NTNU
Innovation and Creativity

# Efficient Algorithms for Video Segmentation

**Vegard Andre Kosmo**

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

# Problem Description

The main goal of this thesis is to find efficient algorithms that automatically build a picture storyline from an input video file. The algorithms will be evaluated based on both output quality and elapsed time during the operation.

Assignment given: 20. January 2006
Supervisor: Herindrasana Ramampiaro, IDI

# Abstract

Describing video content without watching the entire video is a challenging matter. Textual descriptions are usually inaccurate and ambiguous, and if the amount of video is large this manual task is almost endless. If the textual description is replaced with pictures from the video, this is a much more adequate method. The main challenge will then involve which pictures to pick to make sure the entire video content is covered by the description. TV stations with an appurtenant video archive would prefer to have an effective and automated method to perform this task, with focus on holding the time consumption to an absolute minimum and simultaneously get the output results as precise as possible compared with the actual video content.

In this thesis, three different methods for automatic shot detection in video files have been designed and tested. The goal was to build a picture storyline from input video files, where this storyline contained one picture from each shot in the video. This task should be done in a minimum of time. Since video files actually are one long series of consecutive pictures, various image properties have been used to detect the video shots. The final evaluation has been done based both on output quality and overall time consumption.

The test results show that the best method detected video shots with an average accuracy of approximately 90%, and with an overall time consumption of 8.8% of the actual video length. Combined with some additional functionality, these results may be further improved.

With the solutions designed and implemented in this thesis, it is possible to detect shots in any video file, and create a picture storyline to describe the video content. Possible areas of application are TV stations and private individuals that have a need to describe a collection of video files in an effective and automated way.

# Preface

This master thesis documents the work performed by Vegard André Kosmo from January to June in 2006. The main goal of the thesis was to examine different approaches for automatic segmentation of video files.

I would like to thank associate university professor Herindrasana Ramampiaro from the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) for his guidance and sharing of expertise during my work on this thesis.

Trondheim

Vegard André Kosmo

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This report describes the work of designing, testing and evaluating efficient algorithms related to segmentation of video files. This chapter gives a brief overview of the concept of video segmentation, challenges and problems related to this matter and a discussion of some possible strategies to manage such operations.

## 1.1 Problem description

The amount of produced digital video information in our society is growing fast, especially among the private individuals. The main reasons for this growth are cheaper and better video cameras, higher availability of digital video editing software and dramatically reduced storage costs. An obvious consequence of this tendency is increased needs to manage this video information. Management in this context involves production, compression, storage, indexing and retrieval of video.

Because video files actually consist of a series of pictures, and each picture may contain large amounts of information, video management is a quite complex and challenging matter. It is a growing need to find methods that indicate the content of a video file in a short but still descriptive way. A user of a video management system may then decide if a video file is relevant to his/her needs, without actually watching the video. While there are plenty of tasks that have to be solved to make a complete video management system, this thesis will mainly focus on the matter of describing the entire content of video files by segmenting the files into parts and describing each part independently.

## 1.2 Challenges

First of all, describing video content by means of pure text is very difficult. It is a process that have to be done manually, and may suffer from different interpretations from different people. The generation of these textual metadata is also a time consuming process, and the task will be insuperable with the amount of video information growing large. This leads to the desire of using other methods to describe video content, either in combination with textual data or as an independent solution.

One possible way to approach this problem is by using pictures to make a summation of the video file. Pictures are descriptive and nonambiguous, and should be generated in less time than the textual metadata. Another major advantage is the possibility to automate this task, i.e. let a computer carry out the entire operation. This leads to another challenge: How to pick the right pictures. A perfect algorithm would find all parts of the video file, find one descriptive picture from each part and remove pictures that are almost identical. The entire operation would be done in a minimum of time. Unfortunately, it is not that easy. As in many other domains, it is a trade-off between accuracy and speed. Speeding up the algorithm may lead to loss of important parts of the video file, or pictures with poor quality. An accurate algorithm will probably use as much time as it takes to watch the entire video file. Because of this, it is important to attack one problem at a time, and in the end combining the solutions to make the final result as good as possible. That is what this thesis is about.

## 1.3 Thesis title and description

The thesis title is

*Efficient algorithms for video segmentation*

with the description

*The main goal of this thesis is to find efficient algorithms that automatically build a picture storyline from an input video file. The algorithms will be evaluated based on both output quality and elapsed time during the operation*

## 1.4 Solution strategies

Since video segmentation is an area that has been explored for several years, it is a natural solution strategy to use algorithms already known, and utilise these as a platform to test and combine different solutions in a new way.

Since the segmentation process can be divided into three main parts, it is desirable to optimize each part and combine the best solutions from each part. The first part is related to segment the video files into pieces, and make sure that the number of pieces found is equal to the actual number in the video file. There exists several ways to do this, and they will be thoroughly described in the next chapter. The next part is to pick pictures from each part of the video file, and use these pictures to describe the video file content. If the pictures are presented in ascending order, they will form a storyline that can be viewed in much less time than watching the video. From a user's point of view it is desirable that pictures containing almost exactly the same thing will be removed, so the picture storyline will be as compact as possible. The strategies for finding these pictures will be introduced in the next chapter. The final part of the segmentation process is to optimize the operations, to make sure the time consumption will be minimized. It is important to monitor the output quality during this operation, as higher speed probably produces less precise result sets. Different solutions to speed up the segmentation algorithms will be discussed in the next chapter.

To evaluate the quality of each algorithm, it is necessary to define test cases. The video files used in each test case have to be examined manually, to find the segments of each file. The output from each algorithm is then to be compared with the ideal output, and evaluated on the basis of this combined with the overall time consumption. The easiest way to run the test cases will be to make a simple graphical user interface, with the possibilities to open a video file and construct a picture storyline. It is also adequate to implement a timer that shows the progress of the segmentation process, combined with overall time used. The main focus related to the test prototype will not be ease of use and fancy layout, but making a tool to run the test cases and draw conclusions from the produced output.

## 1.5 Thesis contribution

One possible outcome of this thesis is to design and implement a solution that can be used by TV stations to automatically describe their video content. Most TV productions these days are made with digital video cameras, where a cut is defined automatically with the touch on the start/stop button. But even if the camera is stopped for a while, it is not necessarily a cut on exactly that point in the video. If the camera starts to film the same things after the break, the cut point defined by the digital camera is obviously wrong. This is one thing that can be improved by designing new ways to detect shots in video files. Another point of designing shot detection algorithms is to have the possibility to automatically detect shots in video files that are older than the digital video camera technology. Most TV stations have archives that

include older video material, and the possibility to describe this content as well would be of great importance.

## 1.6 Thesis outline

The structure of the remaining part of the report is as follows: Chapter 2 gives an overview of theoretical background information to completely understand the scope of the problem, and how it can be solved. Chapter 3 analyses the requirements that have to be considered during the development of a video segmentation test prototype. Chapter 4 describes the construction of the algorithms that will be tested and evaluated, the test cases and the test case prototype. Chapter 5 describes the most important parts related to the implementation of the shot detection prototype. Chapter 6 gives the results of the test cases, while chapter 7 concludes which algorithms that are best suited for these kind of operations, in addition to an overall summery of the entire thesis.

# Chapter 2

# Background theory

This chapter describes video as a medium, and gives a thorough explanation of the process of video segmentation. Some known methods and algorithms regarding this matter will be introduced. Since segmentation algorithms is a very important part of this thesis, this section is quite comprehensive. I think it is important to illustrate a wide selection of earlier results and conclusions, and use this knowledge to explore new possibilities instead of doing work that already has been done and evaluated. Further, this chapter suggests how to select the best pictures from a video file to build a complete picture storyline, which from a user's point of view will represent a compact summation of the video file content. The final section presents methods for speeding up the entire segmentation process, with the goal of keeping the output quality as unaffected as possible. These methods will create a foundation for further use, related to test cases and evaluation of the algorithms presented in later chapters.

## 2.1 Video as a medium

To design efficient video shot detection algorithms, it is important to examine video as a medium to fully understand this domain. Video is a continuous series of pictures displayed sequentially at a fixed rate. Because all the pictures in a video file have equal size, the pictures are called **frames**. Digital video information consists of a series of 25 frames per second [1]. These frames can be grouped together in related collections, to make the handling of the video file easier. The collections are defined as follows, and will be used through the entire thesis [2]:

- *Clip:* A clip is a digital video document. It can last from a few seconds to several hours, and consists of a sequence of contiguous video frames. A video **segment** is any contiguous portion of a clip.

- *Scene:* A scene is a sequential collection of shots unified by a common event or locale. A clip can have one scene or several scenes.
- *Shot:* A shot is captured between a record and a stop camera operation. A scene can have one shot or several shots.
- *Frame:* A frame is the atomic part related to digital video, and is one picture from the picture sequence.

This can be organized hierarchically, as shown in figure 2.1. A shot is a collection of frames, a scene is a collection of shots and a clip is a collection of scenes. The figure is taken from [2].

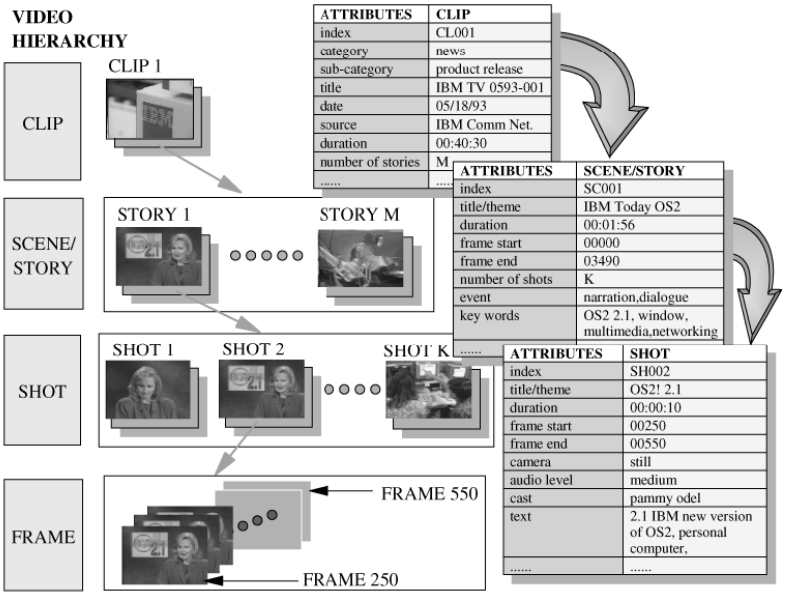


Figure 2.1: A hierarchy of digital video

Since the frame rate of digital video is 25 frames per second, a full motion picture contains a large number of frames. Storing and managing the digital video content is a challenging matter, because of the great amount of information. A compressed full-length movie requires one or two CD-ROMs, where each CD-ROM holds 700 MB of data. A DVD movie requires up to 4.7 GB [3], because of the increased image and sound quality.

## 2.2 Video codecs

The word **'codec'** is a compounding of the two words **co**mpression and **dec**ompression, which describes a device or program capable of performing transformations on a data stream or signal [4]. Video codecs are devices

or software modules that enables the use of compression for digital video. Compression is a conversion of data to a format that requires fewer bits, usually performed so that the data can be stored or transmitted more efficiently. The compression usually employs lossy data compression, where compressing data and then decompressing it retrieves data that may well be different from the original, but is close enough to be useful in some way.

A typical digital video codec design starts with conversion of camera-input video from RGB color format to YCbCr color format. The conversion to YCbCr provides two benefits. First, it improves compressibility by providing decorrelation of the color signals. Second, it separates the luma signal, which is perceptually much more important, from the chroma signal, which is less perceptually important and which can be represented at lower resolution. To reduce the raw data rate before the basic encoding process, spatial and temporal down sampling may be used. The most popular transform is the discrete cosine transform (DCT), which is similar to the discrete Fourier transform (DFT), but using only real numbers. DCT is used both in digital video and image compression. A two-dimensional DCT of $N \times N$ blocks is computed, and the results are quantized and entropy coded. Quantization is the process of approximating a continuous range of values by a relatively small set of discrete integer values, while entropy coding involves the actual compression process. $N$ is typically 8, and the following formula is applied to each row and column of the block:

$$f_j = \sum_{n=0}^{N-1} x_n cos \left[ \frac{\pi}{N} j(n + \frac{1}{2}) \right] \tag{2.1}$$

The result is an 8×8 transform coefficient array in which the (0,0) element is the DC zero-frequency component and entries with increasing vertical and horizontal index values represent higher vertical and horizontal spatial frequencies.

Several codecs are now in everyday use. The most important ones are listed below:

- *MPEG-1:* This codec is used for Video CDs, and also sometimes for online video. The quality is roughly comparable to that of VHS. It includes the .mp3 standard. When it comes to compatibility, VCD has the highest compatibility of any digital video/audio system.
- *MPEG-2:* This codec is used on DVD and in another form for SVCD, and used in most digital video broadcasting and cable distribution systems.

- *MPEG-4:* This codec is an MPEG standard that can be used for internet, broadcast, and on storage media. It offers improved quality relative to MPEG-2.
- *DivX, XviD and 3ivx:* Video codec packages basically using MPEG-4 video codec, with the .avi, .mp4, .ogm or .mkv file container formats.
- *Windows Media Video (WMV):* This is Microsoft's family of video codec designs including WMV 7, WMV 8, and WMV 9. It can do anything from low resolution video for dial up internet users, to High-Definition Television (HDTV). Files can be burned to CDs and DVDs, or put out to any number of devices. It is also useful for Media Centre PCs.

## 2.3 Video segmentation

As described in the section 2.1, a **video segment** is any contiguous portion of a video clip. But in the context of building a structure to increase management and retrieval efficiency, picking segments randomly in a video file is not a good idea. It is important to have a strict plan to follow, to make sure the output of the segmentation task is exactly what was intended in the first place. The main idea is to segment the video files into shots, and then pick one picture from each shot to show the content at that point in the video file. Such a picture is called a **representative frame**, or r-frame for short. Some papers use the name **key-frame** for this purpose, but it is in fact exactly the same as an r-frame. The name r-frame will be used throughout this thesis.

### 2.3.1 Detecting shots

Shot detection in video files can be carried out in many different ways. Most of the activity is concentrated on the detection of shot boundaries using sharp transition detection, cuts [5]. A cut, or shot boundary, is a sharp transition between a shot and the one following. Great difference in brightness pattern from one picture to the other is in most cases a cut. The main reason for this is the simple fact that two consecutive frames inside the same shot do not change significantly in their background and object content, and their overall brightness distribution differs little.

Examples of other types of shot boundary are **fades**, where the frames of the shot gradually change from or to black, **dissolves**, where the frames of the first shot are gradually morphed into the frames of the second and **wipes**, where the frames of the first shot are moved gradually in a horizontal or vertical direction into the frames of the second [1]. These are all matters

that complicate the process of automatic shot boundary detection. Since the changing occurs gradually from one shot to the next one, the frame difference is much smaller than the gap between two frames in sharp transition shots.

Automatic shot boundary detection is also difficult because of camera and object motion [1]. Sports and music videos usually have much object motion, while cameras have a variety of movement such as panning, tilting, booming, tracking, zooming in and out, or a combination of these. Another case is sudden changes in light or color values inside a shot. These matters can lead to false shot detections in an automatic process.

### 2.3.2 State of the art in r-frame generation

Video segmentation has previously been divided into two main types of video abstraction, the **video skimming** and the **video summary** [5].

Video skimming can be divided further down into two sub-categories: The **video highlight** and the **summary sequence**. The video highlight is like a movie trailer, containing the most interesting parts of the original video file. Important people and/or objects, or frames with the largest difference with the respect to color, motion and audio estimation are usually selected. In the end are all the selected scenes organized according to their time relevance in the movie. The summary sequence approach speeds up the playback, so that the whole film can be watched in a shorter amount of time. This is most challenging in the case of getting the audio to fit with the images, so this solution is most in use related to documentaries with text content.

Video summary uses still images (r-frames), to describe the content of video shots. This method is much more effective in the case of video content retrieval. There are five categories of constructing r-frames from a video file: The **sampling-based**, **segment-based**, **motion-based**, **mosaic-based** and **shot-based**.

In the **sampling-based** approach, r-frames are selected randomly or uniformly at certain time intervals. The process is automatic. Since the method does not take shot length into consideration, a possible drawback is to miss shots that are small in time. On the other hand, sample-based detection may generate an unnecessary amount of r-frames from the same shot, if the shot is long in time. This does not scale for long videos.

As a consequence of this, the **segment-based** r-frames constructing model has been developed. This method is based on each segment's length and rarity. Segments with their importance lower than a predefined threshold

are discarded. The selected r-frame of a segment will be the frame which is closest to the center of that given segment. The main drawback in this model is that semantically properties might be missed because of the threshold filtering.

**Motion-based** r-frames construction controls number of frames based on temporal dynamics in the scene. Pixel-based image differences or optical flow computation are commonly used. Optical flow is calculated for each frame, and a motion metric is computed on the basis of this calculation.

The **mosaic-based** approach is used to generate a synthesized panoramic image that can represent the entire content of a video scene. Static background mosaic is used for background scenes, while synopsis mosaic summarizes the entire dynamic foreground event in the video clip by detecting the object trajectory.

The most sophisticated video summery method is the **shot-based** approach. This method demands content interpretation by means of low-level visual features. The three most common features used for this purpose is **color**, **texture** and **shape**. These will be described in the next subsection. Video retrieval under the shot-based method is divided into two main categories. The **dependent-video features** category focus on environment, as a background street in a picture, or on static or moving objects, such as a car. The goal is to distinguish the dominant objects between consecutive frames. Dominant objects in a frame are those color objects with the largest number of pixels. To determine if a shot is a part of a scene, the algorithm performs a correlation score calculation. The shot is a part of the scene if the result satisfies the condition. The **independent-video features** category integrates domain-independent video features such as color and shape of semantic objects and object locations.

### 2.3.3 Visual features in video shot segmentation

Constructing r-frames to describe the content of video files has one big advantage: Efficient search and retrieval. While it is quite complex to search a video file based on text or pictures, several efficient image retrieval algorithms have been developed. The basis for the search is then a limited amount of images instead of a full length video file, which clearly simplifies the retrieval process. Because most shot detection algorithms compare one frame with the next one, visual features in video along with efficient image retrieval algorithms is highly relevant in the regard of video shot segmentation.

**Color**
To make use of the color features in image and video retrieval, a standardized

color model has to be present. One of the most common color models in use is the **RGB model**. In this model, the primary colors are red, green and blue. All other colors are produces by combining the three primary colors into different composition, where white contains all the colors and black being the absence of any color. When written, the RGB model represent each color by a unique 24 bit value. The 24 bits are divided into three integers between 0 and 255, where each integer represent one of the three primary colors. White has the bit value (255, 255, 255), while black has (0, 0, 0). With this system, approximately 16.7 million discrete colors can be reproduced. Figure 2.2 illustrates the RGB model.

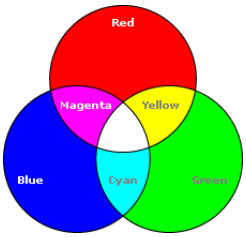


Figure 2.2: An illustration of the RGB model

Given the knowledge that each pixel in every frame of the entire video file has a 24 bit integer value, these values have to be organized in a way that makes shot detection and image retrieval possible. One solution to this is to make use of **color histograms**. Their main advantages are efficiency and insensitivity to small changes in camera viewpoint. In addition to this, color histograms are computationally trivial to compute. In [6], the following definition of color histograms is given:

All images are scaled to contain the same number of pixels $M$. The color space is discretized to contain $n$ distinct colors. Since it is not very efficient to use all 16.7 million possible colors, a few of the most significant bits are used. If the 2 most significant bits are used, the histogram may contain up to 64 distinct colors. A color histogram $H$ is a vector $< h_1, h_2, ..., h_n >$, in which each bucket $h_j$ contains the number of pixels of color $j$ in the image. Typically images are represented in the RGB color space. For a given image $I$, the color histogram $H_I$ is a compact summary of the image. To compare two images based on their color histograms, two different distances can be used. The first one is called **L1-distance**, and is defined as the sum of absolute value of differences:

$$|H_I - H_{I'}| = \sum_{j=1}^{n} |H_{I[j]} - H_{I'[j]}| \tag{2.2}$$

The second approach is called **L2-distance**, and is defined as the sum

of squared differences:

$$||H_I - H_{I'}|| = \sum_{j=1}^{n} (H_{I[j]} - H_{I'[j]})^2 \tag{2.3}$$

The differences are weighted evenly across different color buckets for simplicity.

Even if color histograms are very efficient in the case of image comparing and retrieval, they still have some drawbacks. They do not contain spatial information, so images with very different appearances can have similar histograms. This is shown in figure 2.3. The figure is taken from [6].

Figure 2.3: Two images with similar color histograms

One way to overcome the limitations of color histograms is use **color coherence vectors**. A color's coherence is the degree to which pixels of that color are members of large similarly-colored regions. The significant regions are referred to as coherent regions. The images shown in figure 2.3 are quite different in content and context, but the color histograms are similar. The amount of red regions are equal in both images, but the picture to the right has all the red pixels gathered inside one coherent region. This is the idea behind coherence vectors. Coherent pixels are a part of some sizable contiguous region, while incoherent pixels are not. A color coherence vector, or CCV for short, represents this classification for each color in the image. This allows fine distinctions that cannot be made with color histograms.

In [6], a method for CCV computing is given. The initial stage is similar to the computation of a color histogram. The picture is blurred by replacing pixel values with the average value in a small local neighborhood. The color space is discretized, such that there are $n$ distinct colors in the image. Then the pixels are classified within a color bucket as either coherent or incoherent. A coherent pixel is part of a large group of pixels of the same color, while an incoherent pixel is not. The next step is to construct **connected components**, which are maximal sets of pixels such that for any two pixels

*p,p'* as elements of *C*, there is a path between *p* and *p'*. Connected components are only computed within a given discretized color bucket. When this is complete, each pixel will belong to exactly one connected component. Pixels are classified as either coherent or incoherent depending on the size in pixels of its connected component. A pixel is coherent if the size of its connected component exceeds a fixed threshold value, otherwise the pixel is incoherent.

The test results in [6] show that the overall computation of CCVs is slower than color histograms, but the results of image retrieval and ranking by relevance are much better with CCVs.

**Shape**

Shape features in images can also be used in the purpose of detecting shots in video files. In [7], shape of the objects within an r-frame is represented using **moment invariants**. The moment of an image *f(x,y)* is defined as:

$$m_{pq} = \sum\sum x^p y^q f(x,y) \tag{2.4}$$

Invariance to scale change, rotation and translation are some characteristics that make moment invariants an ideal representation mechanism in a video browser. Moment invariants are derived from normalized central moments defined as:

$$\eta_{pq} = \frac{1}{m_{00}^{\gamma}} \sum\sum (x - x_{\prime})^p (y - y_{\prime})^q f(x,y) \tag{2.5}$$

where $\gamma = (\frac{p+q}{2} + 1)$, $x_{\prime} = \frac{m_{10}}{m_{00}}$ and $y_{\prime} = \frac{m_{01}}{m_{00}}$.

The first few moment invariants are defined as:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{2.6}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{2.7}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{2.8}$$

The shape of each r-frame is then represented using the vector $\vec{\sigma}$ defined as:

$$\vec{\sigma} = \{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_7\} \tag{2.9}$$

Finally, the euclidean distance is used to measure the similarity of two r-frames:

$$\psi(\alpha, \beta) = |\vec{\sigma}_\alpha - \vec{\sigma}_\beta|^2 \tag{2.10}$$

In [8], shape is classified as the most important low level visual feature among color, texture and spatial localization. The reason for this is that shape represents significant regions or relevant objects in images. In general, shape representations are classified into two categories: **Boundary-based** and **region-based**. The first one uses the pixels along the object boundary, while the second one uses the pixels contained in the region.

In [9], the focus is on finding shapes that become similar to a given shape after being rotated. To do this, **Fourier descriptors** are used. Given a boundary function $b_t$, its Fourier transform can be written as:

$$B_f = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} b_t e^{\frac{-j2\pi tf}{N}} \tag{2.11}$$

where $f \in \{\lfloor \frac{(N-1)}{2} \rfloor, ..., 0, ..., \lceil \frac{(N-1)}{2} \rceil\}$ and $j = \sqrt{-1}$ is the imaginary unit. The coefficients $B_0$, $B_{\pm 1}$, called Fourier descriptors, describe the shape of the object in the frequency domain. The energy in the frequency domain is the same as the energy in the spatial domain, and the inverse Fourier transform gives the original boundary function. Given two boundary functions, $b_t$ and $b'_t$, a typical measure of similarity between the two boundaries is the **Euclidean distance**, which corresponds to mean-square error and which is also directly related to the cross-correlation:

$$D^2(\mathbf{b}, \mathbf{b}') = \sum_{t=0}^{N-1} |b_t - b'_t|^2 \tag{2.12}$$

**Texture**

Another way to approach shot detection and image retrieval in general, is to make use of the image texture features. In [10], **Gabor filters** are used in this purpose. Gabor filter, or Gabor wavelet, is widely used to extract texture features from the images. The method has shown to be very efficient. Other texture methods to mention are **pyramid-structured wavelet transform**, **tree-structured wavelet transforms** and **multi-resolution simultaneous autoregressive model**. Image retrieval with Gabor filters outperforms these methods [10].

Gabor filters are a group of wavelets, with each wavelet capturing energy at a specific frequency and a specific direction. Texture features can then be extracted from this group of energy distributions. For a given image *I(x, y)* with size $P \times Q$, its discrete Gabor wavelet transform is given by a convolution:

$$G_{mn}(x,y) = \sum_s \sum_t I(x-s, y-t)\psi^*_{mn}(s,t) \tag{2.13}$$

where $s$ and $t$ are the filter mask size variables, and $\psi^*_{mn}$ is the complex conjugate of $\psi^*mn$, which is a class of self-similar functions generated from dilation and rotation of wavelets.

After applying Gabor filters on the image with different orientation at different scale, an array of magnitudes is obtained:

$$E(m,n) = \sum_x \sum_y |G_{mn}(X,Y)| \tag{2.14}$$

The magnitudes represent the energy content at different scale and orientation of the image. The main purpose of texture-based retrieval is to find images or regions with similar texture. The mean $\mu_{mn}$ and standard deviation $\sigma_{mn}$ of the magnitude of transformed coefficients are used to represent the homogenous texture feature of the region:

$$\mu_{mn} = \frac{E(m,n)}{P \times Q} \tag{2.15}$$

$$\sigma_{mn} = \frac{\sqrt{\sum_x \sum_y (|G_{mn}(x,y)| - \mu_{mn})^2}}{P \times Q} \tag{2.16}$$

A feature vector $\mathbf{f}$(texture representation) is created using $\mu_{mn}$ and $\sigma_{mn}$ as the feature components. Five scales and six orientations are used in common implementation and the feature vector is given by:

$\mathbf{f} = (\mu_{00}, \sigma_{00}, \mu_{01}, \sigma_{01}, \ldots, \mu_{45}, \sigma_{45})$

The texture similarity measurement of a query image $Q$ and a target image $T$ is defined by:

$$D(Q,T) = \sum_m \sum_n d_{mn}(Q,T) \tag{2.17}$$

where

$$d_{mn} = \sqrt{(\mu^Q_{mn} - \mu^T_{mn})^2 + (\sigma^Q_{mn} - \sigma^T_{mn})^2} \tag{2.18}$$

In [10], retrieval tests were run both on texture images and natural images. The extracted texture features are used to measure the similarity between the compared images. The method is most useful if the entire image or main part of the image has a uniform texture.

In the relation to video segmentation, shots may be detected because of sudden changes in image texture. In [11], Gabor filters are used for this purpose. The Gabor energy method measures the similarity between neighborhoods in an image and Gabor masks. Each Gabor mask consists of Gaussian windowed sinusoidal waveforms with parameters of wavelength $\lambda$, orientation $\theta$, phase shift $\phi$ and the standard deviation $\sigma$. The filter is given by:

$$G(x,y) = e^{-\frac{(x-X)^2+(y-Y)^2}{2\sigma^2}} \times sin(\frac{2\pi(xcos\theta - ysin\theta)}{\lambda} + \phi) \quad (2.19)$$

A set of filters is generated by varying the $\theta$ and $\lambda$. The texture energy for a filter is calculated as the sum over the phases of the squared convolution values. Next, the texture energy response of each frame is used to find the difference between the adjacent group of frames. To compute the distance measure at frame $i$, a Gaussian window at scale $s$ is selected around the frame. The weighted average texture energy is calculated to the left and right of the frame. The normalized distance between the average texture energy is used as the estimate of the change across segment boundary.

## 2.4 Shot detection algorithms

This section describes how shot detection algorithms can be evaluated, along with the results from an article that compares a few selected methods of shot detection. Further, some algorithms and methods from the literature will be presented in detail, to give a broadly illustration of how many ways these kinds of problems may be solved.

### 2.4.1 Evaluating shot detection algorithms

Test results in [12] shows that it is probably not possible to find only one optimal segmentation algorithm without being supplemented by heuristics which involve both semantic information and priori knowledge about the images under consideration. Hundreds of segmentation algorithms have been presented in the literature, but there is no single general algorithm which can be considered good for all images, nor are all algorithms equally good for a particular image. Selecting different algorithms to segment different types of video seems to be the most straightforward and effective solution.

In computer science in general, there are two main criteria that are used to evaluate the overall quality of an algorithm. The first one is related to computation costs during the operation. This factor is in most cases evaluated on the basis of how much time is needed to get the desired output. In the process of video shot detection it is important to use efficient methods to analyse the frames, because one hour of full quality video contains 90 000

different frames.

The second criterion is related to the output produced by the algorithm, compared with the expected output. There are two values that may be used in result evaluation: **Precision** and **recall** [13]. During testing, there are one quantity of relevant results. In the domain of shot detection in video files, this quantity will be all shots that exists in the given video file, named $R$. $|R|$ is the number of shots that exist in the video file. After the shot detection algorithm is run, an answer set $A$ is returned. $|A|$ is the number of shots in this amount. $|Ra|$ is defined as the number of shots in the intersection between R and A, ergo relevant shots in the answer set.

**Precision** is the number of returned shots that are relevant, ergo

$$Precision = \frac{|Ra|}{A} \tag{2.20}$$

**Recall** is the number of relevant shots that are returned, ergo

$$Recall = \frac{|Ra|}{R} \tag{2.21}$$

Figure 2.4 shows the relation between precision and recall in a given result set. The figure is taken from [13].

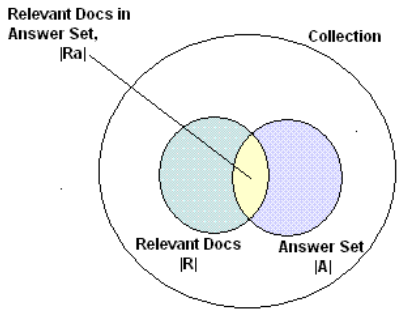


Figure 2.4: Precision and recall values in a given example

To make use of these values, the video files used in the test cases have to be examined manually before the tests are run. All shots have to be found by hand, and the test case outputs have to be compared with this ideal answer set.

In [14], a number of algorithms have been evaluated based on precision and recall values. Five different shot detection algorithms were run on the

same data set, and the output was compared against the predefined ideal output. The data set contained television programs, news programs, movies and television commercials. Plots of recall and precision values were made after running the test cases. All of the algorithms were designed to examine every frame of the test data rather than perform temporal sampling. The five selected algorithms were:

- *Histograms:* A gray-scale histogram was computed over the entire frame. The difference measure was the sum of the absolute histogram difference. A shot boundary was declared if the histogram difference between consecutive frames exceeded a predefined threshold.

- *Region histograms:* Each frame was divided into 16 blocks in a 4 X 4 pattern. A gray-scale histogram was computed for each region. Histogram differences were computed for each region between consecutive frames. If the number of region differences that exceeded the difference threshold was greater than the count threshold, a shot boundary was declared.

- *Running histograms:* A gray-scale histogram was computed over each image. If the histogram difference between consecutive frames exceeded a high threshold, a cut was declared. If the histogram difference exceeded a low threshold, a gradual shot transition was assumed. If this was the case, computing differences from the start of the gradual transition was started. If the running difference exceeded the high threshold, a gradual transition was declared once the run ended. If the difference dropped below the low threshold for more than two frames, the assumed gradual transition was declared to be over, and the running difference computation stopped.

- *Motion compensated pixel differences:* In this approach, 3 threshold values were used: **cut**, **high** and **low**. Each frame was divided into 12 blocks in a 4 X 3 pattern. Block matching with a 24 by 18 search window was used to generate a set of motion vectors and a set of block matching values. The two highest and two lowest match values were discarded and the remaining vales were averaged to produce the match value. If the match value exceeded the cut threshold, a cut was declared. Match values above the low threshold indicated a gradual transition. While exceeding the high threshold, a gradual transition was declared once the match value dropped below the low threshold.

- *DCT coefficient differences:* One threshold was used. The same 15 discrete cosine transform coefficients were taken from each block of frame and a vector was produced. DCT coefficients are used to compare the size of JPEG compressed frames, without any need of decompression.

The difference measure was computed, and a possible shot boundary was detected if the difference exceeded the threshold value.

All video material was manually analysed, and the number of frames, cuts and gradual transitions were found. The algorithms performed best on the television program data set, mainly because of the low number of gradual transitions. The **histogram algorithm** usually produced the first or second best precision for a given recall value. Important points regarding this algorithms are simplicity, along with straightforward threshold selection. The **region histogram** algorithm seemed to be the best algorithm for applications where recall is not the highest priority. The **running histogram algorithm** seemed to be the best algorithm for applications where recall is important. This algorithm was poor in gradual transition detection. The **motion algorithm** did not perform as well as the histogram-based algorithms in general. The greatest weakness was gradual transition detection. The **DCT algorithm** gave low precision values for a given recall. The algorithm generated a large number of false positives in black frames between television commercials. Because of this, a secondary algorithm is needed to examine the false detected frames. This make the DCT algorithm an ineffective filter.

The final conclusion of [14] is that algorithms with more than one threshold value are very sensitive to the threshold settings. In general, the simpler algorithms outperformed the more complicated algorithms.

### 2.4.2 Algorithm 1 - Video cut detection using frame windows

The first algorithm to be described is presented in [15], and makes use of a technique called frame windows. Their technique makes use of the intuition that frames preceding a cut are similar to each other, and dissimilar to those following the cut. For each frame in the video, a set or window of consecutive, ordered frames is extracted centred on that frame. Second, the frames in the window are ordered by decreasing similarity to the current frame. Last, the ranking of the frames are inspected. The number of frames preceding the current frame in the original video that now are ranked in the first half of the list is recorded, and it is called the **pre-frame count**. This process is repeated for each frame. Cuts are detected by identifying significant changes in the pre-frame count between consecutive frames.

The basic approach is to make use of the fact that a cut is indicated by a frame that is dissimilar to a window of those that precede it, but similar to a window of those that follow it. A good illustration of this approach is shown in figure 2.5. The figure is taken from [15]

The figure shows a cut between the 14th and the 15th frame. After


Figure 2.5: Moving query window with a half-window size of 5

processing the first 12 frames in the fragment, the 13th frame is the **current frame** that is being considered as a possible cut. Five **pre-frames** are shown marked before the current frame, and five **post-frames** follow it. With the current frame centered, the pre- and post-frames together constitute a **moving window**. It is referred to as moving because it is used to sequentially consider each frame in the video as possibly bordering a cut. The number of pre- and post-frames is always equal, and called the **half-window size, HWS**. In the figure, HWS = 5.

To detect whether the current frame $f_c$ borders a cut, a collection of frames $C$ is created from the frames $f_{c-HWS}...f_{c-1}$ and $f_{c+1}...f_{c+HWS}$ that respectively precede and follow the current frame $f_c$. The global feature data of the frames in $C$ is summarized, and the distance between the current frame $f_c$ and each frame in $C$ is computed. The frames are ordered by increasing distance from the current frame to achieve a ranking. Only the first $\frac{|C|}{2}$ top-ranked frames are considered. This number is equal to the HWS. The number of pre-frames in the $\frac{|C|}{2}$ top-ranked frames is referred to as the pre-frame count. If the value of the pre-frame count is zero or close to zero, it is likely that a cut has occurred. In figure 2.5, the current 13th frame is not the first in a new shot and therefore does not define a cut. It is expected that the five pre-frames and the first post-frame would be ranked as more similar to the current frame than the remaining post-frames. The pre-frame count is four or five, and a cut is unlikely to be present.

To make cut detection more efficient, it is possible to combine rankings from the moving window approach. A representation of a video is shown in figure 2.6. The figure is taken from [15]. The video contains two shots labelled A and B, and the HWS value is set to 10. The figure shows five different situations that occur as the video is sequentially processed with the frame window algorithm.

The first row shows the situation where the moving window is entirely within shot A. The pre-count is 5, and a cut is not reported. The second row shows frames from shot B entering the window. The ranking process determines that 7 of the most similar 10 frames are pre-frames. Compared to the first row, the pre-frame count is larger because the frames from shot B are less similar to the current frame, and are therefore ranked below all

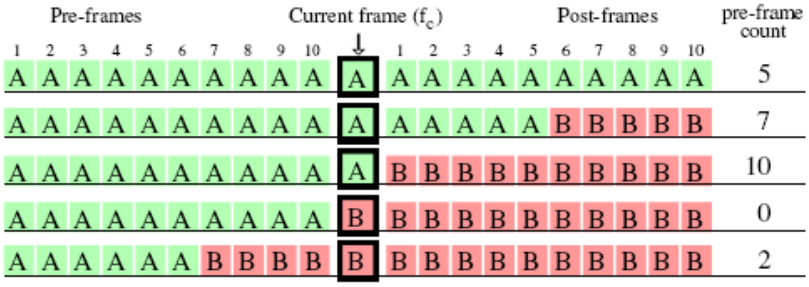


Figure 2.6: Moving query window with a half-window size of 10

frames from shot A. The pre-frame count is not near zero, and a cut is not reported. In the third row, the current frame is the last in the first shot. The ranking determines that the pre-frame count is at the maximum value of 10, since all post-frames are ranked below all pre-frames. Since this is not near zero, a cut is not reported. The fourth row shows the case where the current frame is the first in shot B. Here, the post-frames are all more similar to the current frame than the pre-frames are, so the pre-frame count is 0. The algorithm reports a cut. The final row shows what happens as the frames from shot B enter the pre-frame half-window. Some of the pre-frames are now similar to the current frame, and so the pre-frame count increases to 2. Since a cut was reported for the previous frame, another one is not reported here.

The general trend of this ranking approach is when frames of only one shot are present in the moving window, the ratio of pre-frames to post-frames ranked in the top $\frac{|C|}{2}$ frames is typically 1. As a new shot enters the post-frames of the window, this ratio increases. When the first frame of the new shot becomes the current frame, the ratio rapidly decreases. Then, as the new shot enters the pre-frames, the ratio stabilises again near 1. This algorithm assumes that the value of the pre-frame count falls from near $\frac{|C|}{2}$ to 0 within a few frames. Further, there can be significant frame differences within a shot, so it has to be specified that the pre- and post-frames spanning a cut must be reasonably different. For this, two empirically-determined thresholds are used. It is required the last pre-frame and the first post-frame to have a difference of at least 25% of the maximum possible inter-frame difference. It is also specified that the average difference between $f_c$ and the top $\frac{|C|}{2}$ frames must be at least half the corresponding value of the lower $\frac{|C|}{2}$ frames.

The overall results of the algorithm presented in [15] show that the approach finds 19 out of 20 cuts on the given video test set, and only 1 in 10 cuts that are detected are false alarms. The recall and precision values

are in most cases around 90%. The frame comparisons were performed with one-dimensional color histograms. The best results were found with the half window size value set to between 6 and 10 frames. Small window sizes are preferable as they minor the amount of computation required. However, a very small window size increases the sensitivity to frame variations within a shot, thereby increasing false alarms. The final conclusion is that this approach both increases output quality and reduces computational costs during the shot detection operation.

### 2.4.3 Algorithm 2 - Support Vector Machine classification

The next shot boundary detection approach to be described is presented in [16]. The algorithm makes use of **Support Vector Machine** (SVM) classification. SVMs are used in data modeling, and combine generalization control with a technique to address the curse of dimensionality. The kernel mapping provides a unifying framework for most of the commonly employed model architectures, enabling comparisons to be performed.

The first step of the algorithm is to extract color histograms from the video frames. Global frame histograms and block histograms are extracted. The block histograms use a $4 \times 3$ uniform spatial image grid, as shown in figure 2.7.

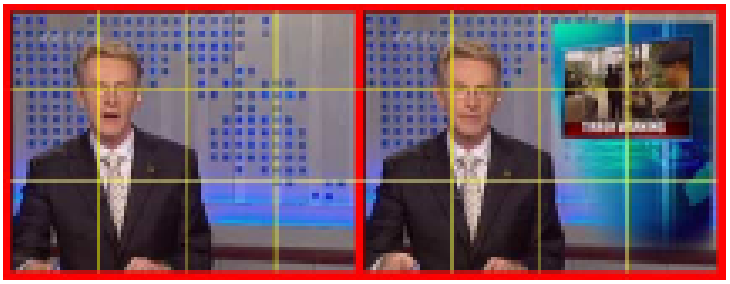

Figure 2.7: A $4 \times 3$ uniform spatial image grid

Then, a **Blocked Color Histogram** (BCH) can be denoted by vector $V_H$:

$V_H = [v_{H1}, v_{H2}, ..., v_{H13}]$

All the BCHs corresponding with frames are jointed, and a matrix $M_H$ containing all the vectors is constructed:

$$M_H = \begin{bmatrix} V_{1H1} & V_{1H2} & ... & V_{1H13} \\ V_{2H1} & V_{2H2} & ... & V_{2H13} \\ ... & ... & ... & ... \\ V_{NH1} & V_{NH2} & ... & V_{NH13} \end{bmatrix}$$

The wavelet coefficients of the column vectors in $M_H$ are calculated, and the coefficients matrix $M_W$ constructed:

$$M_W = \begin{bmatrix} V_{1W1} & V_{1W2} & ... & V_{1W13} \\ V_{2W1} & V_{2W2} & ... & V_{2W13} \\ ... & ... & ... & ... \\ V_{NW1} & V_{NW2} & ... & V_{NW13} \end{bmatrix}$$

Each column vector of $M_W$ is the wavelet coefficient series of each block in frames. By analyzing them, the transition conditions of blocks can be found.

Since shot transitions are temporal random processes, the features of SVM must have temporal processes too. A moving temporal window is set in the wavelet coefficient vectors. The transition processes of shots are captured, and the noise of video is wiped off. It is pointed out in [16] that it is impossible to set a uniform function to resolve the problem of threshold setting. The most important problem of adopting BCH is to set the weight of each block. The system adopts SVM to resolve this problems upon. A 3-class SVM is constructed, where the features of SVM include all the wavelet coefficients in the temporal window. The algorithm can be described as, given training vectors:

$T = \{(x_1, y_1), ..., (x_l, y_l) \in (SWV, Label)\}^l$,
$x_i \in SWV, y_i \in Label = \{1, 2, 3\}, i = 1, ..., l$

The decision function of classifier i-j is:

$$f^{ij} = \begin{cases} i, g^{ij}(x) & , > 0 \\ j & , others \end{cases} \tag{2.22}$$

$(i, j) \in \{(i, j) | i \leq j, i, j = 1, 2, 3\}$

$g^{ij}(x) = \sum_{n=1}^{l} y_n \alpha_n^{ij} K(x, x_n) + b^{ij}, n = 1, ..., l$

where $l$ is the number of **Sliding Window Vectors** (SWVs). 3 classifiers are constructed, and each one trains data from two different classes. Each binary classification is considered to be a voting where vote can be cast for all data points x. When the results from the SVM classification are joint, a series of labels are returned. These indicate NF for Normal Frame, CF for Cut Frame and GT for Gradual Transition Frame. Then CUTs and GTs can

be detected. The r-frames are extracted as a by-product of multi-resolution analysis for shot boundary detection. This algorithm approach offers three kinds of r-frames:

- *(1)* The minimal points of wavelet coefficients in shots are chosen to represent r-frames
- *(2)* The maximal points of wavelet coefficients in shots are chosen to represent r-frames
- *(3)* In long shots, a series of local minimum points are chosen to represent r-frames

The conclusion of the article is that the testing result of the experiment shows that the new method has good accuracy for the detection of shot boundaries. It basically resolves the difficulties of detection caused by sub-window. The framework also greatly improves accuracy of gradual transitions of shot.

### 2.4.4 Algorithm 3 - Fusion of multiple cues

In [11], an interesting approach is presented. The main point of the article is to find a solution to the conflicting evidence provided by the different features in video frames. It is pointed out that, since there is a strong correlation between the features, it is not easy to fuse the information from the features to make interpretations. The goal of the article is to develop a framework to address the problem of information fusion when the features are noisy and highly correlated. This is done by presenting a method based on **Bayesian networks** that models the dependence between the segmentation decision and the different features. A major advantage related to this framework is the possibility to select the best set of techniques that are sufficient to make reliable and robust decisions for a given class of video data.

An important thing related to video shot boundary detection is that although the mechanisms do not perform well for all situations, they perform well in a subset of the situations. The approach in [11] is to make use of a combination of four distinct methods. The first set of features is based on the color distributions in each frame of the video, as described in subsection 2.3.3. The second feature set is based on the response of each frame to a set of texture filters, also described in subsection 2.3.3. The third feature set scores the frames for a segmentation boundary based on the tracking of significant point features in the video. The features are selected from the frames and tracked across the video. A score is assigned to the tracking of the features, and is used to evaluate the segmentation boundaries. The score is computed by weighting the contribution of each feature that

is tracked from the last frame to the current frame. The weight for the $i^{th}$ feature is computed as:

$$w_i = 1 - e^{\frac{-p_i}{k}} \tag{2.23}$$

where $p_i$ is the number of frames in the past through which the $i^{th}$ feature was tracked and $k$ is the same constant that determines the sensitivity to the history of the tracks. To compute a distance measure across the segment boundary, the difference between the average track scores in windows on either side of the candidate boundary frame is computed. At frame $i$, a window of size $S$ is selected around the frame and the average track score is calculated to the left and right of the frame. The difference between the average track scores and the fraction of missed tracks is used as the distance measure of the tracking module. The fourth feature class computes the likelihood of a segmentation boundary by detecting edges in the spatiotemporal volume that represents the video. Video data is three dimensional data where the temporal dimensions is the third dimension. It is possible to make a projection detection filter to detect cuts. Planes parallel to the $x-t$ and the $y-t$ planes are used. Edges perpendicular to the $t$ axis in frame sections indicate possible video segment boundaries. This is shown in figure 2.8. The figure is taken from [11].

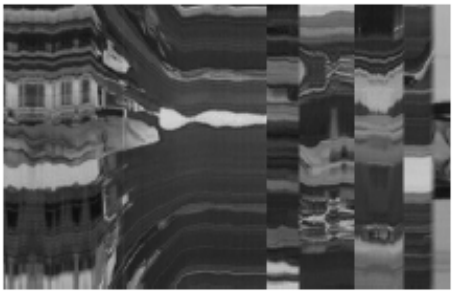

Figure 2.8: x - t section through the spatiotemporal video volume

The fraction of the pixels at any $t$ covered by the horizontal edges is taken as a measure of the segment boundary. Averaging this measure across many sections gives a probability measure of the existence of a segment boundary based on the evidence from edges in spatiotemporal volumes.

The segmentation is computed by comparing the change across the candidate segment boundary. The change can be measured as a distance $D = F(S_{b-\epsilon}, Sb+\epsilon)$ between the video features, $S_{b-\epsilon}$ and $S_{b+\epsilon}$, in the two temporal intervals $[b-\epsilon, b]$ and $[b, b+\epsilon]$ around the boundary $b$. The distance measure is then compared with a threshold value to determine if there exists a boundary at $b$. The different approaches select different properties $S_{b-\epsilon}$

and $S_{b+\epsilon}$ to represent the intervals and the function that evaluates the distance. The challenge is then to find fusion techniques that either use the $D$ from each module for making the fusion decision or the individual decisions from each module to make the fused decision.

The fusion process is carried out by using a Bayesian network. A Bayesian network over a set of variables $\mathbf{X} = \{X_1, ..., X_n\}$ is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathbf{X}$. Formally, a Bayesian network is a pair $B =< G, L >$. The first component, $G$, is a directed acyclic graph whose vertices correspond to the random variables $X_1, ..., X_n$, and whose edges represent direct dependencies between the variables. The second component, $L$, represents a set of local conditional probability distributions $L_1, ..., L_n$. A Bayesian network $B$ defines a unique joint probability distribution over $\mathbf{X}$ given by the product:

$$P_B(X_1, ..., X_n) = \Pi_{i=1}^{n} L_i(X_i|pa(i)) \tag{2.24}$$

where $pa(i)$ denotes the set of parents of $X_i$ in $G$. In [11], it is assumed that there is one variable $A_i$ for each feature, and a distinguished variable *Outcome* that can take value from the set $\{0, 1, 2\}$ depending on whether the frame is *normal*, a *boundary* or a *flash*. The objective is given a set of vectors $\mathbf{X} = \{A_1, ..., A_n, Outcome\}$, to induce a probability distribution $Pr(A_1, ..., A_n, Outcome)$ from this data in the form of a Bayesian network. The dependence among attributes in a Bayesian network will be represented via a tree structure. An example of this is shown in figure 2.9. The figure is taken from [11].

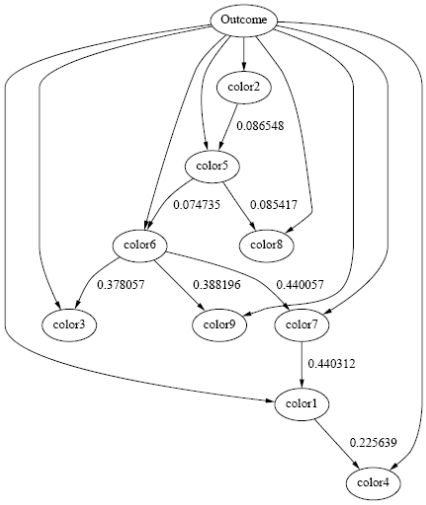


Figure 2.9: Bayesian network using color features

In such a network as shown in 2.9, an edge from $A_i$ to $A_j$ implies that the influence of $A_i$ on the assessment of *Outcome* also depends on the value of $A_j$. These kinds of networks are learnable in polynomial time, which is an attractive property.

The results in [11], after experimenting with Bayesian network model induction and segmentation, were very encouraging. Few boundaries were missed, although the number of false positives were large. Since the framework developed was very general and in an early phase, it is important to notice the positive results among with the possibility to extend the model with more sophisticated methods and decision computations.

### 2.4.5 Algorithm 4 - A feature based algorithm

A new shot detection approach presented in [17] is focused on avoiding false shot detections because of camera motion or moving objects that make a large amount of pixels change their values from frame to frame. The method is using the observation that during a cut or dissolve in video, new intensity edges appear far from the locations of old edges. Further, old edges disappear far from the location of new edges. An edge pixel that appears far from an existing edge pixel is defined as an **entering** edge pixel, and an edge pixel that disappears far from an existing pixel as an **exiting** edge pixel. By counting the entering and exiting edge pixels, cuts, fades and dissolves can be detected and classified.

The algorithm presented in [17] takes as input two consecutive images $I$ and $I^{'}$. An edge detection step is first performed, resulting in two binary images $E$ and $E^{'}$. $\rho_{in}$ denotes the fraction of edge pixels in $E^{'}$ which are more than a fixed distance $r$ from the closest edge pixel in $E$. $\rho_{in}$ measures the proportion of entering edge pixels. A fade-in, cut or an end of a dissolve should assume a high value of $\rho_{in}$. $\rho_{out}$ denotes the fraction of edge pixels in $E$ which are farther than $r$ away from the closest pixel in $E^{'}$. $\rho_{out}$ measures the proportion of exiting edge pixels. It should assume a high value during a fade-out, cut or at the beginning of a dissolve. The basic measure of dissimilarity is

$$\rho = max(\rho_{in}, \rho_{out}) \tag{2.25}$$

This represents the fraction of edges that have entered or exited. Scene breaks can be detected by looking for peaks in $\rho$, which in [17] is termed as the **edge change fraction**. An example of this is shown in figure 2.10. The cut between frames #9-#10, the dissolve in frames #25-#35 and the fade out starting at frame #55 are shown as clear peaks in the $\rho$ value. The figure is taken from [17].

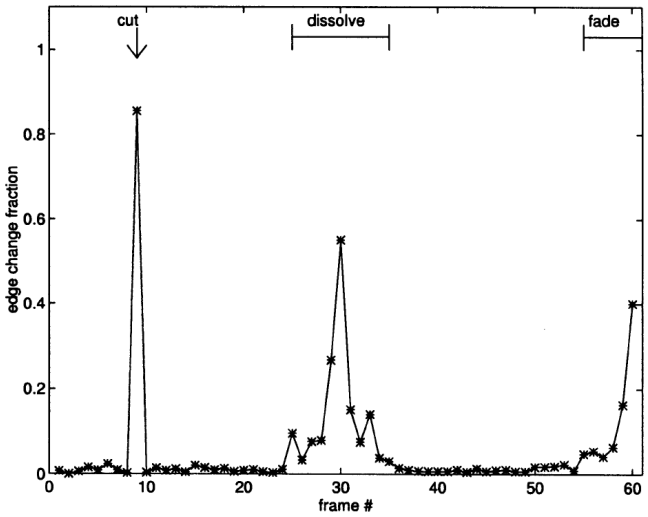


Figure 2.10: Example of peaks in the edge change fraction value

The first step in the algorithm is edge detection. At first, the image is smoothed by convolving it with a Gaussian of width $\sigma$. Next, the gradient magnitude is computed. This value indicates how fast the local intensities are changing. Copies of $E$ and $E^{'}$ are created with each edge pixel dilated by a radius of $r$. The copies are named $\overline{E}$ and $\overline{E'}$. The equation for $\rho_{out}$ is

$$\rho_{out} = 1 - \frac{\sum_x^y E[x+\delta x, y+\delta y]\overline{E'}[x,y]}{\sum_x^y E[x,y]} \tag{2.26}$$

which is the fraction of edge pixels which are exiting. The equation for $\rho_{in}$ is

$$\rho_{in} = 1 - \frac{\sum_x^y \overline{E}[x+\delta x, y+\delta y]E'[x,y]}{\sum_x^y E[x+\delta x, y+\delta y]} \tag{2.27}$$

The edge change fraction $\rho$ shown in equation 2.25 is the maximum of these two values.

The nest step of the algorithm is to detect scene breaks by looking for peaks in the edge change fraction $\rho$. When a peak is located, it is necessary to classify it as a cut, dissolve or fade. A cut is detected by the fact that is the only scene break that occurs entirely between two consecutive frames. This will lead to a single isolated high value of $\rho$. During a fade in, $\rho_{in}$ will be much higher than $\rho_{out}$, since there will be many entering edge pixels and few exiting edge pixels. At a fade out, the opposite is the case, and $\rho_{out}$ will be much higher than $\rho_{in}$. A dissolve consists of an overlapping fade in and fade out. During the first half of the dissolve, $\rho_{in}$ will be greater, but during the

second half $\rho_{out}$ will be greater.

The algorithm has been tested on a number of image sequencies, containing various scene breaks. Intensity histograms have been used to detect edge change fractions. The sum of the histogram differences used is given by

$$\sum_{i=0}^{N-1} |H_t[i] - H_{t+1}[i]| \tag{2.28}$$

where $N$ denotes the number of histogram buckets, and $H_t$ denotes the intensity histogram of the t'th frame. 50 MPEG movies were used for testing. The results show that 115 out if 118 cuts were detected. 17 false positives were detected. If focusing on performance, nearly linear speedup were obtained, because this approach makes use of parallel processing. The weakness of the algorithm mentioned in [17] is involving failure in the case of handling rapid changes in overall scene brightness, or scenes which are very dark or very bright. These may lead to false positives.

## 2.5 R-frames selection

When the shot boundary detection algorithm has found all the boundaries in the video file, r-frames selection is the next task. It is important to find frames that fully describe the video content, as well as not picking too many frames because of the storyline's length in the browsing window. It is also important to present the r-frames in a way that describes the video content in a well arranged way.

### 2.5.1 Selecting the right r-frames

After the video file has been transformed into one long series of frames, there are several methods to pick r-frames for video shot descriptions. To reduce the final number of r-frames, the best solution is to pick only one r-frame from each shot. The drawback of this is that if the quality of that one chosen r-frame is poor, there is no other r-frame to fully describe that particular shot from the video file. On the other hand, no matter what method is chosen, there is never guaranteed that the quality is good enough.

One simple method is to choose the first frame after the shot boundary, and use that frame as an r-frame for the particular video shot. This method is computationally cheep, as the r-frames can be picked immediately after the shots have been detected. But it is impossible to evaluate the picture quality without further operations.

Another method is to choose the frame in the middle of the shot as an

r-frame. The algorithm keeps control on how many frames each shot contains, and then picks the middle frame. The result is much the same as the previous solution, with no control on output quality.

One final solution is to evaluate the content of the frames, and choose the one frame with the best content description as an r-frame. This method demands more computational resources, but the output quality is almost certain to be good. It is hard to decide what quality criteria to use, but it is possible to make a histogram middle value for the entire shot, and then pick the frame nearest that value. Frames with poor quality and other major differences will not be chosen, and the shot content will be well described based on that one r-frame.

### 2.5.2 Redundancy avoidance

One problem with automatic video shot detection algorithms is the possibility of getting too many r-frames as output. If one r-frame is picked from each shot in the video file, and the video contains dialogues, the r-frame summary will contain repeated similar r-frames [18]. As an example, there are 300 cut-points in a 15-minute segment of the movie "Terminator 2". The result is 2400 r-frames to be displayed to browse a two-hour move. From a user's point of view, this will be far from acceptable. The solution proposed in [18], is to effectively reduce the number of redundant r-frames extracted from long video sequences by detecting similar r-frames, and link them together. Table 2.1 shows the definition of terms used.

| *Term* | *Explanation* |
|---|---|
| DC-picture | A $90 \times 60$-subsampled still image |
| Cut | The moment when the camera changes |
| Shot | A series of frames divided by cut-points |
| Pattern | A group of shots with similarity in camera, location and angle |
| Similar shots | Two shots which are members of the same pattern |
| Act | A sequential group combining several patterns |

Table 2.1: Definition of terms

Before the search for similar parts can start, the video stream is preprocessed with the following steps:

- *1:* Subsample video stream
- *2:* Detect cuts
- *3:* Compare shots
- *4:* Link similar shots

- *5:* Confirm similarities
- *6:* Label patterns
- *7:* Detect acts

The subsampling part involves extracting still images out of the video stream. Each frame is divided into units of $8 \times 8$ pixels in units called blocks. The blocks are encoded by the DCT method mentioned in section 2.2. The result is a subsampled version of the original picture, but the number of pixels is reduced to 1/8. The pictures are named DC pictures, and used to detect cuts in the video file.

Further, two shots are compared to see whether they are similar shots. Several DC-pictures from each shot are chosen, and the differences for all pairs calculated. When a pair of similar DC-pictures is01 found, the two shots are linked. The operation is a matter of solving three different problems:

- *1:* Which DC-pictures should be selected when comparing the shots?
- *2:* How many DC-pictures should be sampled from each shot?
- *3:* How should the DC-pictures be compared?

The hypothesis in [18] is that it is very difficult to find a general formula for which pictures to select, as the first picture in one shot can be similar to the last picture in another. By experimenting with input parameters, the best number of DC-pictures to select from each shot was 8. To compare the DC pictures, two different methods are used. The first method is **histogram comparison** based on the picture's color values, as described in section 2.3.3. Since this method occasionally detect false matches, as shown in figure 2.3, another method is used to properly distinguish differences. This method is named **layout analysis**. Here, the image formed by the pixel-wise difference between luminance images is analyzed. A similarity counter adds up the number of pixels below a certain threshold. If the similarity counter is large, two frames are declared to be similar.

When the similar shots are detected, the next step in the process is to group similar shots and detect patterns in the groups. To avoid two different patterns to be grouped, a link certainty variable $C$ is used. It estimates the certainty of the similarity process. A similarity variable for shot $k$ and shot $l(S_{kl})$ is set to 1 when shot $k$ and shot l are evaluated as similar, and to 0 when they are not. A certain algorithm is used to calculate $C_{kl}$, and if it has a value larger than a given threshold value, shots $k$ and $l$ are determined to be linked. As a result, similar shots are grouped into patterns.

The final step of the operation is act determination. The concept of an act is useful in arranging key-frames more clearly on a display terminal. In [18], several views have been tested. One possibility is to omit all redundant r-frames, and only show one frame from each act. Another view is to arrange frames vertically in temporal order, with r-frames in the same act displayed horizontally.

The conclusion in the article is that the proposed method is efficient in most cases, and reducing the number of r-frames by more than half.

Another approach for redundancy avoidance is presented in [19]. The main point in this method is to filter out the blank unicolor frames and repetitive frames. When an r-frame is detected by the video cut detection process, a frame signature is derived from the composition of the DCT coefficients. Blocks with similar signatures are compared to determine size and location of groups of blocks in order to derive region signatures. After deriving block signatures for each frame, regions of similarly valued block signatures are determined. Regions consist of two or more blocks which share similar block signatures. Each region is then assigned a region signature:

$$Region(Block\_signature, region\_size, Rx, Ry) \tag{2.29}$$

Rx and Ry are the coordinates of the center of the region. Each region corresponds roughly to an object in the image. Since a frame is represented by a number of regions, the similarity between frames can be regulated by choosing the numbers of regions that are similar, based on their block signature, size and location.

Block signatures can be used to remove unicolored frames. The color composition of the frame is checked, and if the frame is composed of mostly similar DC values, it is filtered out as a unicolor frame. Alternatively, after the region signatures are computed, the frame is unicolor if one of the regions takes up to 90% of the frame size.

To filter out repetitive frames, a frame comparison procedure compares a current r-frame $F^{"}$ with all r-frames $F^{'}$ in a list of past r-frames. Their respective region signatures are compared using the absolute value of the region size differences:

$$\begin{aligned} frame\_difference = \\ |reg\_s_1^{'} - reg\_s_1^{"}| + |reg\_s_2^{'} - reg\_s_2^{"}| + |reg\_s_3^{'} - reg\_s_3^{"}| \end{aligned} \tag{2.30}$$

where $reg\_s_i^{'}$ and $reg\_s_i^{"}$ is the region size of the region $s_i$ in the frame $F^{'}$ and $F^{"}$ respectively. If the frame difference is greater than a certain

threshold, then the frames are considered similar, and the current frame is not added to the table of contents. If the frame difference is smaller than the threshold, then the frames are considered different enough and the current frame is kept in the visual table of contents. In addition to the region size, the frames are compared based on the position of the region centroids.

### 2.5.3 R-frames presentation

When the r-frames are chosen, they have to be presented in a way that shows the content of the video file in a most descriptive way. There are several ways to organize this view, and no answer exists to decide which is best suited. The first solution is to show all the selected r-frames in one horizontal storyline, with the r-frames sorted in ascending order from left to right. This view is shown in figure 2.11. The drawback of this view is that it is difficult to navigate in the storyline, as the segmentation is only on frame level. The number of frames may be large, and the user has to look through the whole line to get a view on the content.

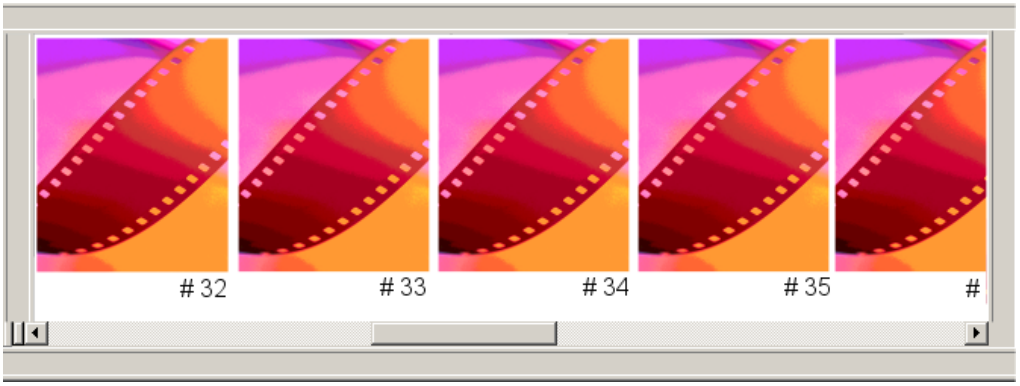


Figure 2.11: Horizontal ascending view

The next solution is to make use of both horizontal and vertical organisation. The scenes from the video may be sorted in ascending order in the vertical direction, and the r-frames belonging to the scenes in ascending horizontal order. If several frames are chosen from each shot, it is also possible to use shot organisation instead of scenes. An example of this view is shown in figure 2.12. It is also a great advantage to have a bar in the bottom of the window which shows scene/shot duration and relative placement in proportion to the entire video. Further functionality may include the possibility to play one particular scene or shot by clicking on a belonging r-frame.

The final solution discussed in this section is the hierarchical view, with a possible tree structure as illustration. The root node may be the entire video, which is divided further down into scenes, shots and r-frames. A suggestion of this kind of view is shown in figure 2.13. It is very easy to navigate in

Figure 2.12: Horizontal/Vertical view

this structure, and possibilities to play scenes and shots may be included.

Figure 2.13: Hierarchical view

## 2.6 Speeding up the algorithms

Methods like comparing corresponding pixels in images, color grayscale histograms or changes in edges are all referred to as the linear shot boundary detection approach, due to the fact that they sequentially measure inter-frame differences and study their variance values. Such sequential searching is an expensive operation, as the number of frames to be examined is very large. It is important to find efficient solutions to speed up the shot detection algorithms, without loosing information on the way. In [20], they have experimented with a non-linear approach in which most frames do not need to be compared.

The idea is motivated by two facts. First, there is little difference between consecutive frames within a shot. Thus, most comparisons under linear techniques are wasteful. Second, frames do not have to be examined in the order they appear in the video clip. It is then possible to develop search techniques to skip over unnecessary comparisons much in the same way that binary search skips over data items which have no chance of being matched. Two different approaches have been tested in [20], named **regular skip** and **adaptive skip**.

In regular skip, every other $d^{th}$ frames are compared. If d=2, the first frame is compared with the third, the third with the fifth, and so forth. When two frames $i$ and *i+2* are identified to be in two different shots, frame $i$ and *i+1* are compared to determine the shot boundary. If frame $i$ and frame *i+1* are identified to be in two different shots, the shot boundary lies between frame $i$ and frame *i+1*. Otherwise, it lies between frame *i+1* and frame *i+2*. Frame *i+1* does not have to be compared with frame *i+2*, since the approach assumes that a shot has at least two frames. This assumption is generally true in practice. Once a shot boundary has been identified, the same procedure is then repeated for the next shot starting from its first frame. This method reduces the number of comparisons almost in half.

In the regular skip method, the value of $d$ is static throughout the operation. But the optimal value of this variable varies from video to video. In the adaptive skip strategy, $d$ is determined dynamically. In each iteration, the algorithm determines how many frames to skip by comparing the current frame with the one last examined. If this comparison is more similar than the last comparison, $d$ is increased for the next comparison. If it becomes less similar, $d$ is decreased. If the current comparison indicates that the two frames are in two different shots, the algorithm scans backward to look for the boundary using regular skip in the reverse direction. Once the boundary has been determined, the scan continues forward again using the same procedure.

Figure 2.14 illustrates the methods discussed in this section. The figure is taken from [20]. 2.14(a) shows a given video clip with three shots. 2.14(b) shows the regular linear approach, where every two consecutive frames are compared. The computational cost $N_c^{linear}$ for this method can be calculated as $F - 1$, where F is the total number of frames in the video. In this example $F$ is 50, so $N_c^{linear} = 49$. 2.14(c) illustrates the regular skip approach. The odd numbered frames are compared, and when two frames are identified to be in two different shots, one additional comparison is necessary to determine the shot boundary. Thus, the number of comparisons required by regular skip can be computed as $N_c^{regular} = \lfloor \frac{F}{2} \rfloor + s$, where $s$ is the number

of shot boundaries in a given video. This example video has 50 frames and two shot boundaries, and $N_c^{regular} = 27$. This represents 45% saving over the linear approach. With the adaptive skip method, the value of $d$ is 2 in the beginning. Frame # 1 and frame # 3 are compared. Both frames are in shot 1, and the value of $d$ is increased to 3. Frame # 3 and frames #6 are compared next. The frames are still in shot 1. The value of $d$ is increased, and the process continues until $d = 6$ and frame # 15 and frame # 21 are compared. Since these frames are in two different shots, a backward scan is required to find the shot boundary between frame # 17 and frame # 18. For the entire example video, $N_c^{adaptive} = 20$, which is a 59% improvement over the linear approach.

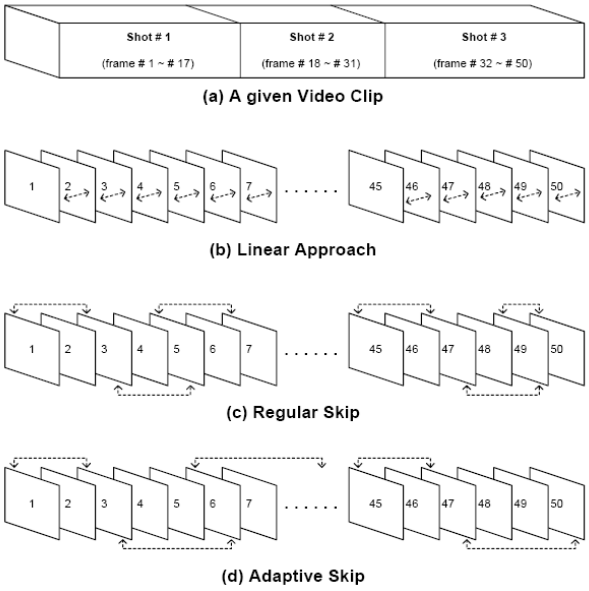


Figure 2.14: Non-linear shot boundary detection strategies

Experimental results presented in [20] show that it is possible to save a great amount of resources with the non-linear approach methods discussed in this section, both regarding the total time for execution and the total number of frame comparisons. In average, regular skip reduces the costs of the linear technique in half, while the adaptive skip method saves more than 80%. The exact savings depend on the lengths of the shots in the video. The savings are greater for videos with longer shots. The best result presented shows that adaptive skip is more than 16 times better than the linear approach.

# Chapter 3

# Requirements analysis

This chapter describes requirements related to the process of designing and testing a prototype for video shot detection. This involves requirements for the video test material, the shot detection algorithms and how to present the results in the end of the segmentation process.

## 3.1 Video test material requirements

When designing and testing different algorithms for shot detection, it is important to have video test material that covers a wide area of different domains, situations and genres. The algorithms have to be tested and evaluated based on the results from all the different test videos. A specific algorithm may be the best in one genre, and completely useless in others. For the same reason, it is also important to have great variations related to colors, length, light settings and fade-ins/outs in the video test material.

## 3.2 Algorithm requirements

The shot detection algorithms have to be designed independent of the test video format, content and genre. It should be possible to choose algorithm and threshold values from the graphical user interface.

## 3.3 User interface requirements

The best way to design the user interface is to present all the functionality in the same view, and simultaneously keep it simple and well arranged. This interface can be divided into two different regions, the algorithm test bench view and the r-frames presentation view.

### 3.3.1 Algorithm test bench view

This view shall present a way to choose which segmentation algorithm to use, and which threshold values that have to be exceeded to define a cut in the video file. In addition to this, there has to be a possibility to open a video file from an internal or external disk and functionality to start the segmentation process.

### 3.3.2 R-frames presentation view

This view shall present one r-frame from each shot found in the input video file, in the most tidy and perspicuous way. It is desirable that the user has the possibility to start the video from a specific point by clicking on the appurtenant r-frame.

# Chapter 4

# Construction

This chapter describes the construction of a video shot detection prototype. This involves the shot detection algorithms, the graphical user interface and the video test material.

## 4.1 Algorithm construction

The main idea regarding the construction of a video shot detection prototype is to test different algorithm approaches, and hopefully find the most efficient algorithm in the end. It is important to build the different algorithms from different view points, and approach the problem from different angles. This section will give an overview of the algorithm construction part, and explain why and how the different algorithms are built.

### 4.1.1 Global color histograms algorithm

The first algorithm to be implemented for testing is called the *Global color histograms algorithm*. The main idea with this algorithm is to treat every frame picked out as one unit, and build one color histogram for each image. This is illustrated in figure 4.1, where the red arrows include the whole image inside the region of interest.

The global color histograms is built up by defining 4 regions of equal size, spanning from the minimum value 0 to the maximum value 256. The first region covers the values from 0 to 63, the second covers the values from 64 to 127, the third covers the values from 128 to 191 and the fourth covers the values from 192 to 255. It is possible to use smaller regions as well, but this will in the end result in computational slow-downs of the algorithm. The color scale is illustrated in figure 4.2.

For every pixel of the image, the **R**, **G** and **B** values are detected and placed into one of the 4 regions. During the histogram comparing process, the number of pixels within each color band and region are counted for the

Figure 4.1: Region of interest in the global color histogram algorithm

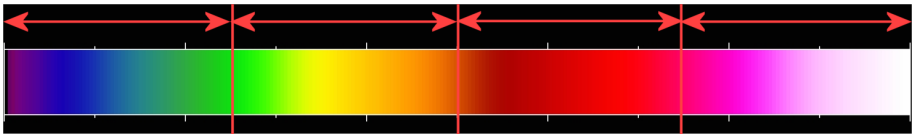

Figure 4.2: The image RGB color scale

two histograms that are compared. The sum is stored and compared in the end. Great differences from one histogram to another may indicate a cut in the video. The sum is compared to a fixed threshold value, and a cut is defined if the difference is greater than the threshold value. The image that constitutes the foundation of the given histogram is written out to be shown as an r-frame in the end of the segmentation process. The weakness of this approach is that great differences in pixel values may well occur without a cut in the video. One well-known example is a sudden flash of sun light in one frame within a shot. Another weakness is that this method probably will not detect a cut that gradually takes place, as the overall pixel difference from one frame to the next one changes little during a fade-out or fade-in.

### 4.1.2 Local color histograms algorithm

The next algorithm is called the *Local color histograms algorithm*. The same regions of color and color scale are used here, as in the global approach explained in the previous subsection. The difference is that this algorithm divides the image into small regions of interest, instead of looking at the entire image. This is illustrated in figure 4.3, where each square is one independent region.

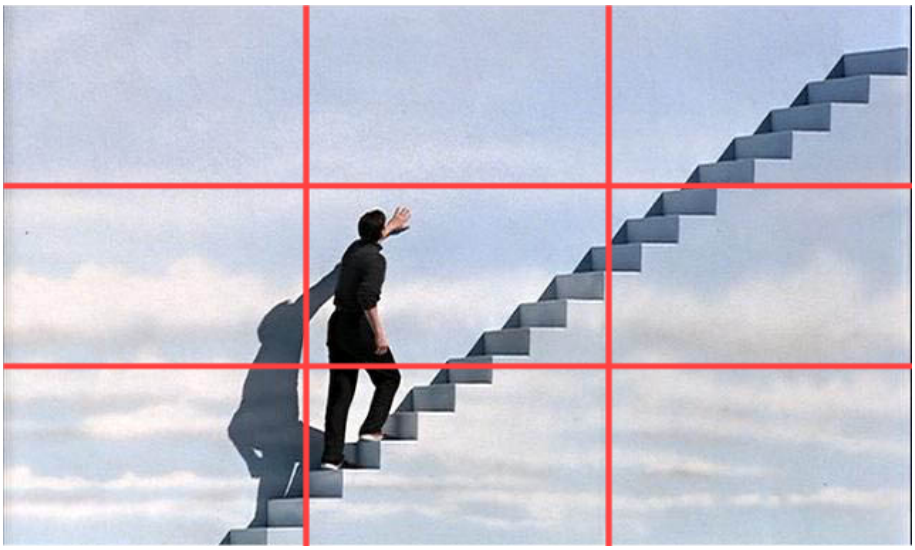

Figure 4.3: Regions of interest in the local color histogram algorithm

The histograms are generated in exactly the same way, but one distinct histogram is created for each region. This algorithm is designed with 9 regions, which figure 4.3 shows. In the histogram comparing process, corresponding regions are compared. That is; region 1 from histogram 1 is compared with region 1 from histogram 2 and so on. A cut is defined if the pixel values from more than a given number out of the 9 regions exceed

a fixed threshold, and the belonging frame is written to the GUI as an r-frame. This given number may be changed during the testing phase. The same anticipated weaknesses exist for this algorithm as the global approach, in addition to the growing computational costs regarding the histogram generation and comparing processes. The advantage is that the shot detection hopefully will be more precise, as the algorithm has increased control over exactly where in the image the differences occur.

### 4.1.3 Global edge chasing algorithm

The third algorithm is called the *Global edge chasing algorithm*. The main point of this algorithm is to detect sharp contrasts within the images, and define these as edges. To illustrate this, figure 4.4 shows an original example image, and figure 4.5 shows the same image with the detected edges as white lines on the black background.

Figure 4.4: Original image before edge detection

After the edges have been detected, the local color distribution algorithm explained in the previous subsection is used to make color histograms of the 9 regions of interest that figure 4.6 illustrates.

The same procedure is used, but the histogram comparing process is accomplished a bit different. Since the major part of the image regions is black, the comparing is based on the number of non-black pixel values within

Figure 4.5: Modified image after edge detection

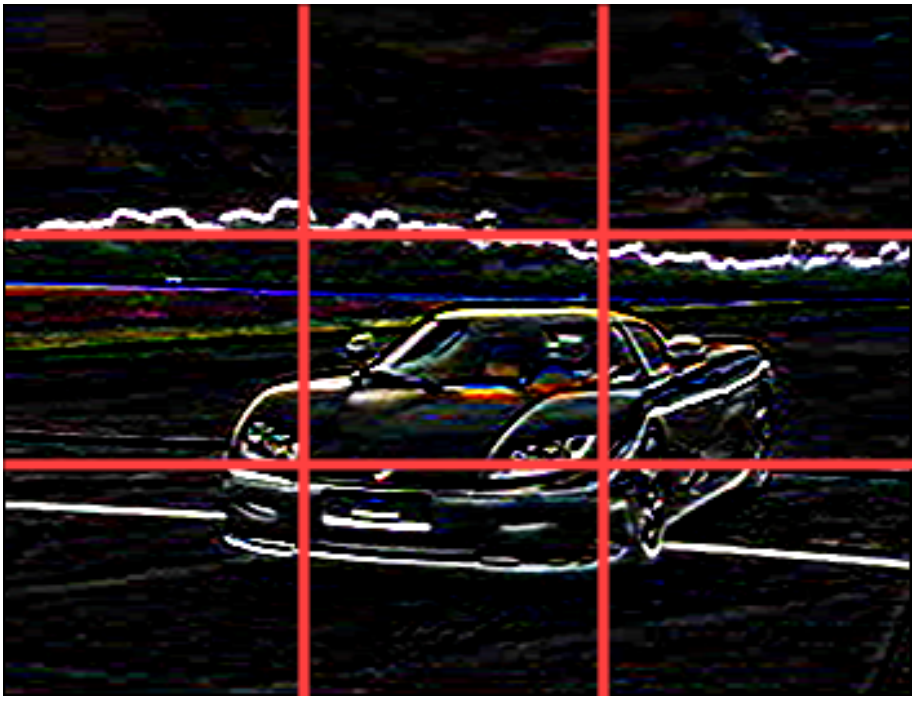

Figure 4.6: Regions of interest in the global edge chasing algorithm

each region. The threshold has to be much lower than the threshold used by the regular local color histogram comparator.

## 4.2 User interface construction

To make the user interface provide all the functionality from one window, the best solution is probably to use a frame with a menu bar to the algorithm operations, and one frame to view the output results.

### 4.2.1 Algorithm test bench view

Three menus are needed, where the first one handles the *Open file* and *Exit* functionality. A suggestion to how this menu can be designed is illustrated in figure 4.7. The *Open file* option opens a file dialog box, and gives the opportunity to open a video file for segmentation. The *Exit* option shuts down the program, included the r-frames presentation view.


Figure 4.7: Menu to open video files for segmentation

The second menu contains options for choosing which algorithm to use in the segmentation process. Informative names have to be used, so it is easy to find the right algorithm. A suggestion to how this menu can be designed is illustrated in figure 4.8. In this illustration, the algorithms just have fictive names. It is possible to run only one algorithm at the time.

The third and last menu contains options for choosing which threshold to use during the segmentation process. The option *Low* will result in a higher number of found shots than the option *Medium*, and the option *High* will result in a lower number of shots. This menu is illustrated in figure 4.9.

When a video file is opened for segmentation, the process starts immediately. The progress can be viewed in the video window, as shown in figure 4.10.

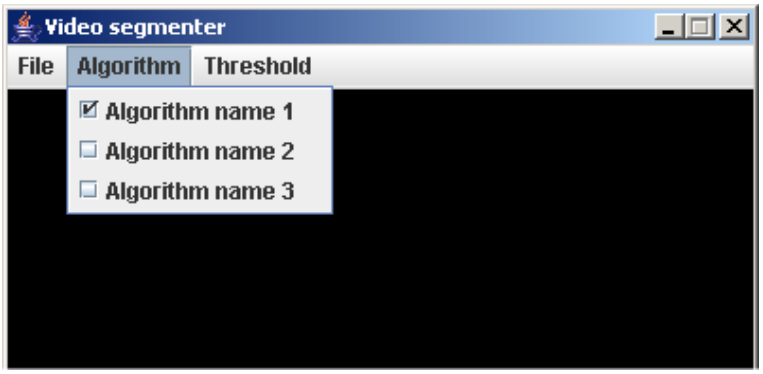


Figure 4.8: Menu to choose algorithm

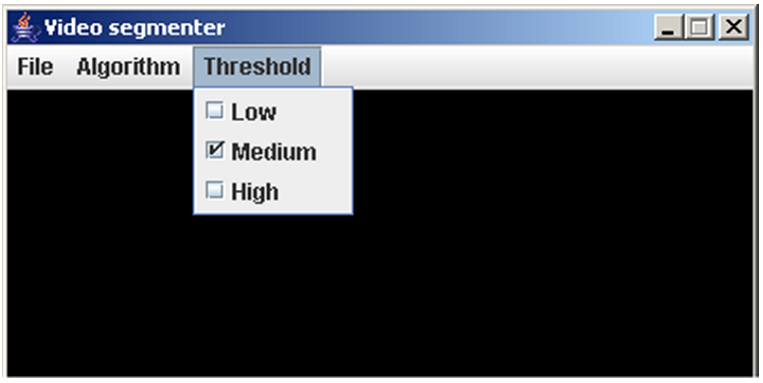


Figure 4.9: Menu to choose threshold

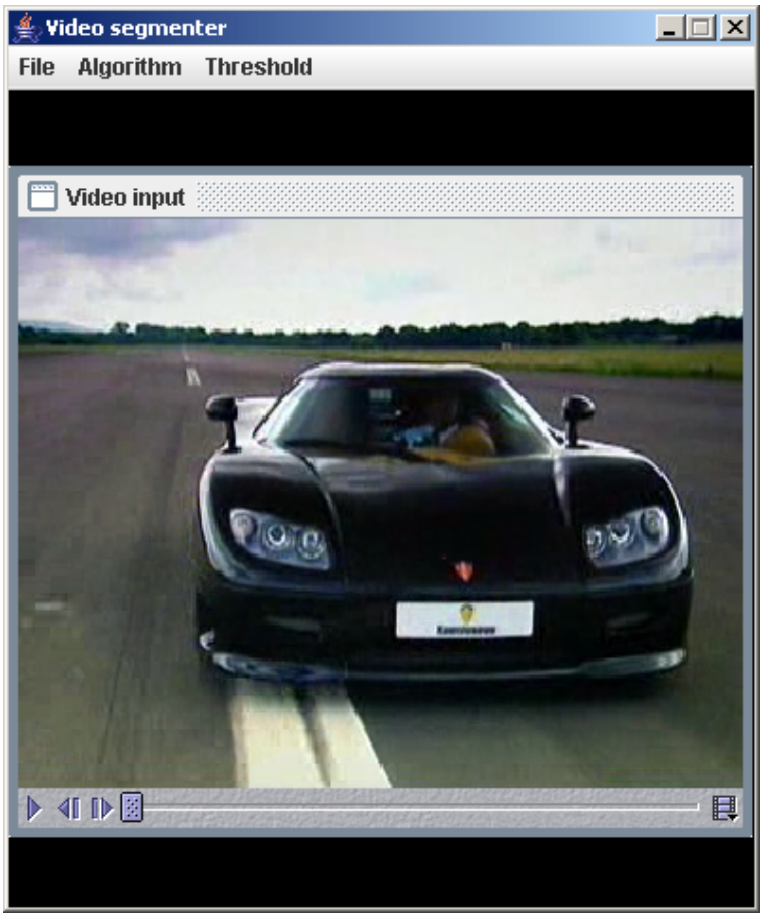


Figure 4.10: The video window

### 4.2.2 R-frames presentation view

When the segmentation process is finished, a new frame is created. One r-frame from each shot in the video is shown as an image in this frame. It is possible for the user to click on one specific image, and the video will then start at that point in the video file. The r-frames presentation view is shown in figure 4.11. In this illustration, the segmentation algorithm found four shots in the input video.


Figure 4.11: The r-frames presentation view

Since longer video files contains a great number of shots, it is necessary to have a vertical scroll bar in the presentation frame to show all the r-frames in one window. This is illustrated in figure 4.12, where the segmentation algorithm found 107 shots in the input video.

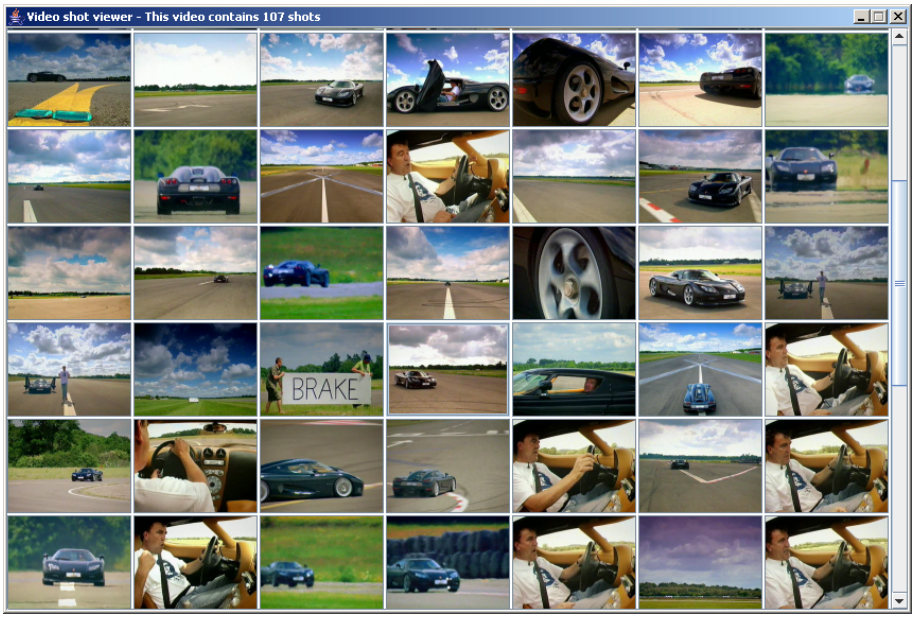


Figure 4.12: The r-frames presentation view with vertical scrollbar

## 4.3 Video test material

Since the main focus in this thesis is to find efficient shot detection algorithms, video format independency is not taken into consideration. Raw and uncompressed video material in *.avi* format is used, meaning the file size to be quite big. The increased disk activity to read these files in the segmentation process slows down the algorithms a bit, but this is minimal compared to the overall time consumption. The file size means it is difficult to use video files with duration of more than one hour. But the main point related to video file duration is to have files that last for more than a few seconds, not if it is two or three hours. Video files from different genres and situations are used to make sure the algorithms are tested thoroughly. The video test material includes the following video files:

- *Britney Spears - You drive me crazy:* Music video with a duration of 3 minutes and 17 seconds. The video contains some fade-ins and fade-outs, flashing light and bright colors. Since it is a music video, most shots in the video are short in time, so the segmentation algorithms should find a quite large number of shots. This video also contains a lot of people in motion, so it is a challenging task to detect all the shots correctly.

- *Eric Clapton - Tears in heaven:* Music video with a duration of 4 minutes and 34 seconds. This video is significantly different from the Britney Spears video. There is no people in motion, and the colors are dark with no flashing light. The main reason for using this video is because fade-ins and fade-outs are used in almost every camera change. A moving camera is also in use, and this is a great video for testing the different segmentation algorithms in these areas.

- *Metallica - The unforgiven:* Music video with a duration of 6 minutes and 29 seconds. This video is extremely dark, and contains no colors but black and white. It should be great for testing the algorithms based on color distribution. Some fade-ins and fade-outs are also used.

- *Metallica - Turn the page:* Music video with a duration of 5 minutes and 46 seconds. This video contains a mixture of long and short cuts, and shots with many and few colors. The most interesting thing about it is to see if the segmentation algorithms handle the different parts in a satisfactory way.

- *Football goal: Newcastle United - Manchester United:* Football clip with a duration of 21 seconds. This video is included in the test set to test the dominating green field's impact on the algorithm output.

- *Football goal: Manchester United - Tottenham Hotspur:* Football clip with a duration of 47 seconds. This video is included in the test set because it contains a lot of motion and changing in color values.

- *Top Gear - Koenigsegg test drive:* A clip from the british BBC car program Top Gear, with a duration of 8 minutes and 32 seconds. The clip shows a presentation and test drive of the racing car Koenigsegg CC8S. Fade-ins and fade-outs are used during the presentation of the car's design, while frequently camera switching is used during the driving sequences. Long shots occur when the test driver is talking and explaining the properties of the car. The video contains a lot of colors and sun light.

- *Saturday Night Live - Blue Oyster Cult imitation:* Satirical sketch with a duration of 5 minutes and 51 seconds. This video has low resolution and few camera switches, and the colors are not bright. The shots are long in time, and there is nearly no motion from the people involved.

- *Throlltech - Qt4 dance:* Amateur music video made by the software company Throlltech, with the duration of 4 minutes and 22 seconds. This video is included in the test set because it contains a lot of motion, and several effects are used when the camera changes.

## 4.4 Algorithm speed-up

To avoid the costly sequential operation of selecting all frames from the test videos, one variant of the skip methods described in section 2.6 has been constructed. This variant is called stepwise searching. The main idea is to reduce the number of frames to be evaluated. One fixed step size $k$ is defined. Frame number *k, 2k, 3k....* are evaluated, until a shot is detected. A sequential search is then performed within the last interval of length $k$ to find the exact cut point.

Another approach to be tested is to choose a fixed number of frames to pick per second, and skip the rest. Values to be tested are 1, 2 and 4 frames per second. The probability of missing a cut is minimal, as most shots are at least one second in length. It is important to take the overall time consumption into consideration when the output results are evaluated, as the algorithms will be slowed down by increasing the frames per second rate.

To spare time during the video segmentation process, it is important to perform most of the operations at once. More specifically, the color histogram generation and comparing can be done almost simultaneously, because the previous frame's histogram already is finished. In addition, the output from

the histogram comparing is ready in about no time, and it is already known if this point was a cut in the video or not, according to the given algorithm. In the case of the edge detection approach, this operation has to be done separately from the others and before the histograms can be generated. This will cost extra time, and it will probably be noticed during the testing of the algorithms.

## 4.5 New aspects

Video shot detection based on color histograms is not a new approach, neither is dividing each frame into regions. But the combination of global edge detection and region based color histograms is an approach that is not documented earlier. It is important to have the regular color histogram algorithms implemented to test the new approach against these methods. Both output quality and overall time consumption is important factors regarding this, in combination with which video types that produce good results.

# Chapter 5

# Implementation

This chapter describes some key points of this thesis' implementation part. Frameworks that are used will be explained, and some important code is included to illustrate essential matters regarding the algorithm implementation.

## 5.1 Java frameworks

Since the implementation part is written in Java, it is efficient to use some of the built in media frameworks. The shot detection prototype is designed on a foundation of two different Java frameworks, namely the **Java Media Framework (JMF)** [21] and **Java Advanced Imaging (JAI)** [22].

### 5.1.1 Java Media Framework

The Java Media Framework enables audio, video and other time-based media to be added to applications and applets built on Java technology. The package can capture, playback, stream and transcode multiple media formats. The most important part used in this thesis is the *mediaPlayer*. It takes a video file as input, and can perform different operations on this file. For simplicity only raw and uncompressed *.avi* files have been used, but the player may be configured to handle all kinds of known video formats and codecs. The media player contains information about the video file duration and the frames per second rate. Under the media control class, two important interfaces are used. The first one is the *FramePositioningControl*, that has the *seek* operator included. This operator takes one frame number as input, and positions the media player to the specific frame in the video file. The other interface is the *FrameGrabbingControl*, which contains the method *grabFrame()*. This method grabs a specific frame and returns it as a buffer. This buffer is written to a regular image with the *BufferToImage* class, and handed over to the JAI related classes.

### 5.1.2 Java Advanced Imaging

When the images are returned from the JMF classes, they have to be analysed so that the cuts in the video files can be detected. The JAI framework has some built in functions to ease this process. To create a color histogram for an entire given image, det following code is used (explanatory comments are written in blue):

**Algorithm 1** Color histogram creation

$int[]bins = \{64, 64, 64\}$;
The color value interval. 64 gives 4 intervals between 0 and 256
$double[]low = \{0, 0, 0\}$;
The lowest possible color value
$double[]high = \{256, 256, 256\}$;
The highest possible color value
$parameterBlock = newParameterBlock()$; The component to store the histogram input values
$parameterBlock.addSource(InputImage)$;
Adds the input image to the ParameterBlock
$parameterBlock.add(null)$;
No region of interest means the entire image
$parameterBlock.add(1)$;
Every pixel in the horizontal direction is checked
$parameterBlock.add(1)$;
Every pixel in the vertical direction is checked
$parameterBlock.add(bins)$;
Adds the color value interval
$parameterBlock.add(low)$;
Adds the low color value
$parameterBlock.add(high)$;
Adds the high color value
$return(Histogram)JAI.create("histogram", parameterBlock, null)$;
Returns the created color histogram

For each frame picked out from the video, one color histogram is created. To compare two consecutive histograms, the number of pixels for each value within the 64-valued band is counted for the R, B and G bins. The sum is compared with a fixed threshold value, and a cut is declared if this threshold is exceeded.

To create color histograms for specific locations within an image, the region of interest has to be declared. This is done by using the geometric class *Rectangle*. 9 distinct regions are specified, and one color histogram is created for each of the regions the same way as described in algorithm 1. The 9

regions are made with the following code (explanatory comments are written in blue):

**Algorithm 2** Region of interest specification

$rectangle = newRectangle(startingPointX, StartingPointY, regionWidth, regionHeight)$;
The rectangle to define the region of interest
$roi = newROIShape(rectangle)$;
Adds the rectangle to the region of interest
$parameterBlock.add(roi)$;
Adds the region of interest to the ParameterBlock

The edge chasing algorithm is divided into two parts. The first part defines the edges of an image, and the second part creates region histograms the same way as described in algorithm 2. The edges are defined with the following code (explanatory comments are written in blue):

**Algorithm 3** Edge detection

$float[]kernelMatrix = \{-1, -2, -1,$
$0, 0, 0,$
$1, 2, 1\}$;
Creates a constant array with the Sobel horizontal kernel. The sobel operator enhances the horizontal edges of the image by performing a convolution
$kernel = newKernelJAI(3, 3, kernelMatrix)$;
Creates the kernel using the kernel matrix
$outputImage = JAI.create("convolve", inputImage, kernel)$;
Runs the convolve operator, creating the processed output image
$return(outputImage)$;
returns the processed image

# Chapter 6

# Results

This chapter describes the results of the testing performed with the different algorithm approaches and the given video test material. Output result quality, computational costs and consumption of time are all taken into consideration.

## 6.1 Algorithm speed-up

After experimenting with the stepwise searching approach described in section 4.4, in combination with the Java Media Framework described in subsection 5.1.1, it was quite clear that the *FramePositioningControl* operator is too slow to be changed dynamically during the segmentation process. Too much time is wasted to go back and make the sequential search in the end. The *MediaPlayer* has to be stopped and started all over again, and the overall profits of these extra operations are in the end minimal.

The solution was to use the other approach instead, with fixed frames per second evaluations. The testing has been done with 1, 2 and 4 frames per second. The segmentation process is performed much more smoothly with this solution, and much less time is spent. But still, picking 2 or 4 frames per seconds is much less effective than just picking 1. Test results show that doubling the frame picking rate approximately doubles the time consumed by the segmentation process. And even more important: The outputs results are not much better than the results from the 1 frame per second approach. Because of this, the test results presented in this chapter are all output from the approach with a frame picking rate of 1 frame per second. The variables during the testing process are then limited to three factors:

- *Algorithm:* The three algorithms *Global color distribution*, *Local color distribution* and *Global edge chasing* have been tested.
- *Threshold value:* The threshold value for defining a shot in the video

files has been altered between *high*, *medium* and *low* during the different test cases.

- *Input video:* Video files with different properties have been used to test the robustness of the different algorithms.

All the tests have been run on a system with an AMD Athlon XP 2500+ CPU with 1.83 GHz and 512 MB of RAM. Since the different algorithms have been run on the same system, the relative time difference between them is unaffected by the hardware properties.

## 6.2 Video test material - Cuts and fades

All the video files have manually been examined to find the exact number of cuts, fade-ins and fade-outs. Table 6.1 shows the results from this examination.

| *Video file* | *Shots* | *Fade-ins* | *Fade-outs* |
|---|---|---|---|
| Britney Spears - You drive me crazy | 119 | 1 | 1 |
| Eric Clapton - Tears in heaven | 43 | 42 | 42 |
| Metallica - The unforgiven | 114 | 12 | 21 |
| Metallica - Turn the page | 163 | 2 | 8 |
| Newcastle United - Manchester United | 3 | 0 | 0 |
| Manchester United - Tottenham Hotspur | 12 | 2 | 2 |
| Top Gear - Koenigsegg test drive | 103 | 11 | 15 |
| Saturday Night Live - Blue Oyster Cult imitation | 52 | 4 | 3 |
| Throlltech - Qt4 dance | 39 | 4 | 4 |

Table 6.1: Cuts and fades in the test video files

## 6.3 Results from the global color histogram algorithm

All results described in this section are output from the *Global color histogram* algorithm, with the threshold value in *medium* mode. The number of shots detected with the threshold value in *low* and *high* mode are mentioned for comparison. The number of cuts are always 1 less than the number of shots in the video files.

**Britney Spears - You drive me crazy**
This video contains 119 shots, included 1 fade-in and 1 fade-out. With the threshold value in *low* mode, the algorithm detected 126 shots in the video. With the threshold value in *high* mode, 106 shots were detected. The overall

time consumption was 24.1 seconds.

With the threshold value in *medium* mode, 119 shots were detected. 111 of the shots were detected correctly. 8 shots were detected falsely, while 8 shots were not detected. The number of returned shots that are relevant (precision value) is $\frac{111}{119} = 93\%$, while the number of relevant shots that are returned (recall value) is $\frac{111}{119} = 93\%$.

The video is not the most challenging one according to color histograms, as it contains bright and clear colors, with few fade-ins and fade-outs. The false detections are mostly caused by light flashes and camera motion.

**Eric Clapton - Tears in heaven**
This video contains 43 shots, included 42 fade-ins and fade-outs. With the threshold value in *low* mode, the algorithm detected 58 shots in the video. With the threshold value in *high* mode, 37 shots were detected. The overall time consumption was 35.7 seconds.

With the threshold value in *medium* mode, 43 shots were detected. 33 of the shots were detected correctly. 10 shots were detected falsely, while 10 shots were not detected. The precision value is $\frac{33}{43} = 77\%$, while the recall value is $\frac{33}{43} = 77\%$.

This video is much more challenging than the Britney Spears video, as it contains a large amount of fade-ins and fade-outs. The colors are dark and diffuse as well. False detections are a major portion of the final output, and the fade-ins and fade-outs are probably the main reason for this.

**Metallica - The unforgiven**
This video contains 114 shots, included 12 fade-ins and 21 fade-outs. With the threshold value in *low* mode, the algorithm detected 131 shots in the video. With the threshold value in *high* mode, 96 shots were detected. The overall time consumption was 27.9 seconds.

With the threshold value in *medium* mode, 113 shots were detected. 99 of the shots were detected correctly. 14 shots were detected falsely, while 15 shots were not detected. The precision value is $\frac{99}{113} = 88\%$, while the recall value is $\frac{99}{114} = 87\%$.

This video is also very challenging according to color histograms, because of dark colors and almost black and white only. Most of the areas within the frames are black, meaning the histogram values on each side of a cut is quite similar.

**Metallica - Turn the page**
This video contains 163 shots, included 2 fade-ins and 8 fade-outs. With the threshold value in *low* mode, the algorithm detected 166 shots in the video. With the threshold value in *high* mode, 146 shots were detected. The overall time consumption was 28.6 seconds.

With the threshold value in *medium* mode, 160 shots were detected. 143 shots were detected correctly. 17 shots were detected falsely, while 20 shots were not detected. The precision value is $\frac{143}{160} = 89\%$, while the recall value is $\frac{143}{163} = 88\%$.

This video seemed less challenging than the Unforgiven video because of the bright colors. But the test results show that the outcome was nearly the same, probably because of frequently camera changing.

**Football goal: Newcastle United - Manchester United**
This video contains 3 shots, included 0 fade-ins and 0 fade-outs. With the threshold value in *low* mode, the algorithm detected 4 shots in the video. With the threshold value in *high* mode, 2 shots were detected. The overall time consumption was 2.7 seconds.

With the threshold value in *medium* mode, 3 shots were detected. 3 of the shots were detected correctly. 0 shots was detected falsely, while 0 shots was not detected. The precision value is $\frac{3}{3} = 100\%$, while the recall value is $\frac{3}{3} = 100\%$.

This video is not very challenging. The cut points are clear and easy to spot. It is not surprising that the algorithm detects all cuts without any errors. The video is also very short in time, so the efficient time consumption of 2.7 seconds is as expected.

**Football goal: Manchester United - Tottenham Hotspur**
This video contains 12 shots, included 2 fade-ins and 2 fade-outs. With the threshold value in *low* mode, the algorithm detected 16 shots in this video. With the threshold value in *high* mode, 10 shots were detected. The overall time consumption was 6.8 seconds.

With the threshold value in *medium* mode, 12 shots were detected. 10 of the shots were detected correctly. 2 shots were detected falsely, while 2 shots were not detected. The precision value is $\frac{10}{12} = 83\%$, while the recall value is $\frac{10}{12} = 83\%$.

This video is not very challenging. Both false detections are caused by people suddenly coming in the way of the camera, and the algorithm interprets

this as cuts in the video.

**Top Gear - Koenigsegg test drive**
This video contains 103 shots, included 11 fade-ins and 15 fade-outs. With the threshold value in *low* mode, the algorithm detected 115 shots in this video. With the threshold value in *high* mode, 94 shots were detected. The overall time consumption was 42.2 seconds.

With the threshold value in *medium* mode, 102 shots were detected. 96 of the shots were detected correctly. 6 shots were detected falsely, while 7 shots were not detected. The precision value is $\frac{96}{102} = 94\%$, while the recall value is $\frac{96}{103} = 93\%$.

Some parts of this video are challenging, as a lot of zooming effects are used. These parts are the main reason for the false detections.

**Saturday Night Live - Blue Oyster Cult imitation**
This video contains 52 shots, included 4 fade-ins and 3 fade-outs. With the threshold value in *low* mode, the algorithm detected 56 shots in this video. With the threshold value in *high* mode, 44 shots were detected. The overall time consumption was 8.9 seconds.

With the threshold value in *medium* mode, 48 shots were detected. 46 of the shots were detected correctly. 2 shots were detected falsely, while 6 shots were not detected. The precision value is $\frac{46}{48} = 96\%$, while the recall value is $\frac{46}{52} = 88\%$.

All cuts in this video are relatively easy to spot. The main reason for the low time consumption is the video's frame rate, that is considerably lower than the frame rate of the other test videos.

**Throlltech - Qt4 dance**
This video contains 39 shots, included 4 fade-ins and 4 fade-outs. With the threshold value in *low* mode, the algorithm detected 50 shots in this video. With the threshold value in *high* mode, 34 shots were detected. The overall time consumption was 38.1 seconds.

With the threshold value in *medium* mode, 39 shots were detected. 32 of the shots were detected correctly. 7 shots were detected falsely, while 7 shots were not detected. The precision value is $\frac{32}{39} = 82\%$, while the recall value is $\frac{32}{39} = 82\%$

This video is partly challenging, as some areas contain a lot of zooming affects and people in motion. These areas are the main reason for the false

detections.

## 6.4 Results from the local color histogram algorithm

All results described in this section are output from the *Local color histogram* algorithm, with the threshold value in *medium* mode. The number of shots detected with the threshold value in *low* and *high* mode are mentioned for comparison.

**Britney Spears - You drive me crazy**
This video contains 119 shots, included 1 fade-in and 1 fade-out. With the threshold value in *low* mode, the algorithm detected 131 shots in the video. With the threshold value in *high* mode, 100 shots were detected. The overall time consumption was 18.9 seconds.

With the threshold value in *medium* mode, 115 shots were detected. 107 of the shots were detected correctly. 8 shots were detected falsely, while 12 shots were not detected. The precision value is $\frac{107}{115} = 93\%$, while the recall value is $\frac{107}{119} = 90\%$.

**Eric Clapton - Tears in heaven**
This video contains 43 shots, included 42 fade-ins and fade-outs. With the threshold value in *low* mode, the algorithm detected 60 shots in the video. With the threshold value in *high* mode, 31 shots were detected. The overall time consumption was 23.0 seconds.

With the threshold value in *medium* mode, 43 shots were detected. 36 of the shots were detected correctly. 7 shots were detected falsely, while 7 shots were not detected. The precision value is $\frac{36}{43} = 84\%$, while the recall value is $\frac{36}{43} = 84\%$.

**Metallica - The unforgiven**
This video contains 114 shots, included 12 fade-ins and 21 fade-outs. With the threshold value in *low* mode, the algorithm detected 125 shots in the video. With the threshold value in *high* mode, 97 shots were detected. The overall time consumption was 28.5 seconds.

With the threshold value in *medium* mode, 115 shots were detected. 102 of the shots were detected correctly. 13 shots were detected falsely, while 12 shots were not detected. The precision value is $\frac{102}{115} = 89\%$, while the recall value is $\frac{102}{114} = 89\%$.

**Metallica - Turn the page**
This video contains 163 shots, included 2 fade-ins and 8 fade-outs. With the threshold value in *low* mode, the algorithm detected 169 shots in the video. With the threshold value in *high* mode, 151 shots were detected. The overall time consumption was 24.5 seconds.

With the threshold value in *medium* mode, 160 shots were detected. 150 shots were detected correctly. 10 shots were detected falsely, while 13 shots were not detected. The precision value is $\frac{150}{160} = 94\%$, while the recall value is $\frac{150}{163} = 92\%$.

**Football goal: Newcastle United - Manchester United**
This video contains 3 shots, included 0 fade-ins and 0 fade-outs. With the threshold value in *low* mode, the algorithm detected 5 shots in the video. With the threshold value in *high* mode, 2 shots were detected. The overall time consumption was 2.4 seconds.

With the threshold value in *medium* mode, 3 shots were detected. 3 of the shots were detected correctly. 0 shot was detected falsely, while shots 0 was not detected. The precision value is $\frac{3}{3} = 100\%$, while the recall value is $\frac{3}{3} = 100\%$.

**Football goal: Manchester United - Tottenham Hotspur**
This video contains 12 shots, included 2 fade-ins and 2 fade-outs. With the threshold value in *low* mode, the algorithm detected 17 shots in this video. With the threshold value in *high* mode, 10 shots were detected. The overall time consumption was 6.2 seconds.

With the threshold value in *medium* mode, 13 shots were detected. 10 of the shots were detected correctly. 3 shots were detected falsely, while 2 shots were not detected. Both the fade-ins and fade-outs were found. The precision value is $\frac{10}{13} = 77\%$, while the recall value is $\frac{10}{12} = 83\%$.

**Top Gear - Koenigsegg test drive**
This video contains 103 shots, included 11 fade-ins and 15 fade-outs. With the threshold value in *low* mode, the algorithm detected 113 shots in this video. With the threshold value in *high* mode, 88 shots were detected. The overall time consumption was 33.9 seconds.

With the threshold value in *medium* mode, 105 shots were detected. 91 of the shots were detected correctly. 14 shots were detected falsely, while 12 shots were not detected. The precision value is $\frac{91}{105} = 87\%$, while the recall value is $\frac{91}{103} = 88\%$.

**Saturday Night Live - Blue Oyster Cult imitation**
This video contains 52 shots, included 4 fade-ins and 3 fade-outs. With the threshold value in *low* mode, the algorithm detected 57 shots in this video. With the threshold value in *high* mode, 48 shots were detected. The overall time consumption was 15.1 seconds.

With the threshold value in *medium* mode, 52 shots were detected. 48 of the shots were detected correctly. 4 shots were detected falsely, while 4 shots were not detected. The precision value is $\frac{48}{52} = 92\%$, while the recall value is $\frac{48}{52} = 92\%$.

**Throlltech - Qt4 dance**
This video contains 39 shots, included 4 fade-ins and 4 fade-outs. With the threshold value in *low* mode, the algorithm detected 46 shots in this video. With the threshold value in *high* mode, 30 shots were detected. The overall time consumption was 28.9 seconds.

With the threshold value in *medium* mode, 38 shots were detected. 37 of the shots were detected correctly. 1 shot was detected falsely, while 2 shots were not detected. The precision value is $\frac{37}{38} = 97\%$, while the recall value is $\frac{37}{39} = 95\%$.

## 6.5 Results from the global edge chasing algorithm

All results described in this section are output from the *Global edge chasing* algorithm, with the threshold value in *medium* mode. The number of shots detected with the threshold value in *low* and *high* mode are mentioned for comparison.

**Britney Spears - You drive me crazy**
This video contains 119 shots, included 1 fade-in and 1 fade-out. With the threshold value in *low* mode, the algorithm detected 129 shots in the video. With the threshold value in *high* mode, 114 shots were detected. The overall time consumption was 26.1 seconds.

With the threshold value in *medium* mode, 121 shots were detected. 99 of the shots were detected correctly. 22 shots were detected falsely, while 20 shots were not detected. The precision value is $\frac{99}{121} = 82\%$, while the recall value is $\frac{99}{119} = 83\%$.

**Eric Clapton - Tears in heaven**
This video contains 43 shots, included 42 fade-ins and fade-outs. With the threshold value in *low* mode, the algorithm detected 57 shots in the video.

With the threshold value in *high* mode, 34 shots were detected. The overall time consumption was 38.8 seconds.

With the threshold value in *medium* mode, 45 shots were detected. 36 of the shots were detected correctly. 9 shots were detected falsely, while 7 shots were not detected. The precision value is $\frac{36}{45} = 80\%$, while the recall value is $\frac{36}{43} = 84\%$.

**Metallica - The unforgiven**
This video contains 114 shots, included 12 fade-ins and 21 fade-outs. With the threshold value in *low* mode, the algorithm detected 105 shots in the video. With the threshold value in *high* mode, 94 shots were detected. The overall time consumption was 35.9 seconds.

With the threshold value in *medium* mode, 105 shots were detected. 86 of the shots were detected correctly. 19 shots were detected falsely, while 28 shots were not detected. The precision value is $\frac{86}{105} = 82\%$, while the recall value is $\frac{86}{114} = 75\%$.

**Metallica - Turn the page**
This video contains 163 shots, included 2 fade-ins and 8 fade-outs. With the threshold value in *low* mode, the algorithm detected 178 shots in the video. With the threshold value in *high* mode, 154 shots were detected. The overall time consumption was 35.9 seconds.

With the threshold value in *medium* mode, 165 shots were detected. 141 shots were detected correctly. 24 shots were detected falsely, while 22 shots were not detected. The precision value is $\frac{141}{165} = 85\%$, while the recall value is $\frac{141}{163} = 87\%$.

**Football goal: Newcastle United - Manchester United**
This video contains 3 shots, included 0 fade-ins and 0 fade-outs. With the threshold value in *low* mode, the algorithm detected 6 shots in the video. With the threshold value in *high* mode, 3 shots were detected. The overall time consumption was 3.5 seconds.

With the threshold value in *medium* mode, 3 shots were detected. 2 of the shots were detected correctly. 1 shot was detected falsely, while 1 shot was not detected. The precision value is $\frac{2}{3} = 67\%$, while the recall value is $\frac{2}{2} = 67\%$.

**Football goal: Manchester United - Tottenham Hotspur**
This video contains 12 shots, included 2 fade-ins and 2 fade-outs. With the threshold value in *low* mode, the algorithm detected 12 shots in this video.

With the threshold value in *high* mode, 9 shots were detected. The overall time consumption was 13.8 seconds.

With the threshold value in *medium* mode, 11 shots were detected. 5 of the shots were detected correctly. 6 shots were detected falsely, while 7 shots were not detected. The precision value is $\frac{5}{11} = 45\%$, while the recall value is $\frac{5}{12} = 42\%$.

**Top Gear - Koenigsegg test drive**
This video contains 103 shots, included 11 fade-ins and 15 fade-outs. With the threshold value in *low* mode, the algorithm detected 118 shots in this video. With the threshold value in *high* mode, 95 shots were detected. The overall time consumption was 43.5 seconds.

With the threshold value in *medium* mode, 102 shots were detected. 80 of the shots were detected correctly. 22 shots were detected falsely, while 23 shots were not detected. The precision value is $\frac{80}{102} = 78\%$, while the recall value is $\frac{80}{103} = 78\%$.

**Saturday Night Live - Blue Oyster Cult imitation**
This video contains 52 shots, included 4 fade-ins and 3 fade-outs. With the threshold value in *low* mode, the algorithm detected 65 shots in this video. With the threshold value in *high* mode, 45 shots were detected. The overall time consumption was 17.2 seconds.

With the threshold value in *medium* mode, 54 shots were detected. 44 of the shots were detected correctly. 10 shots were detected falsely, while 8 shots were not detected. The precision value is $\frac{44}{54} = 81\%$, while the recall value is $\frac{44}{52} = 85\%$.

**Throlltech - Qt4 dance**
This video contains 39 shots, included 4 fade-ins and 4 fade-outs. With the threshold value in *low* mode, the algorithm detected 50 shots in this video. With the threshold value in *high* mode, 33 shots were detected. The overall time consumption was 50.7 seconds.

With the threshold value in *medium* mode, 42 shots were detected. 33 of the shots were detected correctly. 9 shot were detected falsely, while 6 shots were not detected. The precision value is $\frac{33}{42} = 79\%$, while the recall value is $\frac{33}{39} = 85\%$.

## 6.6 Algorithms head-to-head

To see the performance of the different algorithms head-to-head, all the results have been written in tables to give an overview of the time consumptions, the precision values and the recall values. The best result for each video file is written in blue. Table 6.2 shows the time consumptions for all of the algorithms on the different test videos. The best results were shared between the global color histogram algorithm and the local color histogram algorithm, with a distinct predominance in the favour of the local approach. The time consumptions are also illustrated by the graph in figure 6.1. As this figure shows, the local color histogram algorithm has the lowest time consumptions in most cases. The edge chasing approach is a little bit higher in all the test cases.

| *Video file* | *Global* | *Local* | *Edge chasing* |
|---|---|---|---|
| You drive me crazy | 24.1 s | 18.9 s | 26.1 s |
| Tears in heaven | 35.7 s | 23.0 s | 38.8 s |
| The unforgiven | 27.9 s | 28.5 s | 35.9 s |
| Turn the page | 28.6 s | 24.5 s | 35.9 s |
| Newcastle United - Man U | 2.7 s | 2.4 s | 3.5 s |
| Man U - Tottenham Hotspur | 6.8 s | 6.2 s | 13.8 |
| Koenigsegg test drive | 42.2 s | 33.9 s | 43.5 s |
| Blue Oyster Cult imitation | 8.9 s | 15.1 s | 17.2 s |
| Qt4 dance | 38.1 s | 28.9 s | 50.7 s |

Table 6.2: Algorithm time consumptions

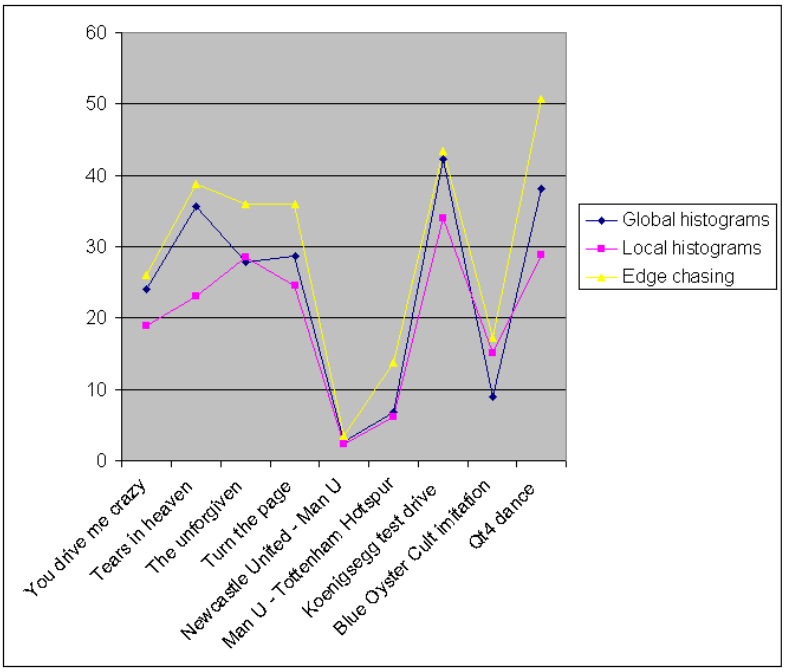


Figure 6.1: Algorithm time consumptions

Table 6.3 shows the algorithm precision values head-to-head. The best results are shared almost in half between the global and local color histogram algorithms. This is also illustrated in figure 6.2. The results are quite even, but the edge chasing algorithm has a drop-point in the *Tottenham* video test case, with a precision value only on 45%.

| *Video file* | *Global* | *Local* | *Edge chasing* |
|---|---|---|---|
| You drive me crazy | 93% | 93% | 82% |
| Tears in heaven | 77% | 84% | 80% |
| The unforgiven | 88% | 89% | 82% |
| Turn the page | 89% | 94% | 85% |
| Newcastle United - Man U | 100% | 100% | 67% |
| Man U - Tottenham Hotspur | 83% | 77% | 45% |
| Koenigsegg test drive | 94% | 87% | 78% |
| Blue Oyster Cult imitation | 96% | 92% | 81% |
| Qt4 dance | 82% | 97% | 79% |

Table 6.3: Algorithm precision values

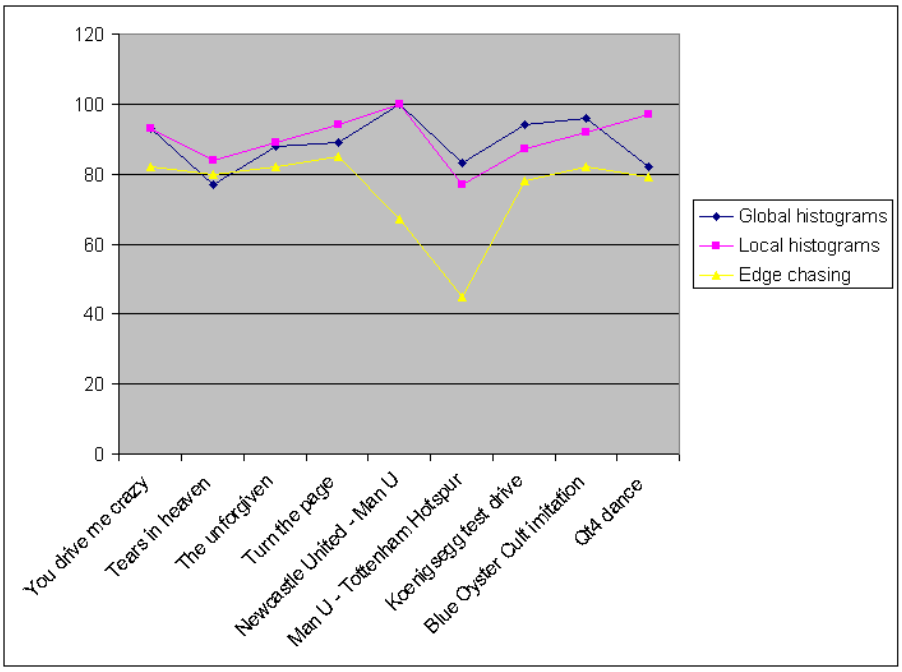


Figure 6.2: Algorithm precision values

Table 6.4 shows the algorithm recall values head-to-head. The local color histogram algorithm has most of the best results. This is also illustrated in figure 6.3. Like with the precision value, the edge chasing algorithm has a drop-point related to the *Tottenham* video. The recall value is as low as 42%.

| *Video file* | *Global* | *Local* | *Edge chasing* |
|---|---|---|---|
| You drive me crazy | 93% | 90% | 83% |
| Tears in heaven | 77% | 84% | 84% |
| The unforgiven | 87% | 89% | 75% |
| Turn the page | 88% | 92% | 87% |
| Newcastle United - Man U | 100% | 100% | 67% |
| Man U - Tottenham Hotspur | 83% | 83% | 42% |
| Koenigsegg test drive | 93% | 88% | 78% |
| Blue Oyster Cult imitation | 88% | 92% | 85% |
| Qt4 dance | 82% | 95% | 85% |

Table 6.4: Algorithm recall values

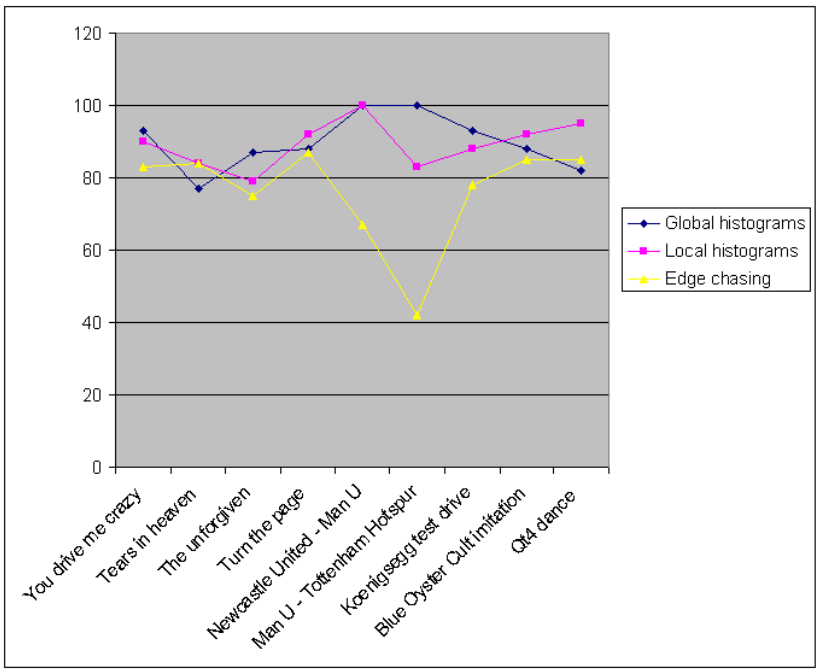


Figure 6.3: Algorithm recall values

# Chapter 7

# Conclusion and further work

This chapter evaluates the results from the different algorithm approaches tested in chapter 6. The main advantages and disadvantages are stated out for each of the algorithms, and a final comparison between all approaches is done in the end.

## 7.1 Global color histograms

This algorithm approach is very good for finding cuts that do not involve fading or zooming effects. The detection rate is high for this type, and few cuts are missed. The precision value is in average 89.1% for the videos in the test set, while the recall value is in average 89.8%. The r-frames picked are also very clear and descriptive. The time consumption for this approach is also effective, as it in average uses 10.4% of the actual video file duration.

The algorithm is much more inaccurate if the cut involves a fade-in and/or a fade-out. The cut may be missed completely, or the picked r-frame may not be good enough to describe the video content at the given point. Another weakness is that objects suddenly moving in front of the camera is detected as a cut. One example of this is the *Tottenham* video, where a person moving in front of the camera makes the algorithm pick out that point as a cut in the video, which is a false decision. Dark colors with few changes between shots is also a challenge related to this approach. In particular, several false detections in the *Metallica - The Unforgiven* video are caused by this.

## 7.2 Local color histograms

As the global color histogram algorithm, the local color histogram algorithm is very good for finding cuts that do not involve fading or zooming effects. This algorithm approach is overall a little bit better than the global histograms solution, and the output results are more precise. The precision

value is in average 90.3% for the videos in the test set, while the recall value is in average 90.3%. In most cases the precision and recall values are slightly better for this approach, as tables 6.3 and 6.4 show. The time consumption is also more efficient, as this solution in average uses 8.8% of the actual video file duration.

The same weaknesses exist for this approach as the global histograms solution, but the output is a little better for this solution regarding fade-ins and fade-outs. The reason for this is probably the region dividing, but the results are not good enough to say that this approach solves the fade-in/fade-out problem.

## 7.3 Global edge chasing

The most interesting results are those from the global edge chasing approach, as this is a new way of combining two approaches into a new one to detect video cuts. The test results show that the output quality is dependent on which video files the algorithm is tested with. If the video file has few colors and indistinct edges, it is hard for this algorithm to detect the right cuts. An example of this is the dark *Metallica - The Unforgiven* video, where the indistinct edges lead to several false detections. In the *Metallica - Turn the page* video, the colors are brighter and the edges are more clear. The recall value for this video is much higher than the for the previous one, as table 6.4 shows. It is quite clear that edges that are easy to spot in the test videos lead to better output results. The precision value is in average 75.6% for the videos in the test set, while the recall value is in average 76.2%. An interesting point about this algorithm is that it seems fade-ins and fade-outs are easier spotted than with the regular color histogram algorithms. This may be caused by the fact that the edges in the pictures change fast even if one image gradually floats into another.

The main drawback of this algorithm is that it is costly to both compute the edge images and the histograms at the same time. This is shown by the average time consumption of 14.0% of the actual video file duration. This value is higher for short video files since the algorithm is slow in the first phase of the shot detection process. Further, some false cut detections may occur because of a large number of edges in a limited area within the picture. This is illustrated by figure 7.1. On the figure, the two images labelled #1 and #2 is two consecutive frames within the same shot in the video. The edge detector creates one edge for each mesh in the net, as shown in the images labelled #3 and #4. Since the threshold for the histogram comparator is low, the small changes from image #3 and image #4 are enough to define a cut, which is clearly a false decision.

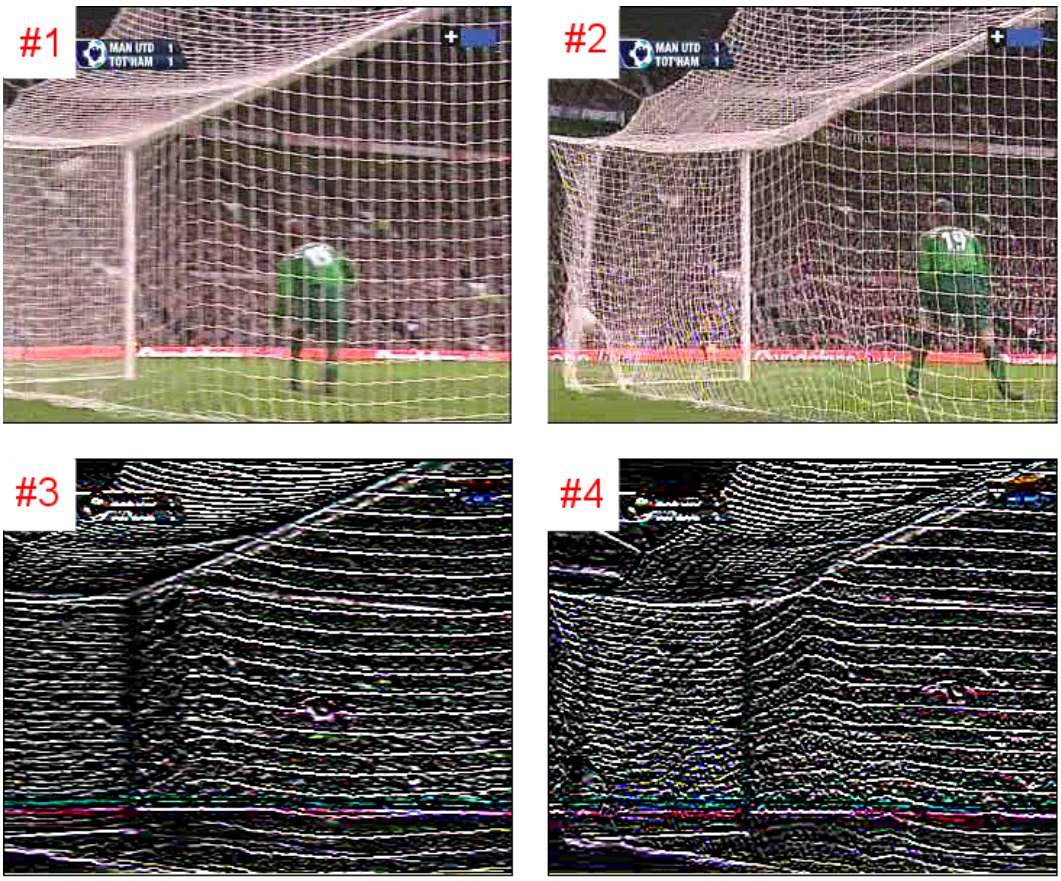


Figure 7.1: False cut detection by the edge chasing algorithm

## 7.4 Conclusion

The results show that the best solution among the approaches tested in this thesis is the *local color histograms* algorithm. This approach has in average the most precise output, and is generally the most efficient related to time consumption. The *global color histograms* algorithm is not far behind, and some of the results are almost identical. The weakness for both of the solutions is the problem related to correct detections of fade-ins and fade-outs. Some of the fade-ins and fade-outs in the test videos were not found by either of the algorithms.

The *global edge chasing* algorithm detected fade-ins and fade-outs in a better way than the color histograms solutions, but the overall results were not good enough. The algorithm is slow compared to the two others, and the final precision and recall values are too dependent on the test video characteristics. Dark videos with indistinct edges revealed some major weaknesses related to this approach.

My personal opinion is that both the color histogram algorithms produce good results, both related to precision/recall values and overall time consumption. As a matter of fact I am a bit surprised that the local algorithm was faster than the global solution. I expected the dividing into image regions to result in larger computational costs, hence larger time consumption. The opposite thing happened, and the local histograms solution outperformed the global histograms solution in most cases.

Regarding the global edge chasing approach, I found it disappointing that the algorithm failed in some test cases. Precision and recall values below 50% is not acceptable in a shot detection application. A shot detection algorithm should deliver satisfactorily output independent of the video file characteristics. The extra time consumption related to this algorithm was expected because of the extra computational operations, and I think the results were as expected compared to the regular color histogram algorithms.

## 7.5 Further work

There are some factors related to the algorithms designed in this thesis that can be improved, to get more precise output and reduce the overall time consumption.

### 7.5.1 Increased sampling rate

During testing it was shown that if the algorithms picked more than 1 frame per second, the final precision and recall values were improved. This im-

provement was not big enough to overcome the fact that doubling the frame picking rate also doubled the overall time consumption, but in combination with another improvement this is possible to utilize. The main idea is to divide the input video file into smaller parts, and perform shot detection in parallel with increased frame picking rate. An example of this is to divide the input video into four distinct parts, and create one frame picking operator to each part. The operators pick frames in parallel, each operator from the appurtenant part of the video. If the frame picking rate simultaneously is increased from 1 to 4, the overall time consumption should stay unchanged. Most likely this will reduce the number of false detections, meaning the precision and recall values to improve.

### 7.5.2 Increased user control

Another possible improvement is to give the user of the shot detection application the possibility to make some adjustments before the shot detection process is started. Based on these adjustments, the algorithm parameters can be regulated to make the output results as good as possible. These factors may involve color features of the video file and the video file's category. If the video contains few and dark colors, the algorithm should be adjusted to be more focused on light changes in the video. Further, if the video category is *Music video*, the algorithm's motion sensitivity should be lowered to reduce the number of false detections because of this.

### 7.5.3 Permanent picture storyline storage

As the implemented shot detection application writes the r-frames directly to the GUI, this information is lost when the program is closed. In a adequate system, as an example related to a TV station, it is desirable to detect the shots in the video archive one time only, and then permanently store the picture storyline. This can easily be done by writing the r-frames to disk when they are detected, or flushing the information to a database after the shot detection process is finished.

### 7.5.4 Image searching

If the shot detection application is used with the intention to describe video content by images, it is probably desirable to have image searching as a built-in feature. An example image may be used for searching, and the data store is then scanned for matching or nearly matching results. Different image features may be used as foundation for the search function, for example image colors, texture and shapes.

# Bibliography


[1] Paul Browne, Alan F Smeaton, Noel Murphy, Noel O'Connor, Seán Marlov, and Catherine Berrut. Evaluating and combining digital video shot boundary detection algorithms. 2000.

[2] Boon-Lock Yeo and Minerva M. Yeung. Retrieving and visualizing video. 1997.

[3] Kjell Bratbergsengen. Lagring og behandling av store datamengder. 2003.

[4] Wikipedia.org. *Wikipedia*.

[5] Markos Mentzelopoulus and Alexandra Psarrou. Key-frame extraction algorithm using entropy difference. 2004.

[6] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. 1996.

[7] F. Arman, R. Depommier, A. Hsu, and M-Y. Chiu. Content-based browsing of video sequences. 1994.

[8] Marinette Bouet, Ali Khenchaf, and Henri Briand. Shape representation for image retrieval. 1999.

[9] Davood Rafiei and Alberto O. Mendelzon. Efficient retrieval of similar shapes. 2002.

[10] Dengsheng Zhang, Aylwin Wong, Maria Indrawan, and Guojun Lu. Content-based image retrieval using gabor texture features. 2000.

[11] Bikash Sabata and Moises Goldszmidt. Fusion of multiple cues for video segmentation. 1999.

[12] Xia Yong, Dagan Feng, and Zhao Rongchun. Optimal selection of image segmentation algorithms based on performance predication. 2004.

[13] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[14] John S. Boreczky and Lawrence A. Rowe. Comparision of video shot boundary detection techniques. 1996.

[15] S. M. M. Tahaghoghi, Hugh E. Williams, James A. Thom, and Timo Volkmer. Video cut detection using frame windows. 2005.

[16] Huamin Feng, Wei Fang, Sen Liu, and Yong Fang. A new general framework for shot boundary detection and key-frame extraction. 2005.

[17] Ramin Zabiha, Justin Miller, and Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. 1995.

[18] Hisashi Aoki, Shigeyoshi Shimotsuji, and Osamu Hori. A shot classification method of selecting effective key-frames for video browsing. 1996.

[19] Nevenka Dimitrova, Thomas McGee, and Herman Elenbaas. Video keyframe extraction and filtering: A keyframe is not a keyframe to everyone. 1997.

[20] Kien A. Hua and JungHwan Oh. Detecting video shot boundaries up to 16 times faster. 2000.

[21] Sun Microsystems. *Java Media Framework 1.0 Programmers Guide*, 1998.

[22] Sun Microsystems. *Programming in Java Advanced Imaging*, 1999.