may be clicked in order to play the video clip in the associated video window. In addition, the user can randomly access any portion of the video clips by entering the time at which they would like to begin playback. This simple interface has proven very useful in the development and testing of our story tracking algorithm.

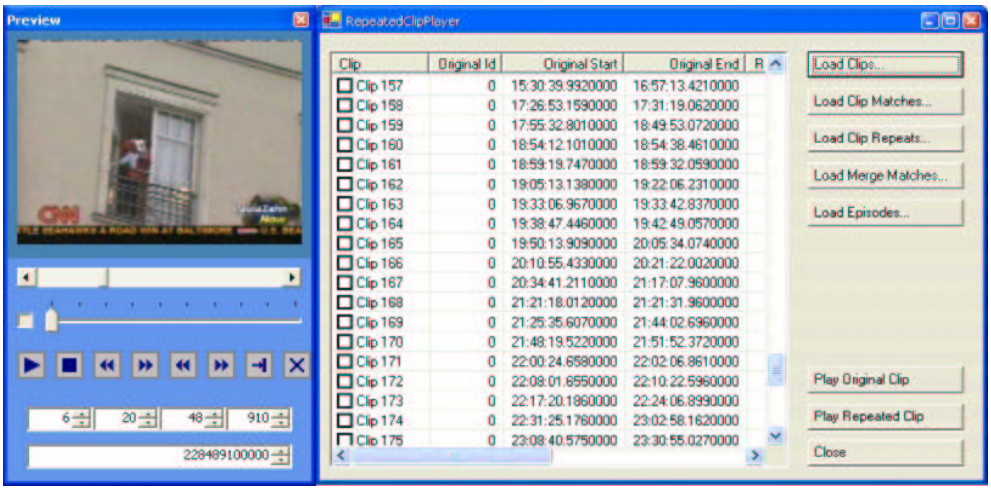

Figure 153 Simple list representation of a complete story view

Summary

In this section we briefly discussed the problem of story presentation. Far from attempting to provide definitive solutions for the general question of optimal presentation of news stories in modern media, we were primarily interested in the methods of organizing and displaying the results of our story tracking technique. To this end we described two story views which follow most naturally from the tracking algorithm. In the complete story view all shots in the story are arranged in their chronological order and presented along with the corresponding audio, whereas in the visual content view emphasizes the visual aspect of the story, by displaying only the story core. At the end, we gave a conceptual overview of a graphical user interface which may be used to present both story views to the user in an interactive fashion.

4.7 Conclusions

This chapter discusses the design of a story tracking algorithm based on repetition of video footage in television news broadcasts. In this chapter we first provided the definition of the problem of story tracking in video, as the detection of all shots relevant to a particular news story. We designed and implemented an algorithm which creates a story from a set of relevant query shots. The story is built gradually,

one segment at a time, as the video content becomes available. The algorithm detects story segments by identifying new occurrences of shots already in the story.

To evaluate this technique in real-world conditions we performed a story tracking experiment on a CNN News broadcast. The results showed that our fully automated story tracking algorithm can achieve good performance (recall and precision around 75%), even using imperfect input data. The experiments also proved that there are some aspects our approach which could be further improved. We observed that the repetitions of anchor person shots and commercials confuse our algorithm and cause it to expand the story past its actual boundaries. Therefore, we postulated that accurate shot classification could be used to enhance our story tracking method, and showed how the method could be extended to incorporate the shot classification information. Finally, we suggested methods of automatic labeling of shots as commercials and anchor persons based on their characteristic repetition patterns.

The research discussed in this chapter demonstrates that repetitions in the visual content of video news broadcasts can be used to successfully track stories. This tracking method constitutes a viable alternative to the textual topic tracking techniques. The two approaches could be combined in order to improve the overall story tracking performance.

Chapter 5

Conclusions

This dissertation considered the problem of inadequate access to video information, particularly in the domain of video news broadcasts. In our research, we addressed one of the main aspects of the problem, which is the tracking of news story development over time. We designed and implemented an effective story tracking technique based on visual characteristics of television news broadcasts. Our method is complementary to textual topic tracking techniques, and may be used in conjunction with them to improve the overall performance. The story tracking method we developed comprises three major components, which required us to tackle the corresponding challenges of effective temporal video segmentation, fast and accurate video sequence repetition detection, and story tracking based on detected repetitions.

Temporal Segmentation

An analysis of typical television news broadcasts showed that effective story tracking requires precise detection and effective matching of very short video sequences. After evaluation of some simple shot detection methods and finding their performance insufficient, we designed a more advanced technique based on mathematical models of the main types of video transitions. We chose the three primary color moments – mean, standard deviation, and skew – as video frame

features, and used this representation to create a temporal segmentation algorithm. The algorithm detects cuts, fades, and dissolves by identifying their characteristic patterns in the time series of the color moments. The algorithm was tested on a video sequence obtained from a typical broadcast of the CNN News channel, and compared with other techniques using less compact video features. Our method achieved similar performance in cut and fade detection, but outperformed the other techniques in the identification of dissolves, yielding a 15% improvement in precision and recall for this task.

Repeated Sequence Detection

In order to track news stories in live video broadcast using repeated video footage, we developed a real-time video sequence matching algorithm. Considering that video clips reused by new stations are often very short and their length is adjusted to the demands of live television, we introduced a number of video sequence similarity metrics which can deal with partial sequence repetition. We examined the advantages and disadvantages of using different metrics for detection of repetitions in the news video stream. We demonstrated that direct calculation of partial similarity between all potential sequences in a live video stream is not viable on state-of-the-art commodity hardware. Consequently, we adopted a heuristic filtering technique based on quantization and hashing of the frame color moments, which substantially reduces the average time complexity of repeated sequence detection, and allows the detection to be performed in real time. We analyzed the effects of color moment quantization on video sequence similarity, and introduced a sequence similarity measure based on equality of quantized color moments between video frames regardless of their temporal ordering. We showed that this measure is a good approximation of the actual video sequence similarity, and used it to filter out dissimilar video sequences without performing direct comparison. Experimental evaluation showed that our repeated sequence detection technique with heuristic filtering successfully detected partially repeated video sequences achieving very high recall and precision.

Story Tracking

Using the results of our temporal segmentation and repeated sequence detection method, we designed and developed a novel story tracking algorithm for television news broadcasts based on repetitions of video footage. Our method builds a set of shots relevant to the news story being tracked, called a *story core*, and identifies new story episodes by detecting occurrences of shots belonging to this set. The story core is allowed to evolve dynamically over time to account for additional video footage as it becomes available. This technique was evaluated on a 24-hour broadcast of CNN News channel and was found to achieve good performance (recall and precision of approximately 75%). We subsequently demonstrated that a number of improvements may be made using the classification of shots into categories typical for video news. Finally, we proposed a set of automated classification methods for commercials and anchor shots using their characteristic repetition patterns.

Future Work

Our work demonstrates that visual characteristics of news video streams may be used to track news stories over time. Nonetheless, there are several research areas which this dissertation does not address, and which certainly deserve further investigation. In the domain of temporal video segmentation, little attention has been devoted to the detection of computer generated transition effects. Due to advances in computer and video production technology, such sophisticated effects are increasingly used by the television news stations. Therefore, an interesting research direction would be the development of effective detection methods for computerized effects in video.

In this dissertation, we focused on detecting repeated footage shown in the full video frame, and obscured only by the on-screen captions at the bottom of the frame. We observed, however, that in certain news programs, video footage is displayed in a smaller window and occupies only part of the screen. Effective methods of recognizing the presence of such windows could improve the story tracking

performance. Hence, the design and implementation of automatic techniques of on-screen window detection constitutes a promising extension to our work.

In Chapter 4, we indicated that information regarding news shot classification could substantially improve the accuracy of our story tracking technique. We believe, therefore, that the area of automatic news shot classification should be further explored. Automatic shot classifiers could be created based on shot repetition patterns, as suggested in this work. Alternatively, other methods could be devised, for instance relying on facial recognition, or speaker identification. Such techniques would not only advance the research in story tracking, but would also be a valuable contribution to the broad domain of video retrieval.

Finally, our work could be a starting point for an entirely new research area of multimodal story tracking. The primary efforts in this area should concentrate on the creation of story tracking algorithms which combine the visual characteristics of the news video stream with the associated textual and audio information. For instance, closed captions included in the news broadcasts could be effectively exploited to improve performance of all three phases of the visual story tracking. Similarly, repeated video footage data could be used to enhance the textual topic tracking techniques. Therefore, the convergence of different modes of news communication should be explored so that the merits of visual, as well as textual and aural information may be fully realized.

Bibliography

1. [Adj98] D. A. Adjeroh, M. C. Lee, and I. King, A Distance Measure for Video Sequence Similarity Matching, In Proceedings of International Workshop on Multimedia Database Management Systems, pages 72-79, Dayton, OH, August 1998.
2. [Aku92] A. Akutsu, Y. Tonomura, H. Hashimoto, Y. Ohba, Video Indexing Using Motion Vectors, Proceedings of Visual Communications and Image Processing, SPIE vol. 1818, 1992.
3. [Ala97] A. M. Alattar, Detecting Fade Regions in Uncompressed Video Sequences, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 1997.
4. [Ala93] A. M. Alattar, Detecting and Compressing Dissolve Regions in Video Sequences with a DVI Multimedia Image Compression Algorithm, IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, 1993.
5. [All98a] J. Allan, V. Laverenko, R. Papka, Event Tracking, CIIR technical report IR-128, April, 1998.
6. [All98b] J. Allan, V. Laverenko, R. Papka, On-line New Event Detection and Tracking, Proceedings of ACM SIGIR 1998, August, 1998.
7. [AsfWeb] J. Assfalg, M. Bertini, C. Colombo, A. Del Bimbo, Solutions in Video Retrieval by Content,

http://delosnoe.iei.pi.cnr.it/activities/researchforum/Brainstorming/PositionStatements/del-bimbo.pdf

8. [Ber00] M. Bertini, A. Del Bimbo, P. Pala, Content Based Annotation and Retrieval of News Videos, International Conference on Multimedia and Expo 2000, 2000.

9. [Bor96] J. S. Boreczky, L. A. Rowe, Comparison of Video Shot Boundary Detection Techniques, Proceedings of Storage and Retrieval for Still Image and Video Databases IV, SPIE Proceedings vol. 2664, January, 1996.

10. [Car99] J. Carbonell, Y. Yang, J. Lafferty, R. Brown, T. Pierce, X. Liu, CMU Report on TDT2: Segmentation Detection and Tracking, Proceedings of the DARPA Broadcast News Workshop, 1999.

11. [Car97] C. Carrick, C. Watters, Automatic Assiociation of News Items, Information Processing & Management 33(5), 1997.

12. [Cha98] S-F. Chang, W. Chen, H. Meng, H. Sundaram, D. Zhong, A Fully Automated Content Based Video Search Engine Supporting Spatio-Temporal Queries, IEEE Transaction on Circuits and Systems for Video Technology, Vol. 8, No. 5, September, 1998.

13. [Chi01] V. Chitkara, Color-based Image Retrieval Using Compact Binary Signatures, Master's Thesis, Department of Computer Science, University of Alberta, 2001.

14. [Chr02] M. G. Christel, A. G. Hauptmann, H. D. Wactlar, T. D. Ng, Collages as Dynamic Summaries for News Video, Proceedings of the Tenth ACM International Conference on Multimedia, December 2002.

15. [Dai95] A. Dailianas, R. B. Allen, P. England, Comparison of Automatic Video Segmentation Algorithms, Proceedings of Integration Issues in Large Commercial Media Delivery Systems, SPIE vol. 2615, October, 1995.

16. [Dun97] M. D. Dunlop, Time, Relevance and Interaction Modeling for Information Retrieval, Proceedings of ACM SIGIR 1997, 1997.

17. [Duy04a] P. Duygulu, M. Y. Chen, A. Hauptmann, Comparison and Combination of Two Novel Commercial Detection Methods, International Conference on Multimedia and Expo (ICME'04), Taipei, Taiwan, June 27-30, 2004.

18. [Duy04b] Duygulu, P., Hauptmann, A., What's News, What's Not? Associating News Video with Words, Proceedings of the 2004 International Conference on Multimedia and Expo (ICME'04), Taipei, Taiwan, June 27-30, 2004.

19. [Fer99] W. A. C. Fernando, C. N. Canagarajah, D. R. Bull, Fade and Dissolve Detection in Uncompressed and Compressed Video Sequences, Proceedings of IEEE International Conference on Image Processing (ICIP) 1999, vol. 3, 1999.

20. [Gar00] U. Gargi, R. Kasturi, S. H. Strayer, Performance Characterization of Video-Shot-Change Detection Methods, IEEE Transaction on Circuits and Systems for Video Technology, vol. 10, no. 1, February, 2000.

21. [Gar98] U. Gargi, R. Kasturi, S. Antani, Performance Characterization and Comparison of Video Indexing Algorithms, Proceedings IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara, CA, June, 1998.

22. [Gar96a] U. Gargi, S. H. Strayer, S. Antani, R. Kasturi, Evaluation of Color Histogram Based Methods in Video Indexing, Technical Report, May, 1996.

23. [Gar96b] U. Gargi, R. Kasturi, An Evaluation of Color Histogram Based Methods in Video Indexing, Proceedings of International Workshop on Image Databases and Multimedia Search, 1996.

24. [Gar95] U. Gargi, S. Oswald, D. Kosiba, S. Devadiga, R. Kasturi, Evaluation of Video Sequence Indexing and Hierarchical Video Indexing, Proceedings of Storage and Retrieval in Image and Video Databases, SPIE, 1995.

25. [Gau99] J. Gauch, S. Gauch, S. Bouix, X. Zhu, Real Time Video Scene Detection and Classification, Information Processing and Management 33, 1999.

26. [Gau98] S. Gauch, J. Gauch, K. M. Pua, The VISION Digital Video Library Project, Encyclopedia of Library and Information Science '98, 1998.

27. [Gau97] S. Gauch, W. Li, J. Gauch, The VISION Digital Video Library, Information Processing and Management 33(4), 1997.

28. [Gau96] S. Gauch, J. Gauch, K. M. Pua, VISION: A Digital Video Library, Digital Libraries '96, Bethesda, MD, USA, 1996.

29. [GauWeb] J. Gauch, Video Authentication: Overview, http://www.ittc.ukans.edu/~jgauch/research/video/vidwatch_overview.html

30. [Hac00] M. S. Hacid, J. Kouloumdjian, A Database Approach for Modeling and Querying Video Data, IEEE Transactions on Knowledge and Data Engineering, 12(5), September 2000.

31. [Ham97] A. Hampapur, A. Gupta, B. Horowitz, C-F. Shu, C. Fuller, J. Bach, M. Gorkani, R. Jain, Virage Video Engine, Proceedings of Storage and Retrieval for Image and Video Databases V, SPIE Proceedings vol. 3022, 1997.

32. [Ham95] A. Hampapur, R. Jain, T. E. Weymouth, Production Model Based Digital Video Segmentation, Multimedia Tools and Applications, 1(1), 1995.

33. [Hau03] A. Hauptmann et.al., Informedia at TRECVID 2003: Analyzing and Searching Broadcast News Video, TREC (VIDEO) Conference, 2003.

34. [Hau98] A. Hauptmann, M. Witbrock, Story Segmentation and Detection of Commercials in Broadcast News Video, Advances in Digital Libraries Conference (ADL'98), Santa Barbara, CA, April 22 - 24, 1998.

35. [Hoa03a] T. C. Hoad, J. Zobel, Video Similarity Detection for Digital Rights Management, Proceedings of 26th Australian Computer Science Conference, Adelaide, Australia, 2003.

36. [Hoa03b] T. C. Hoad, J. Zobel, Fast Video Matching with Signature Alignment, Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval 2003, Berkeley, CA, November 2003.

37. [Hua00] Q. Huang and A. Puri, Multimedia Search and Retrieval: New Concepts, System Implementation and Application. IEEE Transactions on Circuits and Systems for Video Technology, 10(5):679-692, August 2000.

38. [Hua97a] J. Huang, S. R. Kumar, M. Mitra, Combining Supervised Learning with Color Correlograms for Content Based Image Retrieval, ACM Multimedia 1997, 1997.

39. [Hua97b] J. Huang, S. R. Kumar, M. Mitra, Image Indexing Using Color Correlograms, Proceedings of Computer Vision and Pattern Recognition 1997, 1997.

40. [Ide03] I. Ide, H. Mo, N. Katayama, S. Satoh, Topic-Based Inter-Video Structuring of a Large-Scale News Video Corpus, 2003 IEEE International Conference on Multimedia and Expo (ICME2003), vol.3, pp.305-308, Baltimore MD, July 2003.

41. [Ide03] I. Ide, H. Mo, N. Katayama, Threading news video topics, Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval 2003, Berkeley, CA, November 2003.

42. [Ide02] I. Ide, N. Katayama, S. Satoh, Visualizing the Structure of a Large Scale News Video Corpus Based on Topic Segmentation and Tracking, Proceedings of ACM Multimedia 2002 Workshop on Multimedia Information Retrieval, Juan-les-Pins, France, Dec. 2002.

43. [Ide01a] I. Ide, R. Hamada, S. Sakai, H. Tanaka, An Attribute Based News Video Indexing, Proceedings of ACM Multimedia 2001 Workshops -Multimedia Information Retrieval, pp.70-73, Ottawa ON, Canada, Oct. 2001.

44. [Ide01b] I. Ide, K. Yamamoto, R. Hamada, H. Tanaka, An Automatic Video Indexing Method Based on Shot Classification, Systems and Computers in Japan, vol.32, no.9, pp.32-41, Aug. 2001.

45. [Ide00] I. Ide, R. Hamada, S. Sakai, H. Tanaka, Scene Identification in News Video by Character Region Segmentation, Proceedings of ACM Multimedia 2000 Workshops, pp.195-200, Marina del Rey CA, Nov 2000.

46. [Ide99a] I. Ide, R. Hamada, S. Sakai, H. Tanaka, Identification of Scenes in News Video from Image Features of Background Region, First International Workshop on Multimedia Intelligent Storage and Retrieval Management (MISRM'99), Orlando FL, Oct. 1999.

47. [Ide99b] I. Ide, K. Yamamoto, H. Tanaka, Automatic Video Indexing Based on Shot Classification, Advanced Multimedia Content Processing -First International Conference AMCP'98, Osaka, Japan, 1998.

48. [InfWeb] Informedia Web Site, http://www.informedia.cs.cmu.edu/

49. [Jin99] H. Jin, R. Schwartz, S. Sista, F. Wall, Topic Tracking for Radio, TV Broadcast, and Newswire, Proceedings of the DARPA Broadcast News Workshop, 1999.

50. [Kal01] A. A. C. Kalker, J. A. Haitsma, J. C. Oostveen, Issues with Digital Watermarking and Perceptual Hashing, Proceedings of SPIE Conference on Multimedia Systems Applications, 2001.

51. [Kan00] M. S. Kankanhalli, T. S. Chua, Video Modeling Using Strata-Based Annotation, IEEE MultiMedia Vol. 7, No. 1, January-March 2000.

52. [Kas98] R. Kasturi, S. H. Strayer, U. Gargi, S. Antani, An Evaluation of Motion and MPEG Based Methods for Temporal Segmentation of Video, Technical Report CSE-98-014, Department of Computer Science and Engineering, Penn State University, 1998.

53. [Knu97] D. E. Knuth, The Art of Computer Programming, Addison-Wesley, 1997.

54. [Kop98] I. Koprinska, S. Carrato, Temporal Video Segmentation: A Survey, http://citeseer.nj.nec.com/378900.html

55. [Lew97] D. Lewis, The TREC-5 Filtering Track, The Fifth Text Retrieval Conference (TREC-5), NIST Special Publication 500-238, November, 1997.

56. [Lie01a] R. Lienhart, Reliable Transition Detection in Videos: A Survey and Practitioner's Guide, International Journal of Image and Graphics, vol. 1, no. 3, 2001.

57. [Lie01b] R. Lienhart, Reliable Dissolve Detection, Proceedings of Storage and Retrieval for Media Databases, SPIE vol. 4315, January, 2001.

58. [Lie99] R. Lienhart, Comparison of Automatic Shot Boudary Detection Algorithms, Proceedings of Storage and Retrieval for Still Image and Video Databases VII, SPIE vol. 3656-29, January, 1999.

59. [Lie97] R. Lienhart, C. Kuhmunch, W. Effelsberg, On the Detection and Recognition of Television Commercials, Proceedings of IEEE International Conference on Multimedia Computing and Systems, 1997.

60. [Lin02] W. Lin, A. Hauptmann, News Video Classification Using SVM-based Multimodal Classifiers and Combination Strategies, Proceedings of the Tenth ACM International Conference on Multimedia, December 2002.

61. [Lup98] G. Luptani, C. Saraceno, R. Leonardi, Scene Break Detection: A Comparison, Research Issues in Data Engineering, Workshop on Continuous Media Databases and Applications, 1998.

62. [Man94] U. Manber, Finding Similar Files in a Large File System. Proceedings of 1994 Winter USENIX Technical Conference, San Francisco, CA, Jan. 1994.

63. [Mar01] S. Marlow, D. A. Sadlier, K. McGeough, N. O'Connor, N. Murphy, Audio and Video Processing for Automatic TV Advertisement Detection, Proceedings of ISSC, 2001.

64. [Mar97] A. Martin, T. Kamm, G. Doddington, M. Ordowski, M. Przybocki, The DET Curve in Assessment of Detection Task Performance, Proceedings of EuroSpeech '97, 1997.

65. [Mcg83] M. J. McGill, G. Salton, Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983.

66. [Nag92] A. Nagasaka, Y. Tanaka, Automatic Video Indexing and Full-Video Search for Object Appearances, Visual Database Systems II, 1992.

67. [Nap03a] M. Naphade, J. Smith, A Framework for Moderate Vocabulary Semantic Visual Concept Detection, IEEE ICME 2003.

68. [Nap03b] M. Naphade, *et al.*, IBM Research TRECVID-2003 Video Retrieval System, TREC (VIDEO) Conference, 2003.

69. [Nap00a] M. Naphade, A Probabilistic Framework for Semantic Indexing and Retrieval in Video, IEEE International Conference on Multimedia and Expo, New York, 31 July-2 August 2000.

70. [Nap00b] M. Naphade, A Factor Graph Framework for Semantic Indexing and Retrieval in Video, Content-Based Access of Image and Video Library 2000 June 12, 2000 held in conjunction with the IEEE Computer Vision and Pattern Recognition 2000.

71. [Ngo03] C. Ngo, A Robust Dissolve Detector by Support Vector Machine, Proceedings of the Eleventh ACM International Conference on Multimedia, November 2003.

72. [Oos01] J. C. Oostveen, A. A. C. Kalker, J. A. Haitsma, Visual Hashing of Digital Video: Applications and Techniques, SPIE Applications of Digital Image Processing XXIV, San Diego, July 2001.

73. [Pap99] R. Papka, On-line New Event Detection, Clustering, and Tracking, Ph.D. Dissertation, University of Massachusetts Amherst, September, 1999.

74. [Pas99] G. Pass, R. Zabih, Comparing Images Using Joint Histograms, Multimedia Systems, vol. 7, no. 3, 1999.

75. [Pas96] G. Pass, R. Zabih, Histogram Refinement for Content-Based Image Retrieval, Proceedings of 3rd IEEE Workshop on Applications of Computer Vision, 1996.

76. [Pet01a] D. Petkovic, Content-Based Video Retrieval, VII Conference on Extending Database Technology (EDBT), Ph.D. Workshop, Konstanz, Germany, March, 2001.

77. [Pet01b] D. Petkovic, W. Jonker, Content-Based Video Retrieval by Integrating Spatio-Temporal and Stochastic Recognition of Events, IEEE International Workshop on Detection and Recognition of Events in Video, Vancouver, Canada, July, 2001.

78. [Pet96] D. Petkovic, P. Aigrain, H. Zhang, Content-based Representation and Retrieval of Visual Media: A State-of-the-Art Review, Multimedia Tools and Applications, Vol. 3 No.3, pp179-202, 1996.

79. [Pon98] D. Ponceleon, S. Srinivasan, A. Amir, D. Petkovic, D. Diklic, Key to Effective Video Retrieval: Effective Cataloging and Browsing, ACM Multimedia 1998, 1998.

80. [Pua03] K. M. Pua, J. M. Gauch, S. E. Gauch, J. Z. Miadowicz, Real Time Repeated Video Sequence Identification, Computer Vision and Image Understanding, pp. 310-327, 2004.

81. [Pua02] K. M. Pua, Feature-Based Video Sequence Identification, Ph.D. Dissertation, The University of Kansas, May, 2002.

82. [Pua99] K. M. Pua, S. Gauch, J. Gauch, VIDSEEK: Dynamic Multidimentional Browsing of Video Archives, Proceedings of ACM SIGIR '99, Berkeley, CA, USA, 1999.

83. [Row94] L. A. Rowe, J. S. Boreczky, C. A. Eads, Indexes for User Access to Large Video Databases, SPIE Proceedings 2185, February, 1994.

84. [Sab95] C. L. Sabharwal, S. K. Bhatia, Perfect Hash Table Algorithm for Image Databases Using Negative Associated Values, Pattern Recognition 28 (7), 1995.

85. [Sch99] J. M. Schultz, M. Liberman, Topic Detection and Tracking Using idf-Weighted Cosine Coefficient, Proceedings of the DARPA Broadcast News Workshop, 1999.

86. [Sha95] B. Shahraray, Scene Change Detection and Content-Based Sampling of Video Sequences, Proceedings of Digital Video Compression, Algorithm and Technologies, SPIE Proceedings vol. 2419, 1995.

87. [Shi03] F. Shipman, A. Girgensohn, L. Wilcox, Generation of Interactive Multi-level Video Summaries, Proceedings of the Eleventh ACM International Conference on Multimedia, November 2003.

88. [Shi95] N. Shivakumar, H. Garcia-Molina, SCAM: A Copy Detection Mechanism for Digital Documents, In Proceedings of the Second International Conference on Theory and Practice of Digital Libraries, Austin, Texas, June 1995.

89. [Sin97] A. Singhal, M. Mitra, C. Buckley, Learning Routing Queries in a Query Zone, Proceedings of ACM SIGIR 1997, 1997.

90. [Smi96] J. R. Smith, S. F. Chang, VisualSEEk: A Fully Automated Content-Based Image Query System, Proceedings of ACM Multimedia 1996, Boston MA, USA, 1996.

91. [Spa97] K. Spark Jones, P. Willet, Readings in Information Retrieval, Morgan Kaufmann Publishing, San Francisco, 1997.

92. [Str96] M. Stricker, A. Dimai, Color Indexing with Weak Spatial Constraints, SPIE Proceedings vol. 2670, 1996.

93. [Tag92] J. Tague-Sutcliffe, Measuring the Informativeness of a Retrieval Process, Proceedings of ACM SIGIR 1992, 1992.

94. [Tru00a] B. T. Truong, C. Dorai, S. Venkatesh, New Enhancements to Cut, Fade, and Dissolve Detection Processes in Video Segmentation, ACM Multimedia 2000, November, 2000.

95. [Tru00b] B. T. Truong, C. Dorai, S. Venkatesh, Improved Fade and Dissolve Detection for Reliable Video Segmentation, Proceedings of IEEE International Conference on Image Processing (ICIP) 2000, vol. 3, 2000.

96. [Xu94] J. Xu, J. Broglio, W. B. Croft, The Design and Implementation of a Part of Speech Tagger for English, Technical Report IR-52, University of Massachusetts Center for Intelligent Information Retrieval, 1994.

97. [Yam99] J. Yamron, I. Carp, L. Gillick, S. Lowe, P. van Mulbregt, Topic Tracking in a News Stream, Proceedings of the DARPA Broadcast News Workshop, 1999.

98. [Yan03] H. Yang, L. Chaisorn, Y. Zhao, S. Neo, T. Chua, VideoQA: Question Answering on News Video, Proceedings of the Eleventh ACM International Conference on Multimedia, November 2003.

99. [Zab99] R. Zabih, J. Miller, K. Mai, A Feature-Based Algorithm for Detecting and Classifying Production Effects, Multimedia Systems, vol. 7, 1999.

100. [Zab95] R. Zabih, J. Miller, K. Mai, A Feature-Based Algorithm for Detecting and Classifying Scene Breaks, Proceedings of ACM Multimedia 1995, San Francisco, CA, November, 1995.