A video summarisation system for post-production

Ciaran John Wills

A Dissertation submitted to the University of Glasgow in partial fulfillment of the regulations for the degree of Doctor of Philosophy

September 2002

Department of Computing Science
University of Glasgow
Glasgow, UK

Abstract

Post-production facilities deal with large amounts of digital video, which presents difficulties when tracking, managing and searching this material. Recent research work in image and video analysis promises to offer help in these tasks, but there is a gap between what these systems can provide and what users actually need. In particular the popular research models for indexing and retrieving visual data do not fit well with how users actually work. In this thesis we explore how image and video analysis can be applied to an online video collection to assist users in reviewing and searching for material faster, rather than purporting to do it for them.

We introduce a framework for automatically generating static 2-dimensional *storyboards* from video sequences. The storyboard consists of a series of frames, one for each shot in the sequence, showing the principal objects and motions of the shot. The storyboards are rendered as vector images in a familiar comic book style, allowing them to be quickly viewed and understood. The process consists of three distinct steps: shot-change detection, object segmentation, and presentation.

The nature of the video material encountered in a post-production facility is quite different from other material such as television programmes. Video sequences such as commercials and music videos are highly dynamic with very short shots, rapid transitions and ambiguous edits. Video is often heavily manipulated, causing difficulties for many video processing techniques.

We study the performance of a variety of published shot-change detection algorithms on the type of highly dynamic video typically encountered in post-production work. Finding their performance disappointing, we develop a novel algorithm for detecting cuts and fades that operates directly on Motion-JPEG compressed video, exploiting the DCT coefficients to save computation. The algorithm shows superior performance on highly dynamic material while performing comparably to previous algorithms on other material.

Within each shot we attempt to identify and segment the important objects. Our segmentation approach is based on the European COST 211 Group's Analysis Module, which we examine and attempt to improve. The Analysis Module integrates information from a number of different segmentation steps such as colour and motion segmentation. We explore how these individual steps can be adapted to improve performance. Despite producing a small increase in performance over the previous implementation the segmentations generated by the algorithm are still far from ideal.

The rendering process produces a single image for each shot in the sequence, using information gathered by the segmentation phase. Vector outlines are fit to the objects and they are further segmented into their major colour regions. The objects are rendered on top of a representation of the background of the scene which is processed in a similar manner. The image is annotated to indicate the motion of both the objects and the camera in the scene, using conventions from comic and storyboard art.

Contents

List of Figures

List of Tables

Acknowledgements

Thanks first of all to my supervisors, Paul Cockshott and Paul Siebert, for their guidance over the course of this work, and to John Patterson for many interesting discussions and for making it all possible in the first place.

Thanks are also due to my fellow research students in the Faraday lab and the Department of Computing Science, who have made the previous three years in Glasgow most enjoyable—thanks in particular to Alasdair Coull, Stuart Ford, Sylvain Brugnot, Donald MacVicar, Matthew Cairns and Josh Hale.

I have benefited enormously from the presence of the commercial partners in this work, Smoke & Mirrors and Unique-ID. In particular I am forever indebted to Chris Eborn for his assistance and advice on all areas of video post-production, not to mention accommodation on my numerous visits to London.

This work was supported by a EPSRC Case studentship and by sponsorship from Unique-ID and Smoke & Mirrors. Additional travel funding was provided by the Research Student Committee of the Department of Computing Science.

Ciaran J. Wills
London, 24 September 2002.

Declaration of Authorship

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Ciaran J. Wills
London, 25 September 2002.

1. Introduction

Digital imaging is now prevalent in the wider world beyond the Internet—much commercial visual media is produced and distributed digitally. A television program can be produced and distributed completely in digital form, all the way from the camera to the receiver in the home.

Digital video acquisition and editing has brought huge changes to the television industry; broadcast standard cameras and editing equipment are now a fraction of the price they were ten years ago, allowing new channels and production companies to start up with a minimal outlay. News operations have changed radically as material from the field can be transfered back to the newsroom immediately, edited on a desktop computer together with online archive material, and put on air within minutes.

The ever decreasing costs of digital storage and communications means that online digital video is going to play a large part in the future, but new tools and ways of working are going to be required to take full advantage of all this accessible material. Academic and commercial researchers have started to tackle these problems, but few solutions have come on to the commercial market so far. The most successful applications of the technology have been in specialised application areas, for example television newsrooms.

Film and video post-production is a growth industry—the power of digital image manipulation software and the continual efforts of producers to impact upon an increasingly sophisticated audience means that hardly any film is made without some form of manipulation in post-production. This includes not just special effects, but also 'invisible' alterations such as recolouring or the removal of unwanted elements.

The increasing availability of low-cost commercial editing and manipulation packages, such as Adobe's *Premiere* and *After Effects*, means that a lot of simple post-production tasks can be taken care of in-house. However there is a market for specialists to do more complex post-production work, especially for films and television commercials. The large budgets of films and commercials allow the producers to spend considerable amounts of money achieving exactly the image they want, and a multi-

million pound industry exists to provide these specialised services.[1]

This thesis examines how computer processing of video material can aid in the management of the large amount of video that passes through a post-production facility. The particular problem we focus on is the process of searching for material that already exists in the facility. This occurs in several aspects of post-production work as we detail in Chapter 2.

We propose that storyboards provide a useful abstraction of the content of video clips. They provide a static 2-dimensional encapsulation of all the information necessary for the user to quickly understand the contents of the clip and decide whether it is relevant. The storyboard format is familiar to many users through the use of storyboards as a pre-production planning tool, and familiar to people in general due to its similarity to a comic book format.

We will present a process for generating storyboards from unstructured video, using computer analysis of video to extract the important cues that convey the content of the material. A static 2-dimensional vector representation is produced that can be displayed or printed. The entire process is intended to be automatic, generating storyboards from any available online video.

The images produced by the system do a good job of conveying what is happening in each shot. However the motion analysis stage of the process is not robust enough for use in an automated system, requiring instead the manual markup of images in order to produce the final storyboard image. This has precluded the process being applied to more material. However as motion analysis techniques improve we anticipate being able to apply the process automatically to large amounts of video material.

In developing the storyboard process, we have developed a novel shot detection algorithm that gives improved performance on highly dynamic video, improved the performance of an existing object segmentation framework, and explored methods of presenting movement information in a static 2-dimensional context.

[1] Television commercials can have enormous budgets, even compared to films. *Pearl Harbour*, which was the most expensive film made to date at the time of its release, cost $140 million and has a running time of 183 minutes [5], while Levi's recently spent $2.5 million on a 60 second commercial [4]. Television programmes have much lower budgets relative to their length.

The structure of this thesis is as follows: Chaper 2 examines the background of a post-production facility, the type of work and material they deal with, and their use of digital video. Chapter 3 outlines the approach or our storyboarding technique. Chapter 4 investigates the performance of temporal segmentation algorithms on the type of video encountered in post-production, and describes our novel shot and fade detection algorithms. Chapter 5 describes how we identify and extract the moving objects from each shot and Chapter 6 shows how this information is presented in the storyboard frame. Chapter 7 then examines our results and outlines possible future work.

2. Post-production

This chapter examines the environment and workflow of a post-production facility, and the current level of computer-aided asset management (in the form of the *Cakes* system). We then discuss how such systems and current research in visual information management can be further exploited to aid the workflow of a post-production facility.

The details in this chapter are based on a single post-production facility, but are typical of most facilities.

2.1 Inside a post-production facility

Smoke & Mirrors [1] is an independent post-production facility in London's Soho. Positioning itself at the upper end of the market, it specialises in television commercials, music videos ('pop promos') and some film work.

The clients provide the raw footage and one of the facility's artists will manipulate and edit it to produce the desired final result. A variety of software tools are used, but Discreet's *inferno* and *flame* compositing systems [3] are the core of the process. The entire process is digital, and the finished material is returned to the clients in digital form. The audio track is produced elsewhere and dubbed on at the end of the process. The clients are closely involved in the post-production process, requiring regular updates of the work in progress and often sitting in on sessions in the editing suite.

With five editing suites and numerous clients, scheduling and managing resources is a complex task for the facility. Hardware, software and artist time are very expensive and must be kept as fully utilised as possible to earn a return on the investment. Each job also has large amounts of video material associated with it which must be tracked and managed. Upon completion of a job, all the source, intermediate and finished material is archived on tape and stored in the library.

2.2 Post-production workflow

An independent post-production facility makes its living by handling jobs for a variety of external clients. In a larger facility there may be many jobs in progress at any one time, which must be kept separate from each other.

At the lowest level a job consists of receiving raw footage from the client, manipulating it according to their wishes, and returning the finished material to them. However, in reality the process is usually much more complex than this.

Incoming material does not all arrive at the start of the job as post-production often begins before photography is complete. Any delays upstream will cascade down and since post-production is near the end of the pipeline it suffers compression from upstream delays and inflexible downstream deadlines. Airtime for commercials is booked far in advance, and music videos are required by the record's release date, which is also fixed in advance—a large amount of production, marketing and other work revolves around these deadlines and once they are fixed they are very difficult to change. A post-production facility's reputation rests to a large extent on its ability to turn work around quickly close to deadlines. A facility can at times be a 24-hour operation, and when a courier brings a tape to the door it may need to be available for an artist to work on as quickly as possible.

Clients also require regular feedback on the progress of their jobs, and must approve work at various stages. This gives them many opportunities to change various parts of the brief, sometimes radically, which the facility must accommodate.

2.2.1 Anatomy of a post-production facility

The central work of manipulating video is done by compositing artists—these are highly experienced and highly paid individuals who must be able to work quickly and expertly with their tools. Artists are known by their reputations and have their own client base; when attracting work the reputations of a facility's artists are as important as the reputation of the facility itself, and if an artist moves to another facility they will often take their clients with them. There may also be 3-d artists, who

produce computer generated imagery.

The artists are supported by a number of other departments. The production office is responsible for bringing in work and for the administration of jobs in progress. They need to schedule artist and machine time, and arrange for receiving raw footage and giving clients feedback on the progress of their jobs. Due to the highly unpredictable nature of the business, scheduling is a major problem. Often an artist will have several jobs 'pencilled in' for the same time slot; these are fall back jobs if for any reason the first job cannot be worked on. This may happen for any number of reasons, such as late delivery of material from a shoot.

The machine room is the heart of the hardware infrastructure that underpins the facility. Here tapes are loaded when they arrive, tapes produced for clients, or electronic transfer arranged via a computer network or dedicated video transmission lines. A substantial part of the machine room staff's job is managing the limited online storage resources; a lot of video must be transferred between online storage and tape after hours so that the correct material will be available for the next day.

The library stores archived tapes from all the facility's past jobs. These are stored for various reasons such as reference, legal protection, and because clients often request copies of old jobs, a service for which the facility can charge a fee. The indexing system is simple, with tapes being labelled by the client, date and the job name (which may be quite abstract). These details are stored in a simple database. There is no indexing of the actual material on the tape, although there may be a series of still frames at the start of a tape showing keyframes from shots and their locations on the tape.

The engineering department is responsible for keeping the infrastructure of the facility operational, and they can be expected to do everything from electrical repairs to computer and network administration. Larger facilities may have dedicated software developers or research and development teams to work on bespoke software for various jobs.

2.2.2 Types of work

Post-production facilities deal in the manipulation of video images; behind this description there is an almost infinite variety of jobs that a facility may handle. There are common jobs that are encountered on a

regular basis; removing wires, harnesses or other equipment from footage, or compositing elements shot on a blue-screen stage onto a background plate. However the facility's ultimate job is to realise the client's vision on the screen—a task that requires a large amount of interpretation and inventiveness. Many effects are achieved in a non-obvious way, and even when there is an obvious method to do something there is often an ingenious alternative that requires substantially less effort.

The 3-d department may work on purely computer generated jobs, or sometimes they may need to produce single elements for incorporation into live footage.

2.2.3 Systems

Specialist post-production would not exist without modern computer technology, and understandably computers are a fundamental component in the systems that underpin the facility.

An artist spends most of their time in a suite sitting in front of a SGI workstation running *inferno* or *flame*. Each workstation has its own high-speed disk array for storing video, typically each system can store 2–3 hours worth of uncompressed video at television resolution. Each system also has video I/O hardware to allow the importing and exporting of video to a VTR (Video Tape Recorder).

These systems are connected via a fibre-optic HIPPI network which is used for transferring video between them. They are also connected to a conventional ethernet network, as are all the various Macintoshes and PCs in the facility. In parallel to this data networking there is also a video routing infrastructure—any of the video devices (such as *flame* systems or VTRs) can be routed to any of the others via an interconnect.

2.3 The *Cakes* system

In an effort to assist the management and tracking of the large amount of material that passes through the facility, Smoke & Mirrors embarked upon the development of a database system tailored to the needs of a post-production facility. This led to the setting up of a separate company called Unique-ID [2] to develop and market the system, which is called *Cakes*.

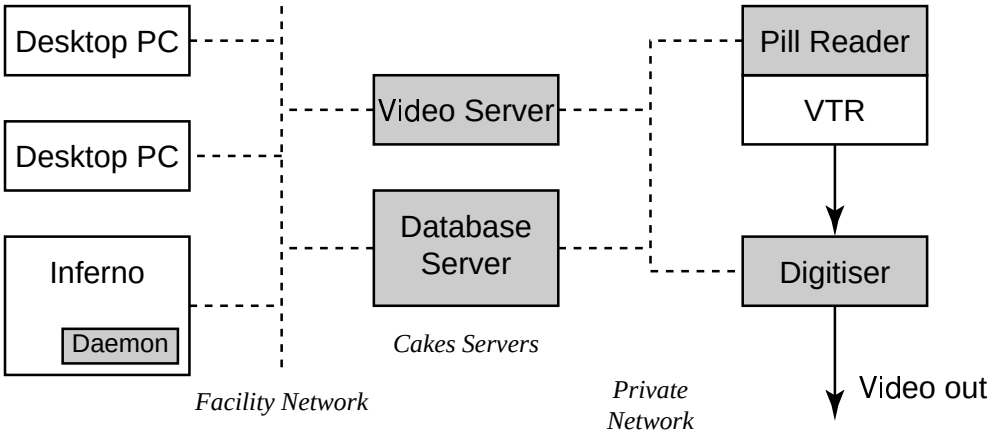


Fig. 2.1: Architecture of the *Cakes* system.

Cakes is a web-based system which knows about the various entities involved in managing a post-production facility, such as clients, jobs, tapes and dubs (copies of tapes). The architecture is deliberately flexible to allow multiple relationships between entities instead of the rigid structure imposed by traditional contact databases, which do not adequately reflect the nature of the post-production industry. Being web-based it can be available on every computer in the facility with a web browser, and even to machines outwith the facility. This means restricted access can be given to clients to monitor the progress of their jobs.

A block outline of the architecture of the *Cakes* system is shown in figure 2.1. A central feature of *Cakes* is that it is integrated into the architecture of the facility and can automatically track material. Physical media such as tapes are labelled with a 'pill', a small transponder with a unique serial number (shown in figure 2.2). A scanner, or 'pill reader' (also in figure 2.2) is mounted above each VTR so that whenever a tape is inserted it can be identified. Each VTR also has a digitiser linked to its video input and output. The digitiser captures reduced resolution copies of every frame of video that is either recorded to or played out from the VTR. The combination of pill readers and digitisers means that the system can maintain a low resolution online proxy of every frame of video that resides on a tape in the facility, providing that the tape has a pill and has been either recorded or played on an equipped VTR. Not all material exists on tape, so the online framestores are also scanned by daemon processes looking for frames that have recently appeared or been

Fig. 2.2: A digitiser and pill reader mounted above a VTR unit, and a pill attached to a tape.

modified.

Pill readers can also be located in other sections of the facility, so the system can track tapes that are entering or leaving the building via reception, or being deposited in the library. The use of pill readers means most of the tracking is automatic, requiring little or no work on the part of the operator, so the system is kept consistent even when people are rushed.

The presence of these online proxies make *Cakes* a powerful system. From any machine in the facility with a web browser a user can quickly scan the contents of any tape. Low resolution video can be streamed over the internet to clients who can view rough cuts or work in progress. Figure 2.3 shows a screenshot of the *Cakes* interface, with several windows showing streaming proxy video.

The proxies are stored online on a video server in Motion JPEG format. This format is used because frame level editing is common, and would require expensive and complicated decoding, splicing and re-encoding if a temporal redundancy compression system such as MPEG were used. At a typical proxy resolution of 176×144 pixels each frame requires around 3-4 kilobytes of storage. With 90,000 frames on a one hour tape the storage requirements are still considerable, but can be accommodated with inexpensive disks. If the facility wishes to keep the proxies online indefinitely then the storage requirements will grow over time as more material passes through the facility; however it is anticipated that the continuingly falling cost of disk storage means that this

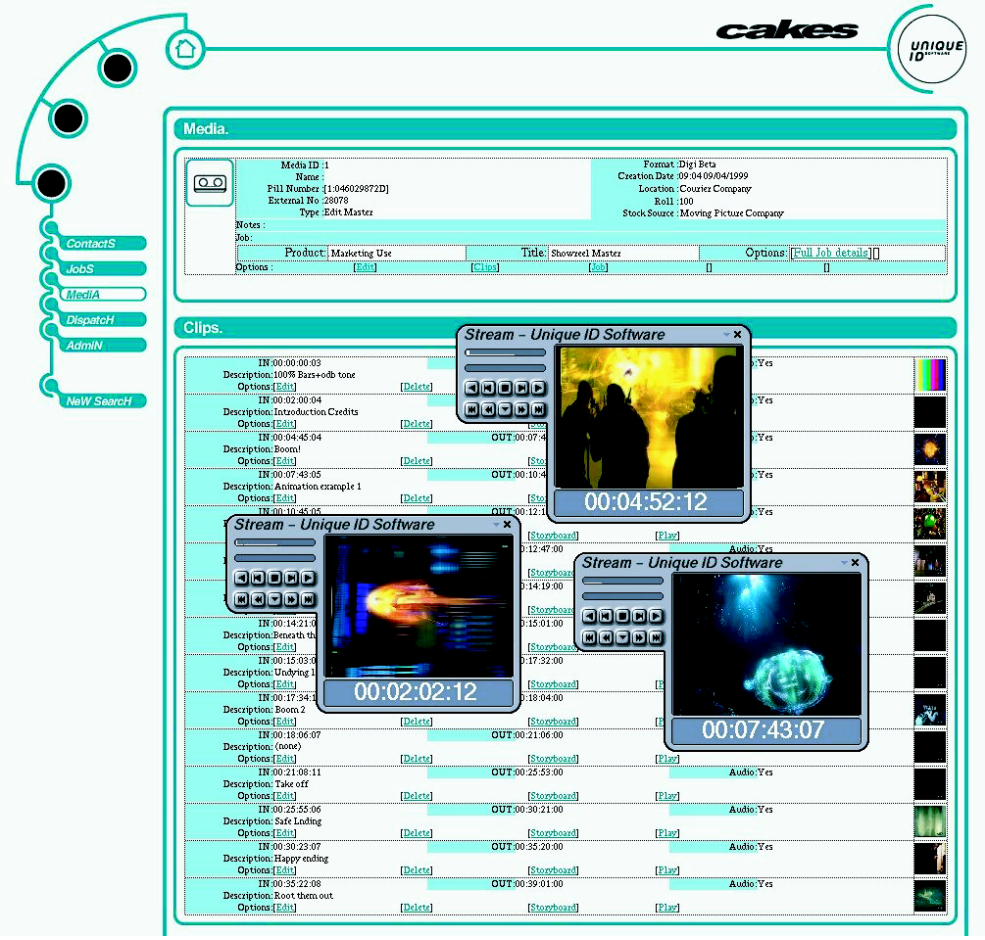


Fig. 2.3: A screenshot of the *Cakes* system, with streaming video proxies.

will not present a problem.

3. Background and proposed approach

This chapter surveys previous work in automatic analysis and processing of video information, looking at proposed approaches to video retrieval, browsing and summarisation. We then describe the different methods used to store and transmit digital video, and the impact these have on processing algorithms. We finish the chapter by outlining our video summarisation technique and justifying our pursuit of this approach over others.

3.1 Video information retrieval

The automatic analysis of the contents of images has long been an area of active research in Artificial Intelligence, and in the last decade the techniques developed for computer vision have been used in combination with those of information retrieval to address the problems of indexing visual information.

Photographic archives have long been indexed using keyword or category based systems, which required users to examine each photograph and assign keywords or categories. There are many obvious drawbacks to such a system, such as the overhead of manually assigning metadata, and misclassification either by accident or due to the subjectivity of the person entering the data. If a limited set of keywords or categories is in use then it may turn out to be too limiting for future uses of the database.

The use of Content Based Image Retrieval (CBIR) techniques has expanded the way visual data can be stored, indexed and searched. Many CBIR techniques (and other image analysis algorithms) use image features, which are computed descriptors that represent certain properties of an image. By precomputing these features for all the images in the database the system can quickly process queries without having to process the actual pixel data of every image for every query.

3.1.1 Image features

Many image statistics have been used for image indexing, including colour, texture and shape. The idea of image features is to compute a concise descriptor for the behavior of an image property over a region. For example a colour histogram can represent with a few values the colour distribution over hundreds of thousands of pixels. Obviously information is lost in the process but the descriptor should be designed so that comparing the descriptors is roughly equivalent to comparing the original images; images that are considered similar by a human observer should have similar descriptors.

Colour is typically represented by histograms, which describe the distributions of different colour values. They discard spatial information, so it is not possible to know what part of the region contains which colours. A feature that preserves the spatial relationship of colours is a co-occurrence matrix, which records the likelihood of two pixels of given colours being a certain distance from each other [100].

Texture can be represented many ways. One is to record the responses of a bank of carefully chosen filters such as Gabor filters [25] to the texture. Another is to transform the texture region into a frequency domain, for instance with a fourier transform, and record the dominant frequencies. Tamura et al. [102] define three human-friendly properties of textures which can be measured: coarseness, contrast and directionality.

The shape of a region is also an important perceptual cue. The outline of a region can be represented by locating the zero crossings of its curvature function [62], which have been identified as important in our perception of shape. The important zero crossings are located by progressively smoothing the outline to remove noise and recording the locations along the boundary of the crossings that remain, producing a scale and rotation invariant feature descriptor.

3.1.2 Query by example

In a query by example system the user provides an image representing what they are looking for and the system returns images from the collection that it deems to be similar. When images are entered into the database a number of low-level features are computed over the image and stored. When a query is made the same features are computed over the

query image and compared to the database to find the closest entries. The computed features result in a number of scalar values which are collected together in a vector that positions the images in a high dimensional feature space. High dimensionality indexing techniques are used to store the vectors for the indexed images and perform nearest-neighbor queries.

Once the user has submitted their query image they are returned a ranked set of images from the database that are considered similar using the system's image features and similarity metrics. The user can then select one of these images as a query image to submit in a new query, hopefully refining their search.

Nastar et al.'s Surfimage system [66] adds a relevance feedback mechanism to this loop. From the returned images the user can mark them as relevant or not relevant. This information is then used by the system to modify the weightings of the different image features it uses.

When computed over the whole image image features don't fully represent the areas of the image that the user is interested in. Therefore many CBIR systems, such as the Blobworld system [19], pass images through a segmentation stage and compute features for the computed regions. Smith and Chang's VisualSEEk system [91] also takes into account the spatial relationship between regions.

An interesting approach that attempts to represent many image properties with a single feature uses the wavelet decomposition of the image [97]. The locations of the n most significant wavelet coefficients for each image in the database are stored, and compared with the wavelet transform of the query image.

The major drawback of query by example systems is the need for a query image. This requires that the user have a concrete idea of what they are searching for and are able to provide a graphic representation of it. It also limits the user's view of the database as images that are not close to the query image will never be revealed to the user.

3.1.3 Browsing

Browsing systems instead allow the user to explore the database interactively, allowing them to discover images they may not have otherwise considered, or even known were in the database. Rather than return-

ing results from searches, browsing systems present a structured view of the database and provide tools to navigate around this framework. As image databases are usually stored as a high-dimensional index of image features browsing systems employ various methods of dimensionality reduction to present the feature space in 2 or 3 dimensions, while attempting to keep images with similar features close to each other.

Craver, Yeo and Yeung [20] use space filling curves to perform dimensionality reduction. By filling the high dimensional space with two curves each point in the higher space can be represented in 2-d by its position along each of the curves. The user can explore the space by moving along either of the curves to find neighboring images.

The El Niño system [81] allows users to control the dimensionality reduction process. The images from the database are initially projected in a 2 or 3 dimensional space. The user can then manipulate this space by moving images around, bringing images that they consider to be similar close to each other. The system then calculates a new projection to reflect these changes.

3.1.4 Automatic classification

Content based analysis can also be used in the context of a traditional classification or keyword based indexing system by training the computer using example images for each category or keyword.

This can be done by manually marking up a training set of images using a chosen set of keywords and using it to train a neural network or other statistical classifier. The Photobook system [73] instead has models for specific objects, such as faces, and detectors to locate them within an image. The models are tailored to their objects, and so are better able to describe and differentiate between examples than generic features such as colour or texture models.

3.1.5 Video

All of the above approaches have been extended to video by adding features based on motion. The extra temporal dimension of video presents additional difficulties, particularly with the volume of data that must be processed and with presenting material efficiently.

A simple approach is to represent each clip with a single static image, such as a frame from the clip, and apply static image retrieval techniques to that. There have been many approaches suggested for selecting or synthesising keyframes for a video sequence, but no static image can convey all of the information contained in a moving video sequence.

Motion of regions has been added as a feature to query by example systems, allowing object and camera motions to be indexed. The user can be provided with a means to sketch approximate motions and retrieve sequences that contain the same motion, in tandem with other search criteria, such as in IBM's QBIC system [29].

Although presenting video in a video indexing system doesn't present much of a technical challenge it does cause problems with the efficient usage of the system. Much attention has thus been given to devising novel ways to allow a user to efficiently browse a video collection.

Video summarisation

Video summarisation techniques try to preserve the important features of a sequence in a manner that can be understood by an observer in less time than it would take to view the video itself. The suggested approaches can be split into those that create a static representation and those that produce a shorter moving sequence.

The idea behind producing a reduced time video sequence is that it can still include the semantically important parts of a sequence, but be viewed in a fraction of the time of the original sequence. This can be done through a combination of speeding up video and removing sections. There are limitations however, as humans have difficulty taking in information if it is presented too fast, placing a limit on how much a clip can be temporally compressed. Audio information also becomes incoherent when sped up or chopped up in this manner. Nam and Tewfik [64] produce a temporally compressed video sequence by adaptively subsampling the original. The sampling rate is determined by an estimation of how important a segment of video is, with more important parts being sampled at a higher rate. They identify important segments of video by looking for 'emotional dialog' and 'violent action', which are identified through a combination of video and audio analysis. Smith [90] presents a video browser application that is designed to be used where

only a low bandwidth connection is available. Initially a reduced spatial and temporal resolution sequence is displayed, which can be refined by selecting a segment and viewing it with increased resolution. Multiresolution representations such as wavelets are well suited to implementing such a system.

The other way of temporally compressing video is to select a number of small clips from the original sequence that will be combined in a summary. Smith and Kanade [92] produce a 'skim' of a video sequence by looking for a number of visual and audio cues. They also perform speech recognition on the audio and assign relevances to the words in the transcript. A compressed soundtrack is generated by combining important words from the original audio, and clips are then chosen from the video to go with the audio. Pfeiffer et al. [74, 53] explore the idea of compressed video in the context of automatically generating movie trailers. They identify types of material from the movie that are often used in trailers, such as actors' faces, conversations, action sequences and the title. They then analyse the whole movie looking for examples of this material and synthesise a new sequence of a specified target length.

Static 2-dimensional summaries have been more popular. These typically try to produce a collection of still images that represent the contents of the sequence. These images may be frames from the original sequence, or synthesised based on the contents of the sequence. Again the aim is to try and represent the most salient details of the sequence. Often 2-dimensional summaries also try to show the relative importance of the different parts of the sequence, or to show the semantic structure of the sequence. These 2-dimensional summaries have several advantages over temporally compressed sequences; they can be viewed by a human much faster, require less storage and network bandwidth and no specialised equipment to view (such as a computer—a summary can be printed and distributed on paper).

Such approaches often represent the semantic structure of video on four levels, as shown in figure 3.1. At the lowest level are the individual frames. A level above this are shots, which result from individual operations of the camera and can be joined with a variety of transitions. A scene is a consecutive sequence of related shots, representing a single situation or location in a larger program. At the top level is the sequence itself, which may be a single episode in a series, a movie in a database,

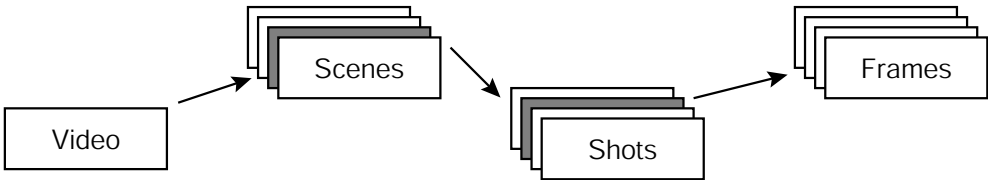


Fig. 3.1: The hierarchical structure of video.

or any other entity in a video collection. When processing video in a computer the input is simply a sequence of frames (possibly with an associated audio channel). The only thing of which we can be certain is the frames—all structure above this must be inferred from the contents of the video and possibly audio channel. Shots can be considered as the syntactical level of video structure and we will examine them further in Chapter 4.

Static summaries are often referred to as storyboards. A storyboard is a pictorial representation of a sequence that is used as a planning aid in production [37]. Based on a script, concept art, and advice from the director and writer, a storyboard artist will produce a series of small drawings, each one representing a single shot. A storyboard, such as shown in figure 3.2, shows the principal contents and actions of each shot, how they are framed, and how objects and the camera will move.

Storyboards have become an essential part of the pre-production process. The storyboards can be filmed along with a scratch soundtrack to produce an *animatic*, which can be viewed as a very rough cut of the final product and used to identify any script problems. For complex shots, storyboards show what is going to be within the frame and so can be used to determine what sets need to be built, and importantly how much of the set is going to be visible, so there is no need to build anything that is never within the frame. In modern big-budget productions animated 3-dimensional visualisations are often used to plan complex action or special effects shots.

A simple approach to generating a 2-dimensional summary is to take sample frames at regular intervals and present them as a simple storyboard, but this has obvious drawbacks. The sampling rate can be adapted to try and allocate more keyframes to the important parts of the sequence, and less to parts where relatively little happens. There are many different ways of deciding how to vary the sampling rate, for

Fig. 3.2: Part of the storyboard for the film *Jurassic Park* (1993) [87].

example based on the length of each shot [75] or using MPEG-7's motion activity descriptor [23].

Since a conventional storyboard will usually have one image per shot, we can use a shot change detection algorithm to identify the shots in a sequence and use one image per extracted shot. This image can be a single *keyframe* selected from the shot. A simplistic approach is to take the first, last or middle frame from the shot, but this may miss out important features of the shot because they are not present in that particular frame. A variety of keyframe selection techniques have been proposed in the literature [43, 105, 72].

It is also possible that there will be no single frame that adequately represents a shot. It is instead possible to synthesise an image that contains the important aspects of all parts of the clip. If the camera moves during a shot then the movement can be tracked and the different parts of the background stitched together to create a panorama [104, 103]. Another form of synthesised image is Arman et al.'s 'R frame' [12] which is a single keyframe annotated at the sides with information about motion and the length of the shot.

Once keyframes are chosen there are different ways they can be presented in a storyboard format. Different shots can be perceived as more important than others and be given more space on the storyboard. Uchihashi et al.'s 'Video Manga' system [108] presents the story board in a comic book style. Keyframes for shots that are determined to be more important are given more space, and a packing algorithm used to arrange the variably sized images on the page.

The methods discussed so far represent a video sequence as a purely linear structure, but as we have seen video can also be considered as hierarchical, with several linked shots constituting a scene. Determining what shots are in a scene is a subjective process, but there are various cues that can be exploited. Shots from the same scene will often be set in the same location and usually have similar colours. If audio is available then the sound for the scene will usually be unbroken, as the scene occurs in real time. Such cues are used by Pfieffer et al. [74] in their movie trailer generator. Ferman and Tekalp [27] attempt to analyse the higher level structure of a sequence by using the different types of transition and other cues such as motion continuity over cuts.

Clustering techniques can be used to find similar shots that may con-

stitute a scene. Yeung and Yeo [117, 116] use such a method, but with an additional time constraint as only consecutive shots should appear in a scene. They segment a video sequence into 'story units', which are a semantic level between shots and scenes, and build up a 'Scene Transition Graph', which is a directed graph showing the relationships between the story units. Clustering can also be applied to the extracted keyframes from shots to build a hierarchical tree of keyframes [99, 69].

3.2 Digital video

Digital video is now prevalent in the broadcast world but the need to maintain compatibility and effect change gradually has constrained many of the decisions in how the digital broadcast chain is implemented.

In the UK broadcast video uses the PAL standard, which defines interlaced frames of 576 scanlines at a frequency of 25Hz. Interlacing means that each frame is transmitted as two fields, one consisting of the even scanlines and one of the odd lines. The fields are captured one after the other, so the second field represents the scene 1/50th of a second after the first. This is visible as a tearing beween adjacent lines when a video frame is frozen, but appears as smooth movement when viewed on an interlaced display.

Analogue video is transmitted as three components: one of luminance and two representing colour. This is partly to maintain compatibility with black and white sets, which only display the luminance component, but also as a form of compression. The eye is less sensitive to changes in colour than luminance so the colour components can be allocated less bandwidth.

Digital video has inherited many of these characteristics. In the United Kingdom digital video is 720 by 576 pixels at 25 interlaced frames per second. This is used for both standard 4:3 aspect and 16:9 widescreen material (almost all new material is widescreen). For widescreen material the pixels are stretched to an aspect ratio of 1.42.

The SMPTE have defined standards for the interchange of digital video. The SMPTE 259 M standard defines the transmission of video over a Serial Digital Interface (SDI) at a rate of 270 Mbit/s. Video is encoded into a luminance channel Y and two chroma channels Cb and Cr, and sampled at a ratio of 4:2:2, with two samples of Cb and Cr for

every four samples of Y. This subsampling is done horizontally along each scanline.

The Y, Cb and Cr values are calculated from RGB values as

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \begin{bmatrix} 65.738 & 129.057 & 25.064 \\ -37.945 & -74.494 & 112.439 \\ 112.439 & -94.154 & -18.285 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (3.1)$$

SDI video is used for interconnecting studio equipment but the data rate is too high for storage and broadcast, so a number of compression systems have entered use. This is even more so for High Definition (HD) material, which has resolutions of up to 1920 by 1080 and uncompressed data rates of 1.5 Gbit/s.

3.2.1 DCT based compression

There are now a plethora of compressed formats in used, mostly used by different tape formats from competing manufacturers, but all are based on the Discrete Cosine Transform (DCT). The DCT is a spatial to frequency space transform and its use in compression is based on the assumption that most detail in photographic images is relatively low frequency compared to the resolution of the image, and so high frequencies can be removed from the image without being noticed.

Figure 3.3 demonstrates the compression of an 8 by 8 pixel block using the DCT. Although the block is taken from a detailed region of the image there are no strong edges at this small scale. The pixel values are transformed to give an 8 by 8 block of frequency coefficients. The low frequencies are represented in the upper left (with the DC value in the top left) and the higher frequencies in the lower right. Typically the lower frequencies dominate. The coefficients are then quantised, which results in many of the high coefficients having zero value. These quantised values are then reordered into a 1-dimensional array using a zigzag pattern that puts the coefficients in frequency order. This data is then compressed with a Huffman compression routine. The efficiency of the Huffman compression is improved by the grouping of the low valued coeffiecients by the zigzag reordering. When the inverse process is applied the important low frequency values are preserved. Many DCT based

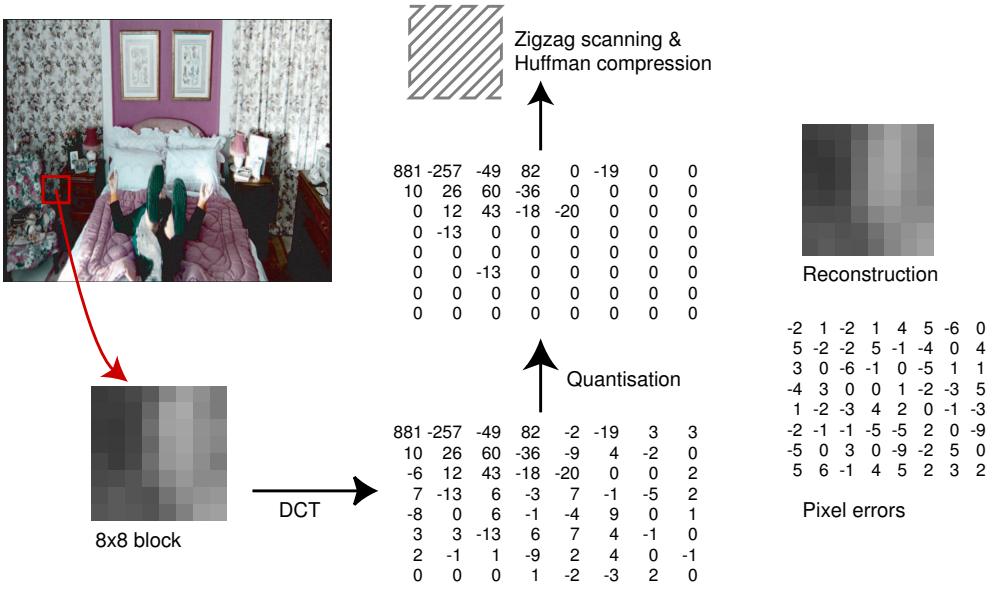


Fig. 3.3: Compression of a 8×8 pixel block using the DCT.

schemes will additionally subsample the chroma channels at a 4:1:1 ratio, so a 16 by 16 pixel block is represented by a 8 by 8 block in the Cb and Cr channels.

The JPEG image compression format works like this and many compressed video formats work by applying this compression to each frame independently, which is commonly called 'Motion JPEG'. However additional compression can be gained by exploiting the temporal redundancy in video.

In most video much of each frame is very similar to the previous frame. Rather than recoding these areas for every frame, they can be referred from previous frames. The MPEG system used on DVDs and broadcast television does this, compressing video and audio to a bandwidth of 8 Mbit/s.

MPEG works with groups of frames, typically 15 at a time. Frames are encoded one of three different ways. *I frames* are encoded as JPEG images. They are independent of other frames and provide an entry point where a decoder can begin decoding the stream. *P frames* are encoded by referring to blocks from a previous I or P frame through a process called motion estimation. The encoder searches for similar blocks in the reference frame and encodes the offset vector and (small) error between

the blocks. Not all blocks have a good match in the reference frame, so these can be encoded normally. *B* frames are encoded using two reference P or I frames, one before and one after the current frame.

Each group of pictures (GOP) is encoded as IBBPBBPBBPBBPBB. The first frame is encoded as an I frame, then the fourth as a P frame and the intermediate B frames can then be encoded. The next P and B frames are then encoded until the end of the group when a new group is started with another I frame.

Although this results in greater compression than Motion JPEG it adds considerable complexity. Encoders and decoders need to code frames out of order and buffer them, which adds delay. Editing an MPEG stream is difficult as a whole group of frames must be decoded, modified and re-coded. For these reasons MPEG is rarely used in studio equipment, but instead for final delivery where bandwidth is scarcer and no further modification to the stream is required.

3.3 Our approach

Discussions with people at Smoke & Mirrors identified several ways that video indexing could be utilised. An obvious area is the library, where the simple indexing system is often insufficient to precisely locate material, and often several tapes will have to be taken out and viewed manually to find what is needed. Queries to the library are typically for a whole video or commercial, which is required for reference, PR purposes, or to show as an example of previous work to potential clients.

Another potential point of use is when an artist is looking for a video element to use in a composited shot. The search criteria in these instances are often very vaugue. Sometimes a specific object or location is required but more often the artist is seeking something that "fits in" with the look they are trying to create. It may be a certain kind of texture or movement, and they are often unable to articulately describe what they are looking for, although they say they will know it when they see it. Often the actual footage they select is quite different what what they express as the search criteria, but they have seen a way that they can manipulate it to get what they want.

A query system may be suitable for the case of the library, where the user will have knowledge of what is in the target video and can express

that as a query. However a browsing system would also bring advantages to the library. Since the search can already be narrowed down using the existing metadata (client, job name, date) a browsing or summarisation system could accelerate the process of reviewing the returned tapes.

The case of the artist looking for elements is much more difficult. The terms of the query are difficult for a human to express and much more difficult to quantify in a way that can be communicated to the computer. The criteria used to judge the suitability of a clip are often too subtle to be differentiated by the low level image features used by indexing systems. A browsing system is much more suitable in this case, allowing the user to make their own judgments about what is suitable, but assisting them in navigating the database. The user may also only have a vague idea of what they want, and a browsing system will allow them to look at what is available.

3.3.1 Storyboards from video

We have decided to focus on developing a video summarisation system to aid a library-type query. The target sequence is either a self contained commercial or promo, or a shot within one of these. Our approach will be to create storyboards from these video sequences. The storyboard provides a quickly understood overview of a sequence at the shot level; for the length of sequence we are concerned with the storyboard will fit on one or two pages. Commercials and promos have little narrative structure and in the post-production environment it isn't important anyway, so there is nothing to be gained from having a structured representation beyond the shot level. Macer et al. [58] have already pointed out many of the advantages of a storyboard representation. an additional advantage is that the storyboard format is already familiar to most users from its use as a preproduction tool.

The storyboard frames must show the principal attributes of the shot. These are the main objects and their movement, camera movement, and the shapes and colours of the principal objects. We believe that a vector representation is most suitable, as this gives several advantages over the usual bitmapped keyframe: smaller storage size, resolution independence and the ability to layer and optionally display different parts of the image.

We develop a system to take an original video sequence, consisting

of only video frames and without any associated audio or metadata, and produce a storyboard-style summary of the sequence. We divide this process into three principle steps:

1. Temporal segmentation (chapter 4): Identification of the individual shots in a sequence. Existing shot change detection algorithms perform poorly on the highly dynamic material typically encountered in a post-production environment, so we develop new algorithms for detecting cuts and fades in highly dynamic material.

2. Motion segmentation (chapter 5): Identification of the principal objects within a shot and their motion. Motion analysis and segmentation is a very difficult problem. We take an existing motion segmentation framework and improve it by tuning or replacing the algorithms used in its various stages.

3. Presentation (chapter 6): The rendering of the extracted information in a storyboard form. We take the output of the motion analysis stage and produce a vector drawing representation of the shot, annotated with motion information.

4. Temporal segmentation

This chapter describes our research into the temporal segmentation (also called *shot change detection*) of video material in a post-production environment. Structurally a video sequence can be decomposed into scenes, which can again be decomposed into shots as shown in figure 3.1. A shot is defined as one uninterrupted operation of the camera [42]. A scene is a number of conceptually related consecutive shots, for instance showing events located in the same time and location before the next scene moves on to another time and location. Scenes can be thought of as the semantic level of the language of film while shots are the syntactic level.

4.1 Types of transition

Shots are joined by transitions and there are a large number of possible transitions which can join one shot to another. However the majority of those encountered fall into one of four categories (examples are shown in figure 4.1):

cuts The simplest transition, where the last frame of one shot is followed immediately by the first frame of the next. It has an effective length of zero frames.

fades The first shot fades out to a blank (usually black) screen, and the following shot fades in.

dissolves The effect of fading out one shot while simultaneously fading in the next; the first shot 'dissolves' into the next.

wipes A wipe is a spatially varying transition between two shots; the shots themselves may also move, or may be static in the frame. Many varieties of wipe are possible by changing the pattern used to make the transition, for example the star shape shown in the figure.

All transitions except cuts are *gradual transitions*, they are incremental changes through a number of consecutive frames.

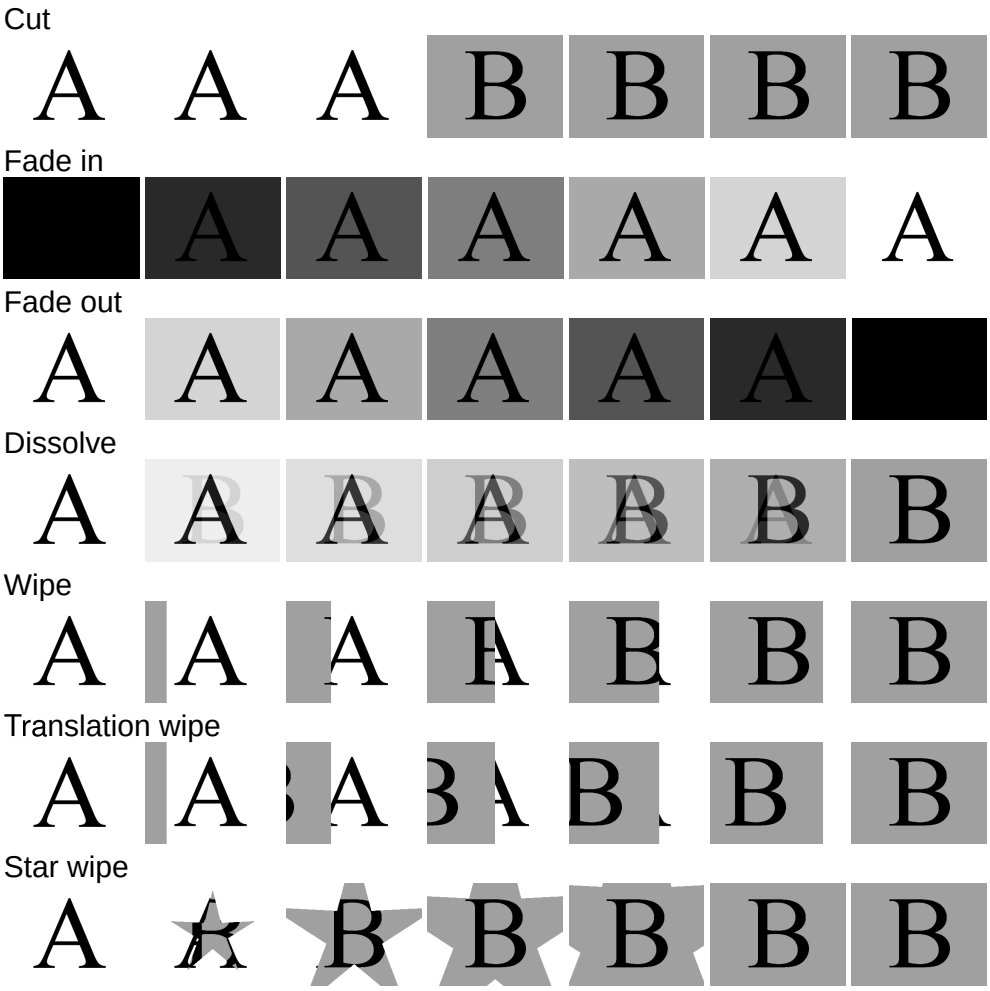


Fig. 4.1: Different types of transition.

Modern digital editing systems make a huge number of transitions possible, and make it easy to devise novel transitions which may not fall into the above categories.

4.1.1 Transitions in context

Cuts, fades and dissolves have been achievable with optical printers for many years, and they have become well established in the film editing vocabulary. Over time they have acquired certain semantic meanings in film and video, which audiences understand on a subconscious level.

Gradual transitions are often used to separate scenes, while cuts are used within scenes to separate shots that are occurring in continuous time in a single location. Dissolves often indicate the passage of time; a dissolve between two shots of the same location indicates that we are returning to the same place, but later in time. Dissolves are also used in montages, which are sequences of related scenes, often with music, which show some significant concept or event [42].

4.1.2 A transition model

In understanding transitions, how they are produced and how they can be detected, it can be useful to use a model. Hampapur, Jain and Weymouth [35] formulate a model of how gradual transitions are produced. Two shots, S_1 and S_2, are combined into the transition sequence E by applying a spatial and chromatic transform to each. Each pixel x, y of every frame t of the resulting sequence is some chromatic and spatial combination of pixels from the two original sequences. The chromatic transform T_c doesn't change the position of pixels but can modify their colour values and also assign an *alpha* value to each pixel which is used by the compositing operator, $\otimes$, to mix the transformed pixels [77]. The spatial transform T_s doesn't modify the colour but can map pixels from one position to another within the frame. The transformed shots are composited to form the final sequence:

$$E(x, y, t) = (S_1 \times T_{s1} \times T_{c1}) \otimes (S_2 \times T_{s2} \times T_{c2}). \tag{4.1}$$

The fades, dissolves and wipes can be produced by using fairly simple transforms. Fades and dissolves are purely chromatic effects, so both

T_{s1} and T_{s2} are identity transforms. The chromatic transforms lower the intensity of S_1 over time and increase that of S_2. For a fade in/out the intensity of S_1 is lowered completely before that of S_2 begins to increase. The colour used between the shots is usually black, but does not have to be. During a dissolve the intensities of both shots are adjusted simultaneously although the transforms of each shot do not have to occur over the same time scale or at the same rate.

Wipes are spatial edits, with S_2 gradually replacing S_1 on a pixel by pixel basis; each pixel in E is from either S_1 or S_2. Each T_c is an identity transform, while the T_s transforms map pixels from the source frames to their new locations. Any number of spatial transformations are possible, from simple translations to spirals and page curl effects.

Hampapur, Jain and Weymouth suggest the transformations be represented by matrices which are multiplied by either the spatial vector of each pixel in the shot, $[x, y, t]$, or the colour vector $[r, g, b]$. Although this is sufficient to represent the simple common transitions, modern digital editing equipment can produce far more complex edits, which can still be represented by the general model.

Others have suggested similar models, and used them to derive their own temporal segmentation approaches [30, 96, 118, 36]; however these are explicit models of individual transitions, unlike the general model above.

4.2 Previous work

Temporal segmentation is an unavoidable problem in many video processing tasks, and there is a large body of work aimed at automatically decomposing a video sequence into its constituent shots. There are many different ways of building a taxonomy of shot change detection techniques; important traits are the types of transitions detected and whether the algorithm processes uncompressed images or works directly on a compressed format.

Almost all digital video is stored and transmitted in some compressed format. Many algorithms have been proposed that work directly on compressed video, saving the computational and memory overheads of decompression—also, the compression schemes themselves apply significant analysis to the video, which can be exploited for detecting shot

changes. MPEG is the most popular format for video compression, and the compressed data is useful in a number of ways. Firstly, a 1/8 resolution subsampled image can be quickly extracted by taking the DC coefficient from each macroblock; this is called the *DC image*. The AC coefficients can also be used to extract edge information without having to do the full inverse DCT [50]. Algorithms that use motion information can utilise the MPEG motion vectors. Although these do not necessarily reflect the true motion of the scene as they are chosen to optimise the coding, they can save the need to carry out an expensive optical flow computation.

In this review we will look first at techniques that detect cuts only and then at those that detect gradual transitions, which build on the techniques introduced by the cut detectors. Finally we will look at some of the integrated approaches to detecting multiple types of transition.

4.2.1 Identifying cuts

Cuts are the easiest type of transition to detect, and have received the most research attention and met with the most success. Almost all cut detection schemes are based on the assumption that there is more similarity between frames within a shot than between frames from different shots. The most popular approach therefore is to define some way of measuring the similarity (or conversely the difference) between two frames and look for pairs of consecutive frames that show a large dissimilarity. Several different methods of measuring similarity and of locating the peaks that indicate cuts have been proposed.

Similarity metrics

Image similarity is a concept that has been studied in depth in the closely related area of content based image retrieval, and many of the similarity metrics developed in this field have also been applied to scene change detection. Most are based on using low-level features of the image which are easily computed.

Pixel statistics The simplest similarity metrics are calculated on the pixel data directly, treating each image as a vector of values and using vector distance metrics such as the Sum of Absolute Differences,

$SAD = \sum_{x,y} |I_1(x,y) - I_2(x,y)|$ or Sum of Squared Differences, $SSD = \sum_{x,y}(I_1(x,y) - I_2(x,y))^2$ which can either be calculated on the intensity image alone, or over all the colour channels. These are very sensitive to noise so a threshold can be introduced at the pixel level with the threshold chosen to be larger than the amount of intensity variation caused by noise. The metric then becomes the count of pixels whose change is above the threshold. Movement also causes pixels to change, and camera motion can cause sizeable pixel differences across the entire image; prefiltering the images can help reduce the effect of small camera movements [120].

Simple statistical properties of the pixel values such as the mean and variance of pixel intensity values can also be used, and tests applied to decide if two frames share the same statistical distribution of intensities. Ford [30, 31] lists several standard statistical tests including the likelihood ratio for two images with means μ_a and μ_b and standard deviations σ_a and σ_b,

$$\lambda_u = \frac{\left[\frac{\sigma_a+\sigma_b}{2} + \left(\frac{\mu_a-\mu_b}{2}\right)^2\right]^2}{\sigma_a\sigma_b}, \tag{4.2}$$

or

$$\lambda_n = \left(\frac{\sigma_0^2}{\sigma_a^2}\right)\left(\frac{\sigma_0^2}{\sigma_b^2}\right) \tag{4.3}$$

assuming a normal distribution and where σ_0 is the standard deviation of the pooled values from both images. Sethi and Patel [86] also use this test, and attribute it to Yakimovsky. Other standard statistical tests suggested by Ford are Snedecor's F-test,

$$F = \frac{\sigma_a^2}{\sigma_b^2}, \text{ where } \sigma_a > \sigma_b \text{ and } F \geq 1, \tag{4.4}$$

and Student's t-test,

$$t = \frac{\mu_a - \mu_b}{\sqrt{\sigma_a^2 + \sigma_b^2}}. \tag{4.5}$$

Ford also suggests two tests of his own devising,

$$\lambda_1 = \frac{|\mu_a - \mu_b| \times |\sigma_a - \sigma_b|}{\sigma_a \sigma_b \left(\frac{\mu_a + \mu_b}{2}\right)},\ \lambda_1 \geq 0 \tag{4.6}$$

$$\lambda_2 = \left(\frac{\mu_a \sigma_a}{\mu_b \sigma_b}\right)^2,\ \text{where } \mu_a > \mu_b, \sigma_a > \sigma_b \text{ and } \lambda_2 \geq 1. \tag{4.7}$$

Since the mean and standard deviation are global statistics computed over the whole image they have a certain amount of tolerance to motion. However such simple statistics are quite a blunt instrument and two very different distributions can have the same mean and standard deviation. These statistics also only utilise the intensity information in the image, making no use of the additional information available in the colour channels.

Colour Histograms Histograms are a more sophisticated statistical descriptor of an image. They also introduce a large amount of choice for the algorithm designer—what data to calculate a histogram of, how to bin the data, and how to compare two histograms. A number of researchers have experimented with histogram based techniques, comparing the performance of different established methods from the statistics literature, and inventing novel techniques tailored for images.

In computing the histograms the main choices are:

Colour space Histograms can be computed on the greyscale pixel intensities, or using all the available colour channels. A colour histogram can be computed in any colour space; common choices are the RGB space and the YUV space used in JPEG and MPEG compression. These colour spaces are very machine orientated and are often criticised for being far removed from how humans perceive colour. For this reason the more perceptually based HSV and $L^*a^*b^*$ colour spaces are sometimes used.

Histograms based on three colour channels are often sensitive to illumination changes; for instance a person walking out of shadow into sunlight can have a pronounced effect on the histogram. This can be countered by using only chroma information to build the histogram; for instance using only the U and V channels of the YUV colour space. Wei, Drew and Li [111] compute what they call

chromaticity histograms by first normalising each of the R, G and B channels over the frame so that the values in each channel sum to one, then define two chromaticity components, $r = R/(R+G+B)$ and $g = G/(R+G+B)$.

Bins To have a histogram bin for each individual possible colour would require a large amount of storage and not be much use; the histograms would be highly susceptible to noise. Instead the colour space is quantised first to reduce the number of different colours. Usually the colours are quantised into evenly distributed bins—if the number of bins is a power of two then this can be done by simply taking the most significant bits of each value, for instance to quantise an 8-bit colour value into 8 bins we take the three most significant bits.

As observed above the most popular colour spaces are not perceptually uniform, so it is possible to use varying sized bins to compensate. However this is more of an issue in content-based image retrieval than for shot change detection.

Dimensionality Colour images typically have three separate colour channels, which gives us the choice of computing one 3-dimensional histogram for the entire colour space, or computing separate 1-dimensional histograms for each channel. 1-dimensional histograms have the advantage of being easier to compute and requiring less storage; $3n$ bins rather than n^3 when each channel is quantised to n values.

Many different metrics exist for comparing two histograms. If we treat the histogram as a vector then the usual vector distance metrics such as the L_1 and L_2 distance can be used. The normalised inner product of two histograms has also been used [31]. Histogram differences are well explored in the field of statistics, and several standard metrics exist such as the χ^2 test (with N degrees of freedom),

$$\chi^2 = \sum_{i=1}^{N} \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)}, \tag{4.8}$$

and the histogram intersection,

$$I = \frac{\sum_i^N \min(H_1(i), H_2(i))}{N}. \tag{4.9}$$

The Kolmogorov-Smirnov test is another popular statistical test, first suggested for use in shot change detection by Sethi and Patel [86]. If $CDF(i)$ is the cumulative distribution function of a histogram up to bin i, then the Kologorov-Smirnov test is computed as

$$ks = \max_i |CDF_1(i) - CDF_2(i)|. \tag{4.10}$$

Many authors have used different combinations of histogram and metric, including comparative studies which have examined the impact of the different choices [31, 33].

Although these histogram measures can be fairly effective, they still have some weaknesses. Two neighbouring shots may have similar histograms, particularly if they are of the same location. As cuts are used between shots within a scene, which is usually restricted to a single physical location, this is a common occurrence. To counter this researchers have introduced some spatial information to the histograms by partitioning the frame into subblocks and computing histograms for each one, the idea being that although two shots may have similar global histograms the local histograms will be different, due to the different positions of objects in the frame. Also, object motion will only affect a subset of the blocks. Nagasaka and Tanaka (cited in [17]) combat problems caused by object motion by discarding the blocks which show the largest changes, while Toller, Lewis and Nixon [106] discard the 25% of blocks with the least change and the 25% with the largest change. Boreczky and Rowe [17] count the number of subblocks that exceed a threshold.

Motion features Motion can be an important cue in detecting transitions as motion can be expected to be consistent between frames within a shot, but not over a shot boundary. Estimating motion within a scene can help differentiate between camera or object motion and transitions, which cause problems for many shot change detection techniques. Motion vectors are expensive to compute, but precomputed motion data may be readily available in some compressed video formats, in particular MPEG. It is important to note that the motion vectors used in MPEG

encoding are chosen to maximise the compression rate and are not designed to reflect the true motion within the scene like the optical flow algorithms use in computer vision; however MPEG motion vectors do generally approximate the true motion.

Fernando, Canagarajah and Bull [28] work on the assumption that motion vectors over a cut will be somewhat random in nature, whereas within a shot they will be correlated. They compute the mean motion vector for a pair of frames, and then sum the Euclidean distances of all the motion vectors from the mean.

Rather than using motion vectors, Xu, Zhu and Stentiford [114] use a robust M-estimator to find the dominant motion in the shot, modelled as a 2D affine transformation. For each frame they find the number of *supporting pixels*, pixels which fit the computed model. Their hypothesis is that cuts will produce a change in the proportion of supporting pixels in the frame. Motion estimation using the robust M-estimator is very computationally expensive, however. Vasconcelos and Lippman [109] also compute the dominant affine motion, using it to register the previous frame to the current frame and taking the residual error between the two registered frames as a difference measure.

Image features Image detail is something else that is coherent within a shot but not between shots. Zabih, Miller and Mai [119] track edges in the image to detect transitions; during a transition a large number of edges enter and leave the scene. They define the entering edges ρ_{in} to be the fraction of edges that are not within a set distance of any edge in the previous frame; the threshold distance allows for movement. Similarly ρ_{out} is the fraction of edges in the previous frame that are not within a set distance of any edge in the current frame. A distance measure is then defined as $D = \max(\rho_{in}, \rho_{out})$. Motion compensation can be applied as a preprocessing step to reduce the effects of motion.

In DCT-based compression schemes detail is represented in the high frequency DCT coefficients. Lee, Kim and Choi [50] show that an edge image can be extracted directly from a MPEG compressed stream. Arman, Hsu and Chiu [13] present a distance metric that operates on DCT coefficients directly; the coefficients for corresponding blocks in two images are concatenated into vectors and the normalised inner product between the vectors is computed, with higher values indicating more

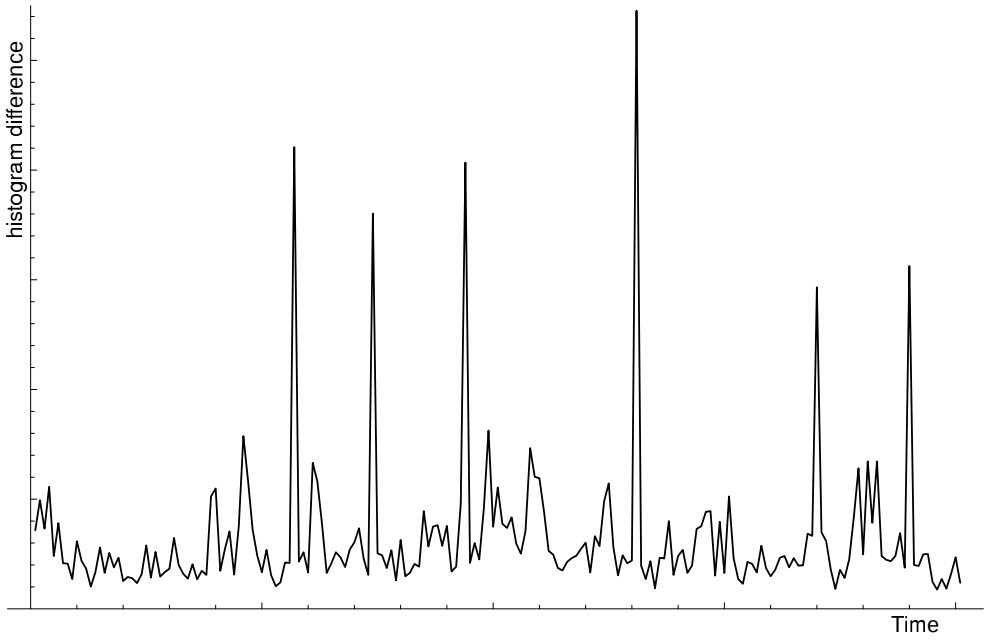


Fig. 4.2: A histogram distance metric applied to an example video sequence. The large spikes are cuts.

similarity.

Porter, Mirmehdi and Thomas [76] correlate blocks in the Fourier domain, doing a block search to find the best match for each block, and using the mean of the correlations of the best matches as a difference measure.

Kobla, Doermann and Lin [46] rely on the MPEG encoder to decide how similar frames are—the MPEG encoder encodes macroblocks one of three different ways depending on whether it can find a suitably similar block in a nearby frame. They use the count of each type of block coding to decide if a cut has occurred.

Locating cuts using a metric

The above techniques provide a similarity (or dissimilarity) measure for each pair of consecutive frames. Figure 4.2 shows a histogram distance metric applied to a short video sequence; the cuts are indicated by the sharp spikes.

A global threshold could be used to identify these spikes, but different types of video material can produce very different difference functions, making it impossible to find a threshold that performs well on all types of video.

Zhang, Kankanhalli and Smoliar [120] choose a threshold for a sequence using the mean and standard deviation of all the frame to frame differences in the sequence being segmented, deriving the threshold as $T = \mu + \alpha\sigma$ and choosing a value of α between five and six. This adapts to the different change characteristics of different types of video, but requires two passes over the sequence. Yeo and Liu [115] instead use a sliding window to process a sequence in a single pass; the local maximum within the window is found, and if it is more than n times larger than the second largest maximum then the maximum is deemed to be a cut. Values need to be chosen for the width of the window and the multiplier n. They report good results for values of n between two and three with a window 15 frames wide. The width of the window limits the length of shot that can be detected; if two cuts occur within a 15 frame window, neither of them may be detected. Gargi, Kasturi and Strayer [33] take a similar approach, testing if the local maximum is greater than n times the average over the remainder of the window.

Vasconcelos and Lippman [109] construct a Bayesian test that incorporates prior knowledge about the likely shot lengths in the video. Boreczky and Wilcox [18] use a Hidden Markov Model, modelling the video as a state machine which moves between states of transitions, camera movements, and regular frames. In addition to histogram differences, audio and motion features are used as inputs to the system.

Clustering algorithms can also be used to identify potential cuts. Naphade et al. [65] compute two difference metrics and use them to represent each frame in a 2-dimensional space. A K-means clustering algorithm with $k = 2$ is then used to identify a cluster of potential cuts. Each potential cut is verified by checking that the frame is a local maximum for both of the difference metrics.

Llach and Salembier [55] take a technique from 2-dimensional image segmentation, namely the watershed algorithm, and apply it to segmenting the 1-dimensional difference function. The difference values are first filtered with a morphological closing operator. Seed markers are required to identify the locations of shots, so markers are placed at the local min-

imums of the curve. The watershed algorithm is then run to segment the curve into shots.

4.2.2 Gradual transitions

The detection of gradual transitions is more complex than finding cuts. There are many variations of transitions, including but not limited to fades, dissolves and wipes, and the actual inter-frame differences can be very small.

Initially researchers tried to find gradual transitions by applying more sophisticated detectors to the output from difference metrics, hypothesising that a gradual transition will reveal itself as a number of consecutive frames with differences lower than a cut but higher than regular intra-frame differences. Zhang, Kankanhalli and Smoliar [120] developed a twin threshold technique. The first threshold, T_s is set quite low, and intended to detect the start of a transition. When a frame difference exceeds this threshold, the succeeding frames are accumulated for as long as they remain above T_s. If the accumulated total for all the frames above T_s is greater than a second, higher, threshold T_b the sequence is identified as a gradual transition. The authors acknowledge problems with their approach; during some transitions the frame difference can fall below T_s, and camera motion cause false hits. Yeo and Liu [115] describe a similar algorithm to locate a 'plateau' in the difference curve.

The distance measures used in these papers are somewhat unsophisticated, and have great difficulty differentiating between motion and transitions. Hampapur, Jain and Weymouth [35] use their transition model to come up with some smarter detectors. In their model fades and dissolves are produced by applying a chromatic scaling to both shots. Assuming that the scaling is applied at a constant rate its derivative will be constant, thus fades and dissolves should reveal themselves as a series of constant difference images. Song et al. [96] use a similar production model and observe that the second derivative with respect to time should be zero. A static shot will also produce a zero second derivative, but can be distinguished as it will also have a zero first derivative. Therefore they look for sequences of frames with a small second derivative relative to the first derivative. Han and Kweon [36] note that due to camera and object motion the derivatives are rarely as well behaved as one would like.

They filter the first derivative and use a Bayesian formulation to decide if patterns in the derivative are caused by transitions or by motion.

An observable feature of fades and dissolves is that contrast is reduced during the transition. Zabih, Miller and Mai [119] observe that cuts, fades and dissolves all result in a high number of entering and exiting edges, the difference being in how the edge activity is distributed over time. In a cut all the edge activity happens between a single pair of frames while in a gradual transition the edges exit over a number of frames, then enter over several more frames. A fade out/fade in sequence will have a number of blank frames with no edges between the exiting and entering edges, a dissolve will not. Yu and Wolf [118] make a similar observation and compute two edge-based metrics for each frame after an edge detecting filter has been applied (they use the Haar wavelet transform); the edge count *EC* which is the count of edges in the frame with a strength greater than a threshold, and the edge spectrum average *ESA* which is the average strength of the edges above the threshold. Both of these features exhibit minima during fades and dissolves. The presence of a dissolve is verified by averaging the last frame of the first shot and the first frame of the second shot, and comparing this to each frame of the transition sequence; a dissolve will ideally show a V shaped curve, with minimal difference at the middle of the transition.

Rather than looking for explicit artefacts caused by dissolves, Lienhart [52] uses a machine learning approach. Low level colour and contrast based features are used as input to a neural network. Rather than marking up existing video for use as training data the training set is synthesised by applying dissolves of various lengths to a library of shots.

Utilising motion information A solution to the motion problem is to use calculated motion vectors for the sequence to decide if coherent motion is occurring rather than a transition. Zhang, Kankanhalli and Smoliar [120] use motion vectors to locate camera pans and zooms and so discount these sequences from consideration as gradual transitions. Pans are identified by finding the mean motion vector and summing the Euclidean distances of all the vectors from the mean; if the sum is below a threshold, the camera is considered to be panning. Zooms are identified by comparing the motion vectors at the extremes of the frame and checking if they are in opposite directions. Toller, Lewis and Nixon [106]

point out some problems with this approach; motion vectors, especially those found with a block matching algorithm, may not represent the true motion in low detail areas, and it cannot detect a zoom combined with another camera movement. Instead they locate the focus of expansion of the motion vectors by finding where the majority of vectors intersect. They use edge information to identify low detail areas and discount motion vectors from these areas as they may not be reliable. Fernando, Canagarajah and Bull [28] instead compute the *resultant vector* for each frame by summing all the individual motion vectors. The dot product between the resultant vectors of two frames is computed to find the angle between the vectors, which should be small if the camera is panning. Zooms are found by calculating the resultant vectors for the top and bottom halves of the screen and checking that they are pointing out of or in to the frame. Xu, Zhu and Stentiford [114] use the same concept of dominant motion and supporting pixels as they use to detect cuts to detect dissolves and 'zoom' wipes, such as the star transition in figure 4.1.

Wipes The spatially varying nature of wipes means different approaches are needed from those used for chromatic transitions. Wipes also come in many different varieties, making their detection harder. Yu and Wolf [118] describe a method for detecting wipes using difference images between consecutive frames. In two frames of a wipe, the region covered by the wipe will have a high difference compared to the rest of the frame. The first two spatial moments (the centre and variance) of this high difference region are calculated and used to identify different types of wipes; A regular wipe will have a constant variance (i.e. a constant sized high-difference region) and a constant vector between the centres in consecutive frame pairs, indicating the direction of the wipe. In 'zoom' wipes the centre is constant and the variance increases.

4.2.3 Integrated approaches

Several authors have attempted to combine detection of different transitions into an integrated scheme. Maybe the most straightforward example of this is the temporal subsampling scheme of Xiong and Lee [113]. If the video sequence is subsampled with a sufficiently large step between frames gradual transitions will fall between the samples and appear as

cuts; this part of the sequence can then be examined with successively smaller sample distances until the transition is isolated. If the transition can be isolated down to a single frame pair then it is obviously a cut, otherwise if the transition appears to occur over multiple frames they examine the edge difference to determine if a gradual transition has occurred. However movement (in particular camera movement) may also cause two frames drawn from different parts of the same shot to appear different.

Several other approaches are born out of thinking of video as a three-dimensional volume with dimensions in x, y and time. Kim et al. [44] and Ngo [68] subsample a set of pixels from each frame and use them to form a two-dimensional projection with spatial and time axes (see figure 4.9). The spatial sampling pattern can be a single horizontal or vertical line, or more complex. The idea is that transitions will appear as identifiable artefacts on the time axis; a cut will be a hard edge, a dissolve a blur between two shots, and a wipe as a non-vertical line. Kim et al. use this as the basis of a manual markup tool, while Ngo applies segmentation techniques to the 2-dimensional slice to identify the shots. Nam and Tewfik [63] and Lin et al. [54] both use wavelet transforms to get a multi-resolution decomposition of video in the time dimension, where transitions will show as high-frequency detail.

Kobla, DeMenthon and Doermann's [45] 'VideoTrails' system projects each frame into three dimensions using the 'FastMap' projection, giving a trail of points. Similar frames project to points near each other so shots appear as clusters of points joined by transition frames; they suggest that the type of transition could be determined from the shape of trajectory joining two shots.

Ford [30] applies several histogram, statistical and pixel difference metrics to a sequence and uses them as input to a fuzzy logic system which decides the likelihoods of different (or no) transitions occurring.

4.3 Post-production material

Video material encountered in a post-production environment has some unique characteristics compared to other types of video material. It is common for temporal segmentation algorithms to be evaluated on test sets of material drawn from various domains. These domains include mo-

tion pictures, broadcast television (programmes, news and commercials), and in-house material (lectures, presentations, meetings).

The majority of work in an independent post-production facility comes from commercials and music videos. The aims of these are different from the other domains listed above, which we will refer to as 'conventional' material. Commercials and music videos are designed to attract and hold the viewers attention over a very short length of time (5 to 60 seconds for a commercial, around three minutes for a music video). This need breeds a highly dynamic style of video.

Commercials and music videos also have very high production values; the importance of advertising in modern society means that a large amount of money is spent on producing a small amount of video, thus commercials can have budgets far exceeding television programmes on a cost/time basis. The high level of resources available means that video is often manipulated on a very detailed level, with each frame receiving individual attention.

Table 4.3 shows statistics about the transitions identified in the test set that will be introduced in the next section. The episode of *Frasier* is an example of more typical broadcast material; note that is has much longer shots than the commercials and music videos. Also dissolves and fades are used quite heavily in some of the music videos. *Frasier* uses fades as an editing device between the different 'acts' of the episode—the music videos do not share the same story structure.

These properties of commercials can be used to differentiate them from other material. McGee and Dimitrova [60] examine methods for identifying the advertisements within a television program; such as system could be used in the 'commercial skip' feature of a video recorder.

Post-production type material also tends to contain effects that are uncommon in regular material. Video is often subject to heavy processing, such as the colour manipulation in figure 4.3. Also, editing devices that are traditionally avoided in video editing, such as the *jump cut*, are often used. A jump cut is a cut between two shots with very similar or identical camera setups, such as shown in figure 4.4. This causes a jarring effect which is usually considered undesirable, but is used in commercials to grab attention and give a dynamic feel to the video.

	Length (m:ss:ff)	Total shots	min length	mean	Transitions cuts	dissolves	wipes	fades
arri2 (showreel)	49:17	78	:02	:16	72	0	4	3
arri3 (advert)	29:24	30	:08	1:00	27	0	2	0
arri4 (advert)	29:24	22	:13	1:09	20	1	0	0
arri5 (advert)	29:24	20	:16	1:12	7	6	0	15
advert1	29:15	31	:02	:24	30	0	0	0
advert2	29:17	41	:08	:18	40	0	0	0
advert3	9:15	6	:14	1:15	5	0	0	0
advert4	39:17	49	:06	:20	47	1	0	0
advert5	9:17	5	1:03	1:23	3	0	0	1
advert6	29:19	14	:06	2:03	13	0	0	0
music1	3:29:23	111	:06	1:22	77	25	0	15
music2	3:45:07	150	:12	1:12	159	0	0	2
music3	3:44:16	160	:08	1:10	159	0	0	2
music4	4:32:24	155	:08	1:19	114	27	0	39
music5	4:05:05	182	:10	1:09	178	2	0	6
music6	3:12:14	222	:04	:22	221	0	0	2
news	9:58:10	100	:09	6:00	88	10	1	0
Frasier	20:40:10	293	:22	4:05	279	1	0	25

Tab. 4.1: Shot and transition statistics of some typical post-production type material.

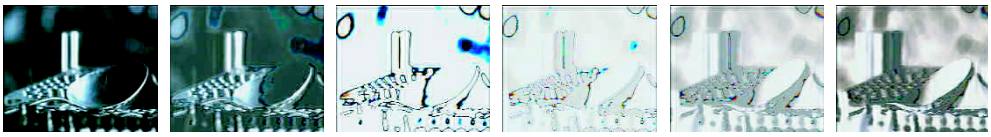


Fig. 4.3: A shot with manipulated colours.

Fig. 4.4: Two examples of a jump cut.

4.4 Evaluation of existing algorithms

As seen in the literature review, many different temporal segmentation algorithms have been and continue to be proposed; no perfect algorithm exists. The disagreements between the various published comparative reviews indicate that no obviously superior algorithm exists either. Performance of segmentation algorithms is highly dependent on the material it is used on, and as we have shown above our material is substantially removed from the 'conventional' material usually used to evaluate algorithms in the literature.

With this in mind we evaluated a selection of the better algorithms from the literature on a test set of video sequences more representative of what is encountered in a post-production environment.

4.4.1 Evaluating algorithms

There are various problems to be faced when doing a comparative evaluation of scene change algorithms:

1. Which algorithms to test? A large number of scene change algorithms have been published, and exhaustively testing all of them is infeasible.

2. Choosing algorithm parameters? Most algorithms have a number of parameters that affect performance, often with different optimal values for different types of material.

3. Choice of thresholding technique—many algorithms produce a per-frame metric that must be thresholded to locate transitions. The choice of thresholding technique is to a degree independent of the algorithm but still has an impact on performance.

4. Implementation details—implementations of published algorithms are rarely available, and the published sources often omit details of how the algorithms are implemented.

5. Testset—algorithm performance is highly dependent on the type of material being processed. The testset should be representative of the video that is to be processed by the algorithm in actual use. When using different detectors for different types of transition, should the test set contain all types of transition, or only the kind being detected?

6. Quantitatively measuring performance—the performance of each algorithm needs to be measurable in some quantifiable way that can be compared across algorithms.

Previous comparative reviews

A number of comparative studies of scene change detection algorithms have been published [17, 31, 51, 33] which have all had to address these problems.

All of the comparative studies test algorithms based on colour and greyscale histograms and DCT-based compressed domain algorithms, although each paper tests different variations. These are the most popular approaches, and are often used as benchmarks against which new algorithms are tested.

Boreczky and Rowe [17] test both global and local 64-bin greyscale histograms and use a L_1 metric; the local histograms use a second threshold for the number of regions that exceed the metric threshold. They also test motion compensated pixel difference and a compressed domain algorithm using the inner product of DCT coefficients, based on Arman's

technique. They use four testsets, totalling 233 minutes of Motion JPEG video at a resolution of 320×240 and at 30 frames per second, divided into television programmes (without commercials), news programmes, films and television commercials. Each algorithm is tested using a range of thresholds and the resulting precision/recall curves compared graphically. Gradual transitions are detected using Zhang et al.'s dual threshold technique with the global histograms and motion compensated pixel differences.

Ford [31] uses 64-bin 3-d histograms, presumably computed in *RGB* since he doesn't specify the colour space, and the χ^2 metric. He also tests chrominance histograms using the inner product as a metric, and the Kolmogorov-Smirnov test on greyscale histograms computed from the DC image of a MPEG sequence; he does not give any details on the implementation of these tests. A number of statistical tests are evaluated: the likelihood ratio, Student's t-test, Snedecor's F-test and two tests of his own devising (equations 4.6 and 4.7). Pixel differences are measured four different ways: as the L_1 distance between two images, a count of pixels with difference above a threshold, the absolute sum of differences, and the inner product of two frames. They also use Arman's inner product of DCT coefficients to work in the MPEG compressed domain. Finally they test Zabih et al.'s edge based metric. Their test set consists of approx 39,000 frames drawn from the Internet in a variety of formats and frame rates, representing a variety of material. All the tests are carried out on both colour and greyscale versions of the testset. A test set consisting of only gradual transitions was used to evaluate the performance of the different metrics in detecting gradual transitions.

Lienhart [51] tests the L_1 distance between global histograms (without specifying the colour space or size of the histogram) and Zabih's edge based algorithm. He also tests two of his own algorithms for detecting fades and dissolves: a fade detector using the standard deviation of pixel intensities, and a dissolve detector using an edge based contrast measure. The histograms are used only for detecting cuts, while the edge based algorithm detects cuts, fades and dissolves. His test set consists of 173 minutes of Motion JPEG video at 25 fps and a resolution of 360×270. Results are presented as percentages of correct and false hits for a single set of parameters.

Gargi, Kasturi and Strayer [33] undertake an in-depth analysis of his-

tograms, testing combinations of 8 colour spaces, four difference metrics, and 256 bin 1- and 2-d histograms, and 512 bin 3-d histograms. They test six MPEG based algorithms: one using the inner product of DCT coefficients, one using statistics of block types, one with statistics of motion prediction, and three applying each of histograms, pixel differences and pixel statistics to DC images. They also test two algorithms using motion compensation on uncompressed video, and one block-matching without motion compensation. The testset contained 76 minutes of video at 320×240 resolution.

Choosing thresholds

Most scene change detection algorithms require some kind of thresholding method, both for detecting cuts and gradual transitions. Global thresholds are unsuitable, as the cut and non-cut frames are rarely separable and a threshold that works well for one sequence is unlikely to produce similar results for another sequence. Zhang [120] preprocesses the entire sequence to choose a suitable threshold for that particular sequence, but this will not avoid the separability problem.

Most authors instead use local adaptive thresholds in one of two similar configurations: searching for a local maximum that is either more than a threshold t times greater than either the average over a local neighbourhood centred in the maximum, or the second highest peak in the neighbourhood. This leaves two parameters to be chosen: the value for t, and the size of the local neighbourhood.

The size of the neighbourhood needs to be large enough that it can be considered representative of the shots on either side of a cut; even a small movement will appear large if it is only considered within a very small neighbourhood. However the neighbourhood also needs to be small enough that it is not going to contain more than one shot boundary at any time; if the neighbourhood contains two cuts, the second will skew the neighbourhood so that the other may not be sufficiently large enough to be detected. Thus the neighbourhood should be no wider than twice the minimum shot length.

The value for the threshold t controls the trade-off between missing transitions and detecting false transitions. A lower value of t will detect more correct transitions, but at the expense of more false hits.

Measuring performance

Many authors express the performance of algorithms using the classic information retrieval scores of precision and recall, which are identified as

$$\text{precision} = \frac{\text{correctly detected transitions}}{\text{all detected transitions}} \tag{4.11}$$

$$\text{recall} = \frac{\text{correctly detected transitions}}{\text{total transitions}} \tag{4.12}$$

Precision and recall have been a staple of retrieval evaluation for many years, and are presented either as a pair of numbers for an algorithm with fixed parameters, or as a curve showing the trade-off between precision and recall as the algorithm's parameters are adjusted; the best compromise performance is usually deemed to be on the knee of this curve. However precision and recall have been attacked as being a disingenuous means of evaluating performance on retrieval tasks; Forsyth [32] points out that the perceived performance depends upon the task in hand. If it is vitally important that every occurrence of an event be identified then a larger proportion of false hits may be an acceptable price to pay. However if the dataset is vast and the user is satisfied to find *most* of the occurrences of an event without being swamped in false hits then it is the algorithm's performance at the other end of the curve that is of interest. But precision and recall give us a measurable quantity that we can use to compare algorithms.

The trade off of precision against recall generally depends upon the parameters of the algorithm, in particular the threshold used in thresholding algorithms. By testing a large range of thresholds we can produce a Receiver Operating Characteristic (ROC) curve of precision against recall, as shown in figure 4.6 with precision increasing as we increase the threshold. If we can produce such a curve then we can compare algorithms a number of ways. One is to fix the precision to a given value (such as 95%) and compare the recall scores that each algorithm can achieve for this level of precision; this will either involve iteratively searching for a threshold that produces the desired precision, or interpolating from neighbouring precision values. If recall is more important for our given application then we can fix the recall value instead and compare the precision values.

Ford [31] proposes two additional metrics; the area above the ROC curve, which should be minimised, and a total probability of error based on summing the precision and recall errors. Boreczky [17] compares precision-recall curves visually. Gargi et al. [33] propose using a local separability measure to show how well local thresholding will perform in detecting cuts. They calculate

$$M_{\text{sep}} = \frac{\overline{C} - \overline{NC}}{\mu}, \tag{4.13}$$

where $\overline{NC}$ is the average value of the metric for a non-cut frame over the sequence, $\overline{C}$ is the average metric value for a cut frame, and μ is the mean value of the metric for all the frames within a local window. This value is computed for a local window (in their case 12 frames) centred on each cut in the sequence, and the mean value used to indicate the local separability of the metric for the entire sequence—a higher value indicates better performance with local thresholding.

Another important criteria that affects reported performance is how correct detection of transitions is classified. Unless the testset is generated synthetically it must be marked up manually with the locations and types of transitions. This can lead to ambiguity, as Gargi et al. note. In particular it is difficult to decide exactly where the start and end of a fade or dissolve are. We must also set the criteria for which an algorithm is judged to have correctly detected a transition. Is a detected yet misclassified transition counted as a hit, miss, or ignored? Does the algorithm have to detect the boundaries of a gradual transition accurately, or do the estimated bounds merely have to overlap with the actual transition. Usually authors accept overlapping regions as a hit, and ignore multiple detections of the same transition. Gargi et al. accept detected events if they fall within three frames of the actual event—in the case of cuts we consider this to be a false economy, particularly given the short shot durations we are dealing with.

4.4.2 Experimental setup

For our own comparison we must also address the problems described above. We now describe the conditions of our test.

The test set

To evaluate the suitability of previously published algorithms for use in segmenting the types of material found in a post-production environment we experimented with a number of algorithms from the literature. A test set was drawn together with examples of television commercials, music videos, and an agency showreel. To represent 'conventional' material a news bulletin and an episode of the sitcom *Frasier* were also included.

Table 4.3 shows the details of the test set. Most of the material was sourced from digital cable broadcasts which were recorded onto VHS tape and then digitised at a resolution of 352×288. The others were obtained directly from Unique-ID's demonstration *Cakes* installation. All clips were stored in M-JPEG format at a resolution of 176×144. Each clip was examined manually to identify the location, type and duration of each transition, which was recorded and used as the ground truth data in evaluating each algorithm.

As some members of the testset are very short we also divide the testset into larger groups for evaluation purposes. The nine commercials are grouped together, as are the six music videos. These two groups together with the 'arri2' showreel form a group of 'post-production' type material. We also group the news bulletin and *Frasier* together as a 'reference' group.

The algorithms

As seen in the review of the literature above, temporal segmentation is an active field, and many different approaches have been proposed. It is impossible to test them all so in choosing a subset we have been guided by published comparative reviews, the perceived popularity of algorithms from their repeated use as benchmarks against which new algorithms are tested, and by algorithms which have a particularly interesting or novel approach. Some algorithms detect only one type of transition, whereas others detect several—however we will examine each type of transition individually.

It should be noted that there are many disagreements between the various comparative reviews in the literature over what algorithm performs 'best'. Performance is affected by the choice of test data and implementation details, and no one algorithm has a clear lead over the

others.

Colour histograms The straight-forward colour histogram has a number of parameters that can be tuned and impact performance. In choosing configurations of colour histograms to evaluate we turn to the comparative reviews in the literature.

Gargi, Kasturi and Strayer [33] test histograms in a number of colour spaces and with several different frame difference metrics. They found that the Munsell colour space gives the best results, but has a significant computational cost. The $L^*a^*b^*$ space was found to be the best compromise between performance and cost, and that using any colour space was better than using greyscale data alone. Conversely, Ford [31] finds the use of colour to provide no advantage over greyscale, and often perform worse. Ford also finds that computing several local histograms over different blocks of the image outperforms a single global histogram.

Of the different frame difference metrics, Gargi finds the histogram intersection to be the best, and that the popular χ^2 metric is significantly worse. But again Ford disagrees, finding it to perform quite well, particularly with global colour histograms. The best metric in Ford's study is the Kolmogorov-Smirnov test (equation 4.10).

As there is no clear 'best' histogram configuration we will evaluate several, testing colour versus greyscale, local versus global, different colour spaces, and different histogram difference metrics.

We test four colour spaces: RGB, Greyscale, YUV and $L^*u^*v^*$. The greyscale histogram uses the six most significant bits of the Y channel from the YUV colourspace to quantise the space to 64 levels. The three channel histograms use the two most significant bits from each channel for a total of 64 bins. Global and local histograms were tested; the local histograms are calculated by dividing the frame into 16 subblocks and using the median value of the metric when calculated on all the subblocks. Four metrics were tested: the L_1 distance, the $\chi 2$ metric, the histogram intersection and the Kolmogorov-Smirnov test.

Pixel statistics We test a number of the greylevel pixel statistics which are reported to work well by Ford [31]: the F-test (equation 4.4), and Ford's own tests λ_1 and λ_2 (equations 4.6 and 4.7).

For detecting dissolves we test Lienhart's algorithm which uses the standard deviation of the pixel intensities [51].

Edge-based metrics We test Zabih's edge based metric, but without the motion compensation step as this adds a significant computational burden and there is no motion data available in our M-JPEG format sequences.

Others We also examined the visual rhythm technique suggested by Kim [44] and Ngo [68].

Evaluation criteria

As mentioned above, there are several ways of measuring the performance of scene change detection algorithms, using precision/recall values, or measuring the local separability of metric based techniques.

Local thresholding is done by thresholding each frame pair against t times the average value of the local window. The choice of window size presents another trade-off; we tested two window sizes in order to observe the effect of the window size. As our material has many short shots, the choice of window size may have a large impact on algorithm performance. The window sizes we test are two frames either side and six frames either side.

As mentioned earlier, by adjusting the threshold used with a metric-based algorithm we can produce a range of precision/recall values. This is because the cuts and non-cuts are rarely separable—a more typical distribution is shown in figure 4.5. In this case there is no threshold that will completely separate the cuts from the rest of the frames. By varying the threshold between the minimum cut values and the maximum non-cut value (as shown in the figure) we can generate a range of precision/recall scores. The lowest threshold value with give us perfect recall at the expense of precision. The higher value with give better precision (as more non-cuts are to the left of the threshold), but at the cost of recall. In the case shown in the figure perfect precision is possible, but if there is a non-cut value above all the cuts then perfect precision will be impossible.

From such a curve it is possible to determine the precision at a given level of recall, or vice versa. In our case we will find the precision at 90%

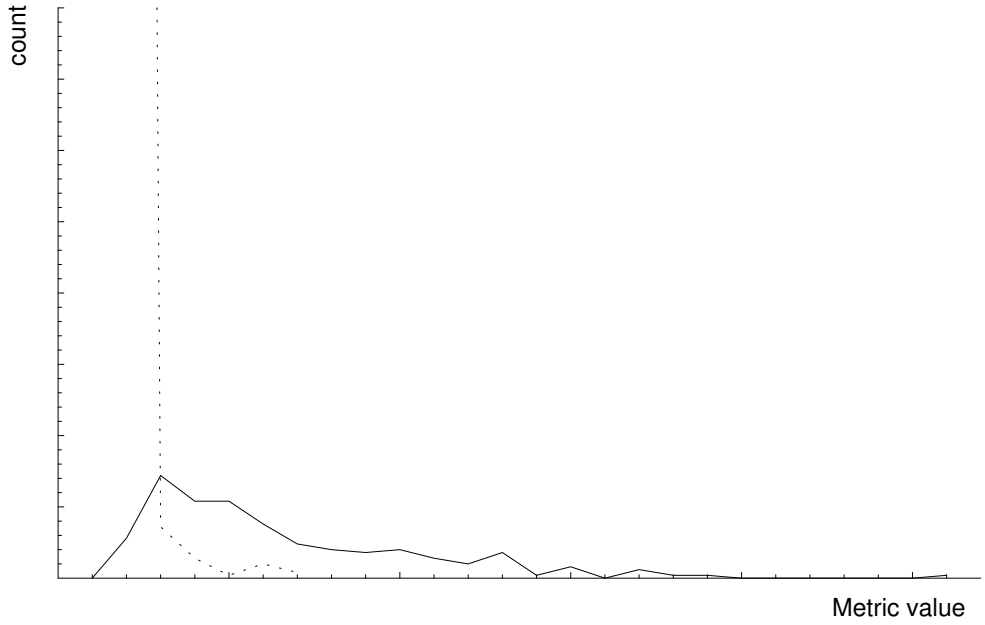


Fig. 4.5: Histogram of the ratios of a metric for cut and non-cut frames over their local neighbourhoods, from the whole of the 'adverts' part of the testset. If the two distributions don't overlap then it is possible to attain perfect precision and recall.

recall and the recall at 90% precision. We calculate these by taking 200 evenly spaced samples in the overlapping region of the distributions and interpolating between the nearest.

Although the recall is guaranteed to decrease as the threshold is raised, the precision may vary depending upon the distribution. Thus it is possible that the precision never reaches 90%, in which case we record the precision as zero at 90% recall.

Each sequence in the testset was tested individually, which means that a different thresholds are used for each sequence. In a real application we are not able to determine the optimal thresholds in such a manner, so we also measured the performances for the aggregated groups of commercials, music videos, post-production material and the reference set.

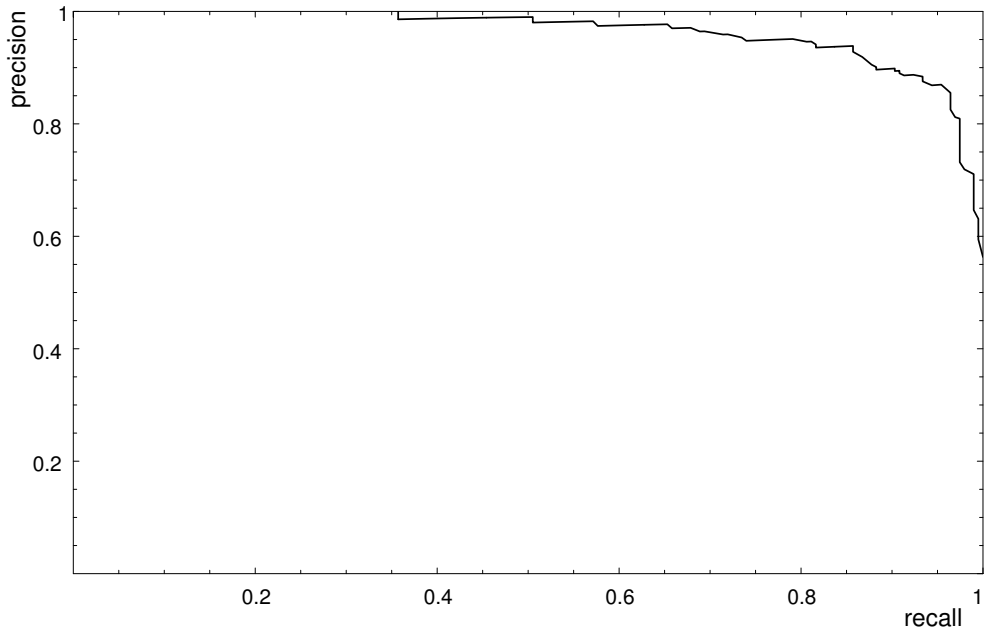


Fig. 4.6: A precision/recall graph generated from figure 4.5.

4.4.3 Results

The tables in appendix B show the results of the tested algorithms on the five groupings of the testset.

Pixel intensity methods

Despite its simplicity, the sum of pixel differences performs fairly well, although not as well as the colour histograms. On the reference material about 95% precision and recall can be achieved. Performance is lower on the post-production type material, as can be seen in the table.

Histograms

Our choices of colour space, metric and local versus global histograms gives 32 permutations. The full results for each histogram run on the testset are presented in appendix B, but we summarise them here.

On the 'conventional' material, perfect precision and/or recall is possible, and all the metrics show good results. This tallies well with the reported results of other authors, who find colour histogram metrics to

perform strongly. Surprisingly the better results come from using the narrower local neighbourhood, which counters the intuition that a wider neighbourhood will give a better context in which to identify the spike in the metric caused by the cut.

But what about the post-production material? Performance on the commercials was generally poor, but the music videos were better. Interestingly the commercials and music videos favoured different metrics. The *RGB* colour space performed best on the music videos, while *YUV* and greyscale histograms were best for the commercials. Using the narrower window was best for the commercials, while the music videos didn't show any strong advantage in either size. Overall using local histograms produced better results, and the best overall was using local *RGB* histograms and the χ^2 metric.

A closer look at one of the commercials reveals some of the problems with colour histogram techniques. Figure 4.7 shows the output of the χ^2 metric with local *RGB* histograms for the 'advert6' sequence. The sequence doesn't involve any particularly complex edits and the only expected difficult part is the product shot at the end. Looking at the graph we can see that as well as the spikes corresponding to the cuts, the metric produces strong responses during three shots, which are shown in figure 4.8. During the first of these (04:06 to 05:17) the man is large on the screen and his movement causes large shifts in the histograms of several sub-regions. These shifts are amplified by the strong contrast of the shot as the man's dark shirt moves to reveal the bright background. In the other two shots (10:14 to 11:19 and 17:05 to 18:08) people move from light into shadow, producing a similar shift in the histograms.

Visual rhythm

Figure 4.9 shows some of the 2-dimensional slices obtained by using the visual rhythm technique [44, 68]—each column of pixels in the image is a sample from a single frame. In the extract from *Frasier* the cuts are obvious; the visual content of the neighbouring shots is suitably different that a clear edge is visible, and the cuts are far enough away from each other that they are clearly differentiable. The same is not true of the other two examples. The short shot lengths mean that the edges are very close to each other and the dynamic nature of the shots means

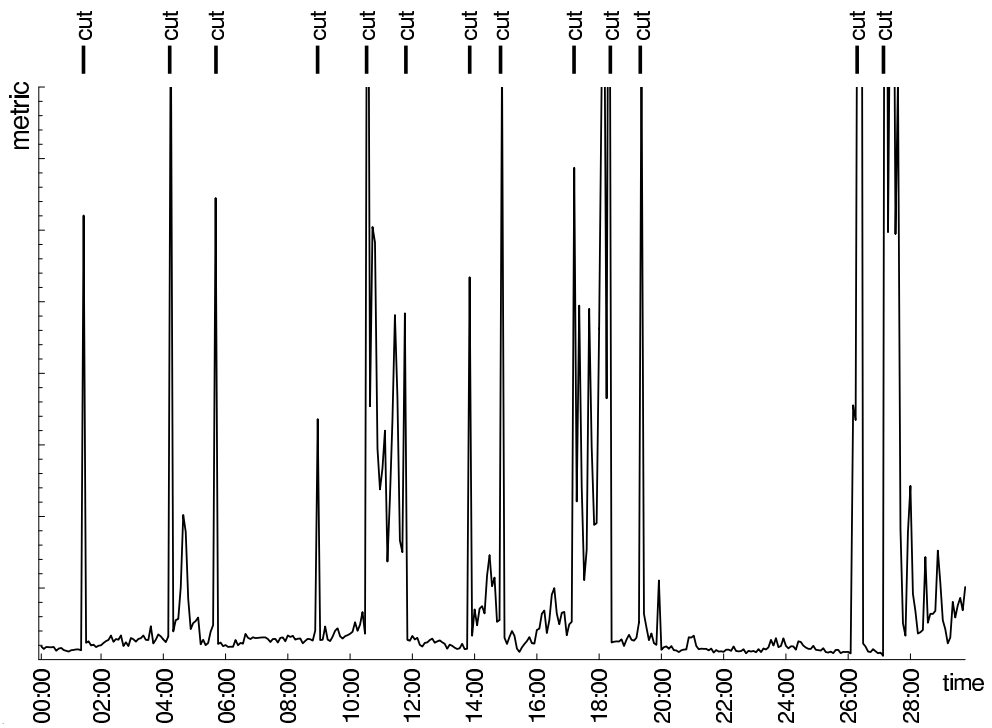


Fig. 4.7: Output of the local $RGB\chi^2$ histogram metric of the 'advert6' sequence.

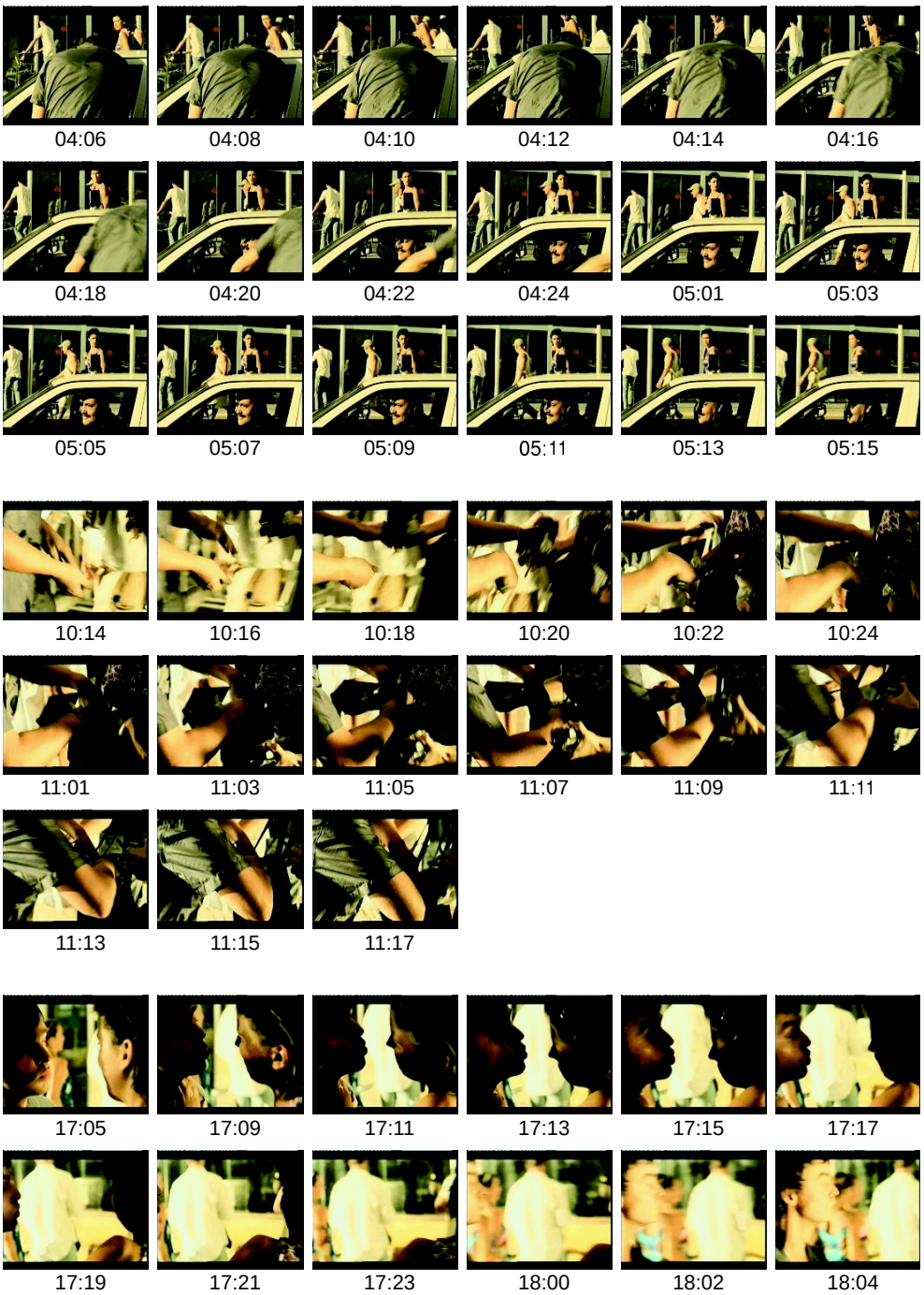
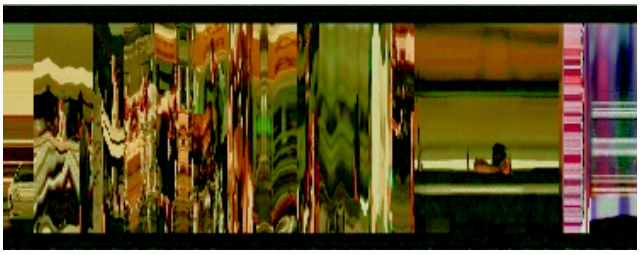


Fig. 4.8: Three shots from the 'advert6' sequence which produce high colour histogram responses.

Advert 6



Music video 6

Frasier

Fig. 4.9: Examples of the visual rhythm technique applied to the test set (all shown to the same scale).

that there is significant noise within the shots, making it difficult to determine the edges which indicate the shot boundaries. It is quite clear that 2-dimensional segmentation techniques will have little success in determining the shots within such sections.

Inner products

Both of the inner product based metrics performed very poorly. Figure 4.10 shows the inner products of the pixel intensities for the 'advert6' sequence; as can be seen the cuts do have an impact upon the metric, but it is difficult to distinguish and is not always a spike.

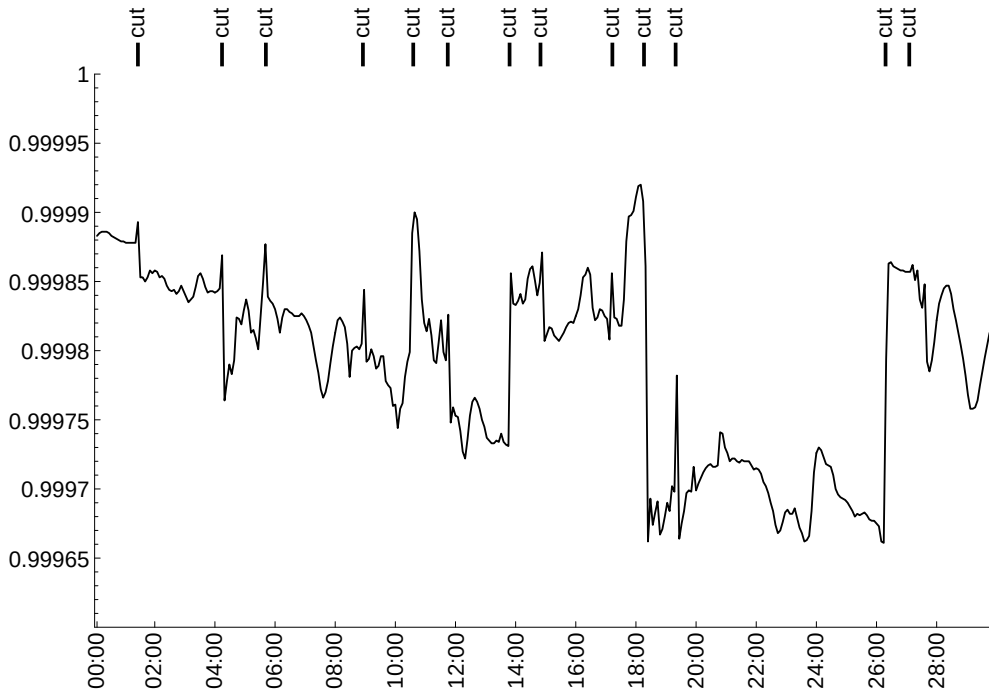


Fig. 4.10: The inner product of pixel intensities for the 'advert6' sequence.

4.4.4 Fade detection

We also examined two fade detection techniques: Zhang's twin threshold technique and Lienhart's technique using the standard deviation of pixel intensities.

Zhang's twin threshold technique is not specifically a fade detector, but rather designed to detect any gradual transition. We use the local RGB χ^2 histogram metric with the detector, as this is the one that performed best detecting cuts. Figure 4.11 shows the output of the metric for the 'arri4' sequence, which contains one fade out to black and one dissolve. Recall that the twin threshold technique uses two thresholds; a threshold for detecting cuts, T_b, and a lower threshold for detecting gradual transitions, T_s. To be detected, each frame of a gradual transition should exceed T_s and the cumulative value of the metric during the transition exceed T_b. From the graph we can see that this approach is going to have problems. During the fade the metric is very high, certainly in excess of the value of T_b required to detect the cuts in the sequence. The dissolve however produces a very low response. The levels of T_s and T_b required to detect the dissolve would be so low that several other parts of the sequence will also be detected as transitions. Figure 4.12 shows the first 30 seconds of the 'music4' sequence, this time using the global RGB χ^2 metric, which contains many fades and dissolves. It is quite clear that there is too much noise present to distinguish the different types of transition using such a metric. Many fades are of very short duration, making if difficult to differentiate them from cuts.

Lienhart's fade detector detects only fades, identifying them as linear changes in the standard deviation of pixel intensities. Figure 4.13 shows the standard deviation of the pixel intensities for the first 30 seconds of the 'music4' sequence. It can be seen on the graph that there are near linear descents during fade outs. However some of the fades are of very short duration, and there are also many other events which cause a drop in the standard deviation.

When we implemented and tested the Lienhart's detection algorithm it detected most of the fade outs in our testset; in order to simplify the results we only look for fade outs. Table 4.2 shows the number of correctly detected fade outs for each of the sequences in the testset that contained fades. We deem a fade to have been correctly identified if the estimated

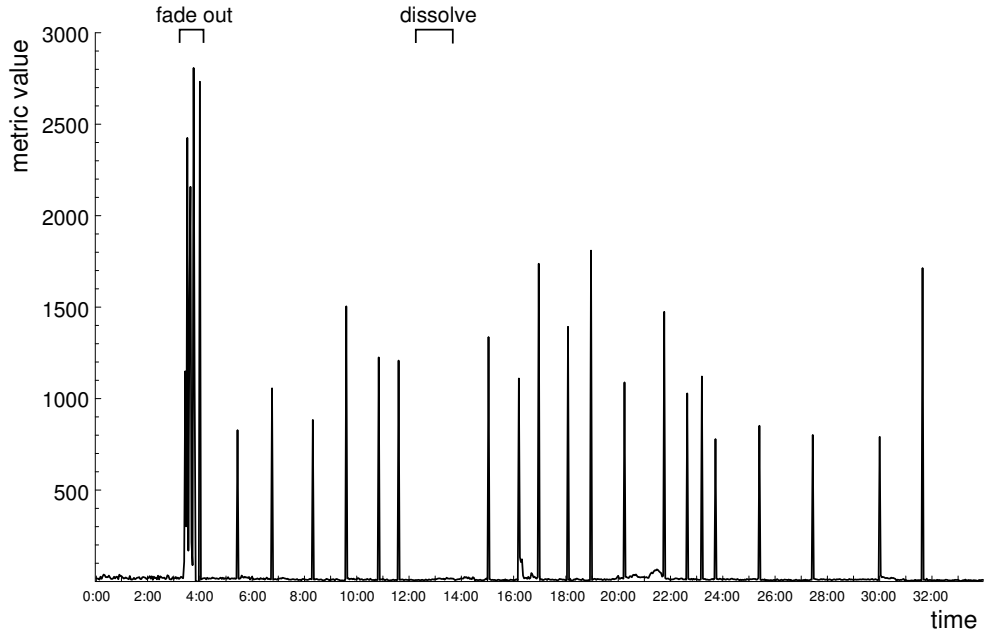


Fig. 4.11: Output of the local RGB χ^2 metric for the 'arri4' sequence with gradual transitions.

fade boundaries overlap with the manually determined fade boundaries. The algorithm often detected the start of the fade to be some distance ahead of the start as identified by a human observer. It is possible that this is because the human eye is less sensitive to changes in intensity at the start of the fade, where the image is brighter.

The algorithm only managed to successfully detect fades to black, thus the low scores for the 'arri5' and 'music1' sequences which contain many fades to white. The only false positive encountered was at the end of the 'arri2' clip, where a figure retreats through a closing door from a black room. While this is not strictly a fade, it is a gradual transition to a black screen.

4.5 A novel algorithm

The performance of the tested algorithms on post-production type material prompted us to explore whether we could design a temporal segmentation algorithm specifically tailored for this type of material.

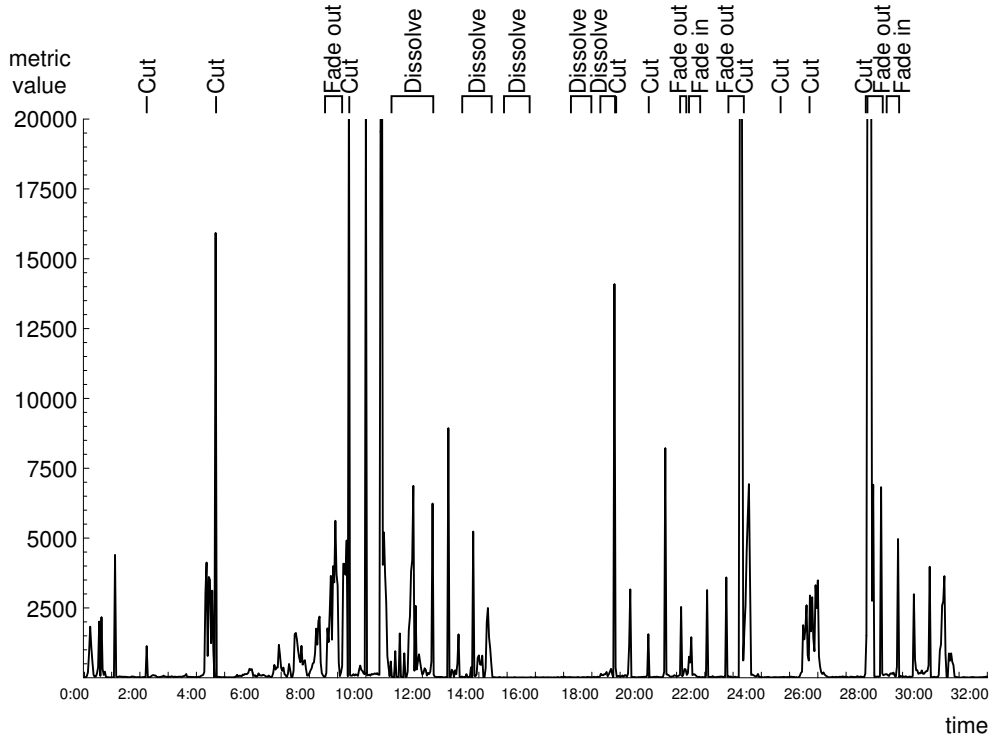


Fig. 4.12: Output of the global RGB χ^2 metric for the first 30 seconds of the 'music4' sequence.

arri2	0/1	(1 false)
arri3	1/1	
arri4	1/1	
arri5	1/8	
music1	0/8	
music2	3/4	
music3	1/1	
music4	18/28	
music5	0/2	
music6	0/1	
frasier	12/13	

Tab. 4.2: Number of correctly detected fade outs using Lienhart's standard deviation of pixel intensities technique.

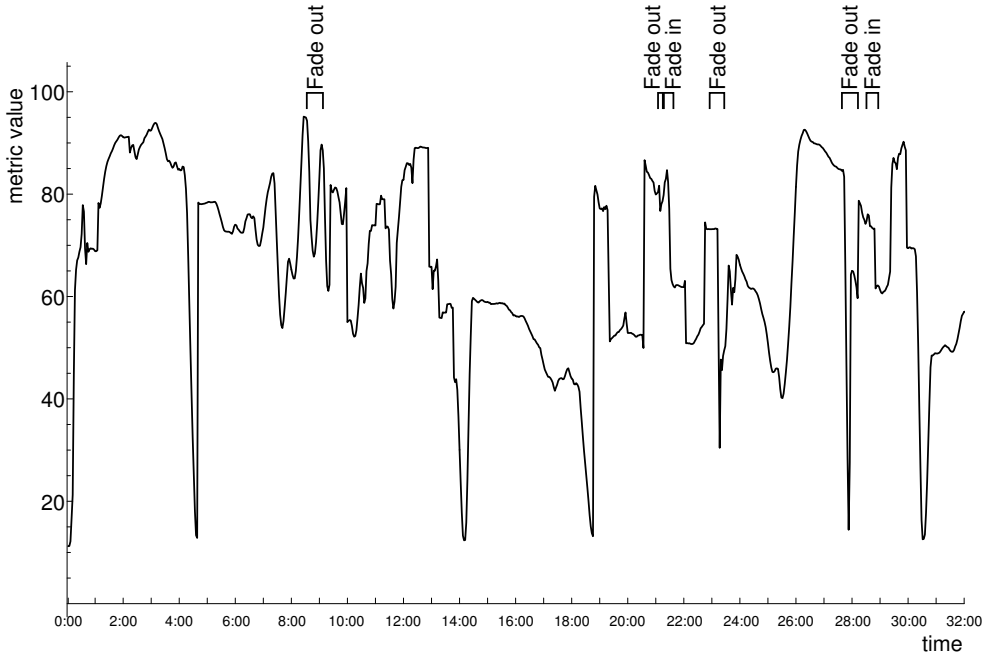


Fig. 4.13: Standard deviation of pixel intensities for the 'music4' sequence.

A primary criteria of our new algorithm is to be robust to the highly dynamic characteristics of video that cause problems for the existing algorithms. We need to be able to handle very short shots, large scale motion, and illumination changes. Other desirable features are for it to be fast, operate in a single pass over a stream without needing to see the entire sequence first, and be applicable to a wide variety of types of video without needing retuning.

4.5.1 Rationale

Our algorithm is based on the concept of analysing small regions of a frame in the context of its spatial and temporal neighbourhood, and making a local observation of the video's behaviour. A region's spatial neighbourhood consists of the regions surrounding it in the frame, and its temporal neighbourhood is the corresponding regions in the same position in the preceding and succeeding frames. If a region's behaviour is consistent with its neighbourhood we can assume that this is representative of the true activity occurring on a larger scale. For example if a

region exhibits fading behaviour, and this is consistent with its temporal and spatial neighbourhood (the same region in the preceding and succeeding frames is also fading), then the locally observed fading behaviour is probably due to a fade transition rather than a transient illumination change. If a region exhibits a large change that might lead us to think a cut has occurred, but this behaviour is not reflected in the spatially surrounding regions then the change was more likely caused by localised motion rather than a frame-wide cut. If a region exhibits a large change that might be consistent with a cut but this behaviour also occurs in its temporal neighbours, then the region is probably in a dynamic shot rather than a cut.

Once all the local observations have been made, they too can be analysed in the context of their neighbours to try and identify whether they are representative of the video's global behaviour or are merely undergoing a local change caused by something happening within the shot.

4.5.2 Operating on JPEG DCT coefficients

As we are operating within the framework of the *Cakes* system, where video is stored in a Motion-JPEG format, we have designed the algorithm to operate directly on JPEG compressed frames. JPEG compresses an image in 8×8 pixel macroblocks, which provide us with a natural local region to operate on.

We build our algorithm on the basis of comparing JPEG macroblocks. We can make a number of comparisons by utilising the DCT coefficients directly, saving the computational cost of the inverse DCT operation required to decompress the image. From the DCT coefficients we can extract the following information:

1. The mean values of the Y, U and V channels over the block (from the DC coefficient).

2. A simple detail or energy measure, calculated by summing all the AC coefficients from the block.

3. A normalised cross-correlation between two blocks, calculated by applying the normalised cross correlation operator to the AC coefficients of each block.

The normalised cross-correlation operator is usually used for template matching in the spatial domain. It is calculated as

$$c = \frac{\sum_i (a_i - \overline{a})(b_i - \overline{b})}{\sqrt{\sum_i (a_i - \overline{a})^2 \sum_i (b_i - \overline{b})^2}} \tag{4.14}$$

where $\overline{a}$ and $\overline{b}$ are the mean values of each block. To avoid having to apply the inverse DCT transform, we instead apply the operation to the AC DCT coefficients rather than to the actual pixel values. Although this is not mathematically equivalent to doing the cross-correlation in the spatial domain, it does give very similar results. On a test using over 18 million block pairs from our test set, the mean difference between the spatial cross-correlation and the DCT cross-correlation was 0.0128 with a standard deviation of 0.05. On the basis of this experiment we deemed the DCT cross-correlation to be satisfactory for our purposes.

DCT coefficients have been used before to compare macroblocks, but using absolute differences or the inner product of the AC coefficients. Our testing in the previous section showed the inner product to be a poor metric. The spatial cross correlation is an accepted and well used operator for comparing the similarity of two regions, and the closeness of our DCT cross-correlation to the spatial version makes it preferable to using the differences or inner product of the DCT coefficients.

The detail measure is a simple calculation of the amount of contrast in a block. We consider blocks with an energy measure of less than 100 to be uniform. This value was determined by trial and error to be sufficient to account for the noise present in uniform areas. The cross-correlation operator becomes unstable with very low detail frames, and is swamped by the noise that the JPEG encoding introduces (it will also cause a division by zero on a perfectly uniform block).

Our implementation uses the freely available JPEG library from the Independent JPEG group [7]. The library takes care of subsampling, quantization and huffman encoding used in JPEG compression and allows us access to the DCT coefficients.

4.5.3 Detecting cuts

In designing our cut detector we need to focus on the types of cuts that caused difficulty for the shot change algorithms examined in the previous

section. We can identify the different types of cuts we want to detect:

1. Cuts between two different and relatively static shots. These are simple to locate, and can be reliably found with techniques such as colour histograms.

2. Cuts between relatively static, but very similar shots, such as the jump cuts shown in figure 4.4. These are difficult to differentiate from object motion as much of the frame may change very little or not at all.

3. Cuts before or after highly dynamic shots. These are difficult to identify because all the frame pairs prior to the cut show substantial changes. For local thresholding techniques this makes the local neighbourhood around the cut very noisy so the metric response to the cut is difficult to identify.

We can also identify the kind of events that cause false positives:

1. Large object motion. Large movement of an object that occupies a significant part of the screen causes significant changes for most difference metrics.

2. Illumination changes. A change in scene lighting causes a shift in colours across the whole frame, which causes a large response from colour histogram based metrics.

3. Post-production effects. Many effects are added synthetically in post-production that can cause falsely detected cuts. An example is colour manipulation such as in figure 4.3, which may cause histogram based metrics to falsely detect a cut.

In most of these cases there are cues we can examine to differentiate these cases. Illumination and colour changes can be caught by also looking at the correlation of image detail across two frames. If a normalised correlation is used the effects of any colour or illumination changes should be minimised, and we can expect to see a correlation between the two frames despite a large change in brightness or colour.

A jump cut appears as a very sudden movement; it will occur over a single frame pair, unlike object movement which can be expected to last longer than the 1/25 of a second of a single frame.

A cut between dynamic shots will show poor correlation and changes in colour and illumination, but so will the frame pairs within the dynamic shot. However the nature of the changes in the cut will be different from that caused by the dynamism in the scene. The correlation may be poorer or the colour change greater than that caused by the dynamic motion in the shot. By analysing the behaviour of the dynamic shot we may be able to identify out of character changes that result from a cut. If a cut joins a dynamic shot to a relatively still shot then we will also observe a marked decrease in frame changes in the still shot.

DCT behaviour during cuts

We design our cut detector to try and take advantage of these cues using the statistics we can extract from the JPEG DCT coefficients. First we examine the behaviour of the DCT coefficients during some representative cuts and potential false hits, and from these design an algorithm that can differentiate between them.

Figure 4.14 shows our DCT-based statistics during a fairly simple cut. Neither of the shots has highly dynamic motion, and although the two shots are of the same location, there is a large enough change in the camera positions that the cut is obvious. This cut is reliably detected by most of the algorithms tested in the previous section.

The close-up shows the same subregion of nine 8×8 macroblocks for each frame, and the DCT statistics for these blocks. The correlations are with the corresponding block in the previous frame, and the other values are the difference between the value for the current frame and the value for the previous frame. The U and V differences are common over a 2×2 block region as the means are calculated from the DC coefficients of the macroblocks in the U and V channels, which are 16×16 pixels due to JPEG's subsampling.

The intra-shot correlations are mixed; there are a few good correlations (> 0.4), more indeterminate values, and a couple of poor correlations (< 0). There is movement in the scene, which affects the correlations, although it is interesting to note that it does not cause all the

blocks to correlate poorly; for example the top centre block shows reasonable correlations despite being in a motion area. The cut frame however shows more poor correlations, although a couple of the blocks have values around 0.3. The blocks that were showing good intra-shot correlations have poor values for the cut though.

The differences between the Y, U and V means show a certain level of noise during the shots and then larger differences on the cut. Again this is not universally true for all blocks, but does hold for the majority. The colour channels in particular can show large differences compared to their intra-shot behaviour.

Figure 4.15 shows a cut following a highly dynamic shot where an object that occupies most of the screen is making large movement. The movement of the nut also causes large changes in illumination as it turns to reflect the light. As expected the correlations within the dynamic shot are poor, and the changes in the Y mean and energy on the cut are difficult to differentiate from those within the shot. However there is a large change in the U and V channels on the cut, although the change in colour may not be visible to the naked eye. The first two frames of the following shot show a marked improvement in the correlations, indicating that these two frames are more likely to be part of the same shot that those before.

Figure 4.16 shows six frames from the 'arri5' sequence during which the scene illumination changes. Illumination changes like this cause a shift in the colour histograms of the frame, which causes colour histogram based shot detection algorithms to detect a cut. Looking at the DCT statistics there are significant changes in the intensity and colour channels at the two points where the illumination changes. However the correlations are strong throughout the illumination changes. Some of the blocks near the centre get completely washed out and so will show poor correlations, but the majority of the blocks within the frame retain strong correlations.

Algorithm

Based on the analysis of the behaviour of the DCT-based statistics in the previous section, we designed a novel cut detection algorithm. Following our earlier design statement of making decisions on a local basis, we aim

Fig. 4.14: DCT statistics behaviour during a cut in the 'advert6' sequence. The cross-correlations and differences between the Y, U, V and energy means of each frame and the previous frame are shown for nine DCT macro-blocks.

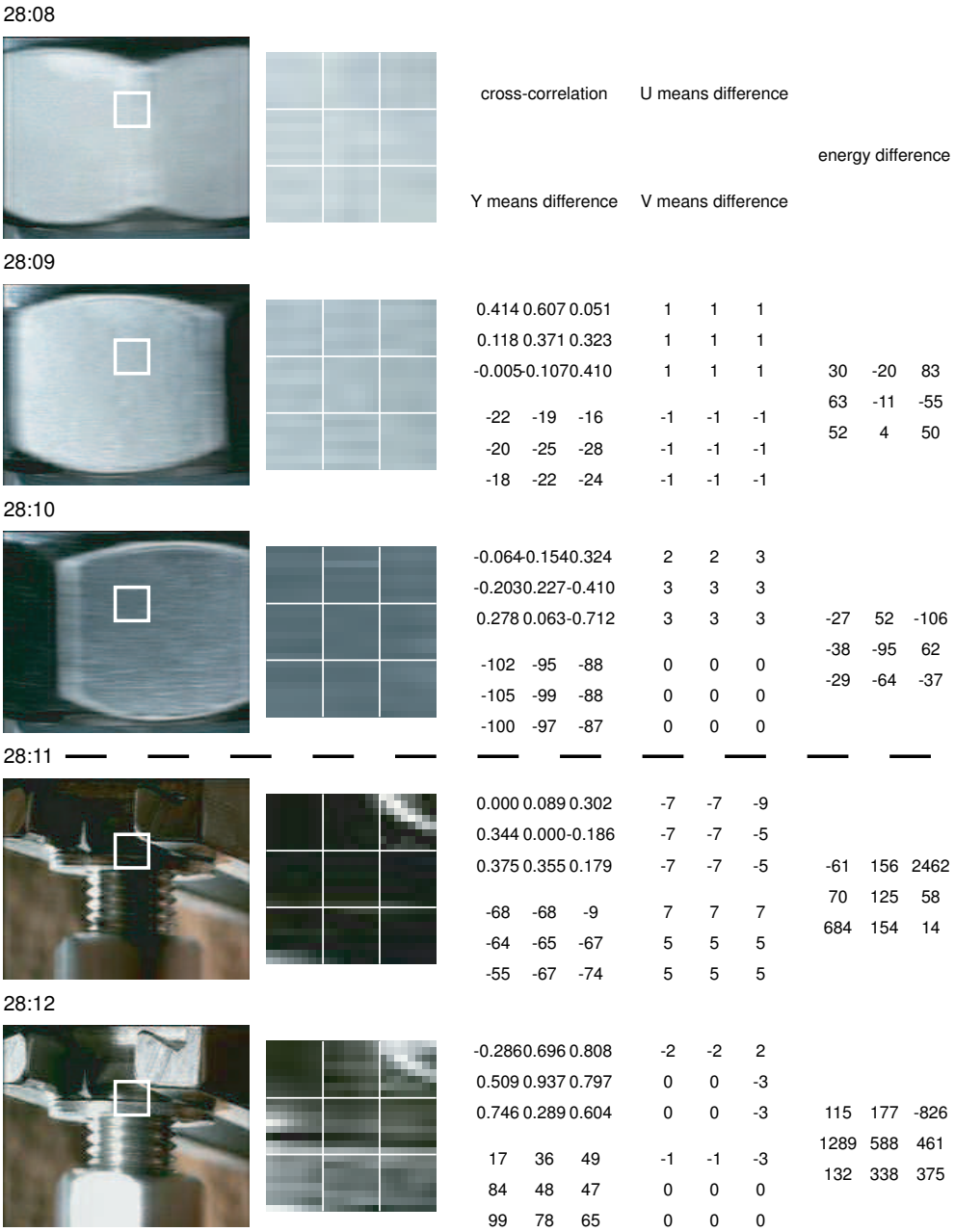


Fig. 4.15: DCT statistics behaviour during a cut from a highly dynamic shot in the 'arri2' sequence.

Fig. 4.16: DCT statistics for frames during the 'arri5' sequence. The change in illumination causes some shot change detection algorithms to falsely detect a cut.

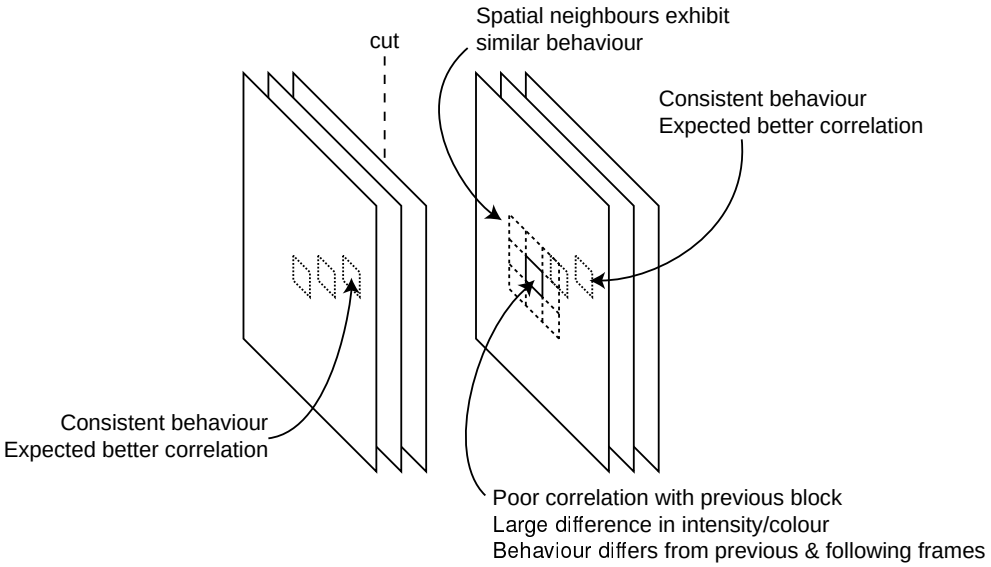


Fig. 4.17: Local region behaviour during a cut.

to make a per block decision as to whether there has been a cut or not.

Figure 4.17 shows a summary of the type of block behaviour we are looking for at a cut. A large change in any of the statistics that is out of character with the behaviour of that statistic in the immediate past may be a sign of a cut, although if there is a strong correlation between the blocks then it is more likely that the changes are a result of illumination or colour manipulation rather than a cut. We also look for improved consistent behaviour after a suspected cut, and this can be used to verify that a cut actually occurred.

We represent the past behaviour of a statistic for a particular block by keeping a trailing window of the value for the last n frames. The behaviour of a statistic over this window can be characterised by its mean and standard deviation. If a value falls outside of a certain multiple of the standard deviation from the mean, then we can say that the value is out of character with its past behaviour. When we detect a cut we must reset the window to start after the cut, so that we only have frames from a single shot within the window. We set n, the size of the window, to be 5 frames.

The need to determine what multiple of standard deviations constitutes uncharacteristic behaviour introduces a threshold into the system, along with the various problems associated with thresholds. However

the threshold is sufficiently decoupled from the actual image data that we should avoid the main pitfalls of threshold selection.

Labelling blocks For each block i in the current frame f under consideration we compute the following statistics:

$Y_{f,i}, U_{f,i}, V_{f,i}$**:** The Y, U and V means over the block, taken from the DC coefficients.

$E_{f,i}$**:** A simple detail measure found by summing the AC coefficients of the Y channel block. When this value is below 100 we consider the block to be uniform.

$C_{f,i}$ The normalised cross-correlation between the AC coefficients of block (f, i) with those of block $(f-1, i)$.

In addition to the statistics for the current block we have the means, $\overline{Y_i}, \overline{U_i}, \overline{V_i}, \overline{E_i}, \overline{C_i}$, and the standard deivations, $\sigma_{Y_I}, \sigma_{U_i}, \sigma_{V_i}, \sigma_{E_i}, \sigma_{C_i}$, over the trailing window for each statistic.

There are three particular block behaviours that we are interested in. We want to examine each block in the frame and possibly label it as one of three types:

1. Blocks with a good correlation which is consistent with good correlations in the trailing window.

2. Blocks whose correlation is significantly worse than they have been in the trailing window.

3. Blocks which show a significantly improved correlation compared to the previous frame pair.

It is also possible that the block type is inconclusive, in which case we label it as type unknown. Instances of the first type of block suggests that the current frame pair are from the same shot. Blocks of the second type signal that a cut may have taken place. Blocks of the third type suggest that a cut may have occurred between the previous pair of frames.

Experimentation with a selection of cuts and potential false positives led to the following rules for labelling blocks as one of the three types. These rules are applied to each block pair where neither of the blocks are uniform:

1. If the block has a good correlation then we disregard any changes in illumination, colour or energy. We require that the correlation be in line with the correlations of previous frames in the trailing window, as a large decrease in correlation would make the block type 2, and a large increase type 3.

   - $C_{f,i} >= 0.3$, and $\sigma_{C_i} < 0.25$, and $C_{f,i}$ falls within one standard deviation of $\overline{C_i}$.

2. If the block is not of type 1 then we look for a large decrease in correlation or a large change in intensity, colour or energy that is outwith the bounds of the block's past behaviour.

   - One block is uniform and the other is not, and the difference between $E_{f-1,i}$ and $E_{f,i}$ exceeds a threshold,
   - or the sum of absolute differences for the U and V means is greater than a threshold and falls outside of two standard deviations of its past behaviour,
   - or $C_{f,i}$ is more than 0.5 less than $\overline{C_i}$.

3. The block may show a significant improvement in correlation compared with the previous frame pair.

   - $C_{f,i} > 0.5$ and $C_{f-1,i} < 0.3$,
   - or $C_{f,i}$ is at least 0.3 greater than $C_{f-1,i}$.

If either of the blocks is uniform and the other is not, then we mark is as type 2. If both blocks are uniform we look for a change in the intensity or colour; if there is a change, the block is marked as type 2. However if both the blocks are uniform and the same colour then we discard the block. This is to cope with shots with very dark or otherwise uniform backgrounds (we will examine this later).

We also disregard all blocks along the borders of the image. This is because material that has been stored or transferred in an analogue format at some stage in its lifetime can pick up noise and artefacts near the edges of the frame. Figure 4.18 shows the corners of two frames on either side of a cut in the 'arri5' sequence. There are visible bands of dark pixels along the edges of the frame that are consistent across the

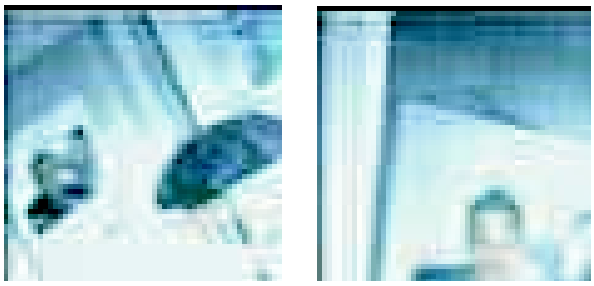


Fig. 4.18: Detail of the corner of two frames from the 'arri5' sequence. Although the frames are from either side of a cut, there is a consistent banding along the edge of the frame.

whole sequence and cause strong correlations between blocks, even when the contents has changed. Although the 'arri5' sequence was sourced digitally from Smoke & Mirrors's *Cakes* system it has passed through an analogue process at some point. For our sequences stored at 176×144 resolution this means discarding 76 (or 20%) of the 396 blocks.

Figure 4.20 shows the block labelling in the frames around a straightforward cut in the 'advert6' sequence. The first frame shows a few changed blocks caused by movement within the frame, but these are comparatively few compared to the well correlated blocks (types 1 and 3). In the cut frame there are a large number of changed blocks. The well correlated blocks along the top and bottom of the image are caused by the black bands at the top and bottom of the image, which produce a strong edge that is common across all the frames. This 'letterbox' format is commonly used for broadcast in the UK to make the image suitable for both conventional 4:3 aspect ratio receivers and the newer 16:9 widescreen format that is being phased in.

Figure 4.21 shows the same cut as figure 4.15, with a highly dynamic shot cutting to a less dynamic one. Using the trailing window means that many of the blocks in the first frame are not labelled, despite having changed from the previous frame (see figure 4.15). The cut frame still shows many type 2 blocks however, as the nature of the changes is different from that during the previous dynamic shot (the change in the U and V values as seen in figure 4.15). Also the second frame of the following shot shows many improving type 3 blocks (despite the illumination change caused by the nut's rotation).

Fig. 4.19: Two frames from either side of a cut in the 'arri2' sequence. Much of the image in both frames is uniformly black.

Once the blocks in a frame are labelled there will most likely be a mixture of all three types of blocks, as well as undetermined ones. We must now aggregate these labelled blocks to decide on the behaviour of the frame itself—is the frame a cut or not. An obvious criteria for a cut frame is that there be more type 2 blocks than type 1 and type 3 blocks. However a highly dynamic shot may cause there to be very few well correlated blocks (as in figure 4.21). Another problem is with very dark shots or graphics where much of the frame is black and detected as being uniform. For example, figure 4.19 shows two frames from either side of a cut in the 'arri2' sequence where many of the blocks are uniform black in both frames. If the uniform blocks are labelled as being type 1 then they will outnumber the type 2 blocks, making the cut difficult to identify. This is the reason we discard all the similarly coloured uniform/uniform block pairs.

Instead of using a fixed rule to determine whether a cut has occurred based on the number of each type of block, we use the same technique as we did for the blocks themselves—compare the current frame with the behaviour of the previous frames. The count of each type of block over the trailing window is kept, and again the mean and standard deviation calculated to represent the past behaviour.

Cuts are identified in two stages; first a potential cut is located by its count of type 2 blocks, then it is verified using the count of type 3 blocks in the following frame. A frame pair is marked as a potential cut if all the following conditions are satisfied:

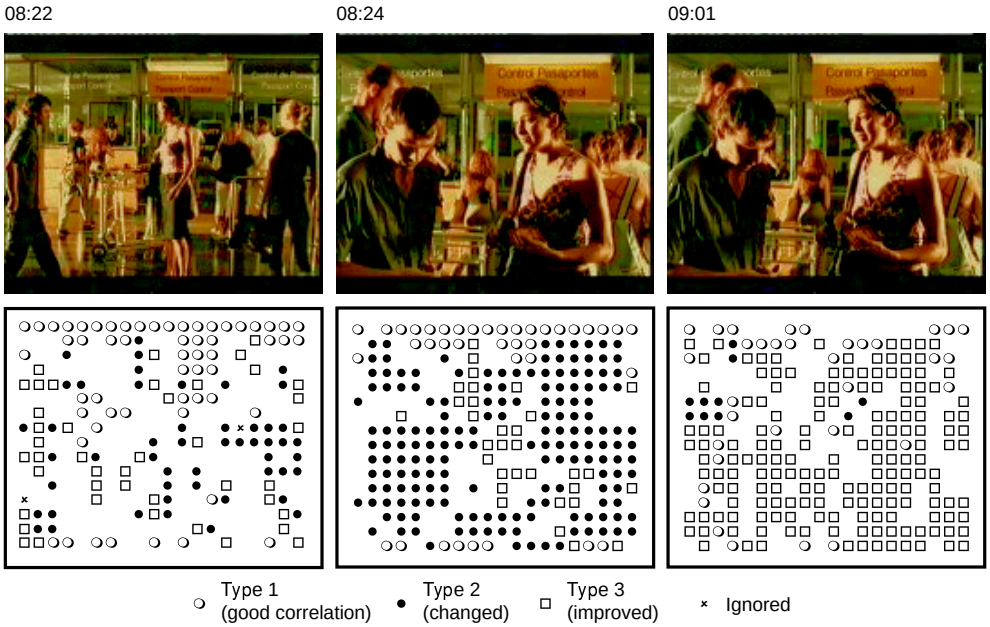


Fig. 4.20: Labelled blocks for a simple cut from the 'advert6' sequence.

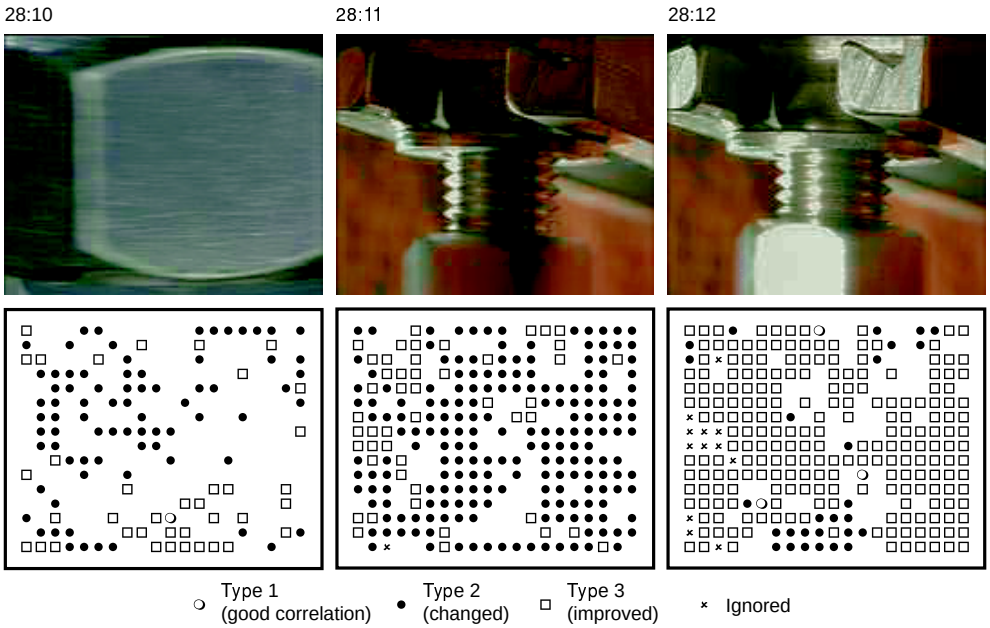


Fig. 4.21: Labelled blocks for a simple cut from the 'arri2' sequence.

1. The number of type 2 blocks exceeds the total number of type 1 and type 3 blocks.

2. The number of type 2 blocks is greater than one standard deviation from the mean of its behaviour over the trailing window.

3. The number of type 2 blocks is at least 25% higher than for the previous frame.

The third rule handles cases where the standard deviation of the count of type 2 blocks is small and so a small increase in movement could appear as a cut under rule 2. This is primarily required for relatively static but dark scenes where many of the blocks are discarded as uniform/uniform pairs. A small amount of movement will cause a small increase in the number of type 2 blocks but there are insufficient type 1 or 3 blocks to discard a cut under rule 1.

Once a potential cut is identified it is verified by examining the count of type 3 blocks in the following frame, which must meet the following conditions:

1. There are more type 3 than type 1 blocks.

2. The number of type 3 blocks exceeds the average number of type 3 blocks in the previous two frames.

3. The average of the number of type 3 blocks in this frame and the number of type 2 blocks in the previous frame is at least 25% of the total number of blocks in the image (excluding the discarded border and uniform/uniform blocks).

There are a number of hardwired values in these rules, and the use of such constants is a source of inflexibility in any algorithm. The values shown are those that we have determined to perform well on post-production type material. They are deliberately lenient towards the changes in illumination and colour that occur regularly in this type of material, and to cope with highly dynamic shots.

The pseudo code for the cut detection algorithm is presented in appendix A.

4.5.4 Detecting fades

Types of fades that cause problems for existing fade detectors are very short fades and non-linear fades. We wish to design our detector to detect both these kinds of fades, and fades to any colour.

We focus here on fade outs—fade ins are the same except reversed, so any technique for detecting fade outs can be easily modified to locate fade ins (the simplest method being to run the sequence backwards).

DCT behaviour of fades

During a fade the image gradually changes to a uniform frame of a single colour. Therefore we expect to see a falling level of detail and changing intensity and colours over the sequence. Correlations will also be affected by the vanishing detail.

Figure 4.22 shows the behaviour of the DCT statistics during a conventional fade to black. The intensity falls throughout the fade, but not in a strictly linear manner. Blocks which started out brighter obviously fall more with each frame, and there is some variation in the rate of intensity change. The colours also change in a fairly consistent manner, although the direction depends upon the initial colour (as the U and V values are signed and range from -128 to 127). The energy also falls with each frame as expected. The correlations are initially good until the detail starts to disappear, where they get significantly worse.

Figure 4.23 shows a trickier fade from the 'arri5' sequence. This sequence contains many illumination changes which caused problems for cut detectors; in this case the fade initially looks like an illumination change but continues until the whole frame is washed out. The fade is also very short and non-linear in nature (there is a large jump in the final frame). The behaviour of the DCT statistics is similar, except this time the illumination increases until each block is fully saturated. Note that in this sequence the artefacts along the borders of the frame are quite clear.

We can characterise the behaviour of the DCT statistics during a fade out as follows (figure 4.24):

1. Energy levels will fall consistently through the fade.

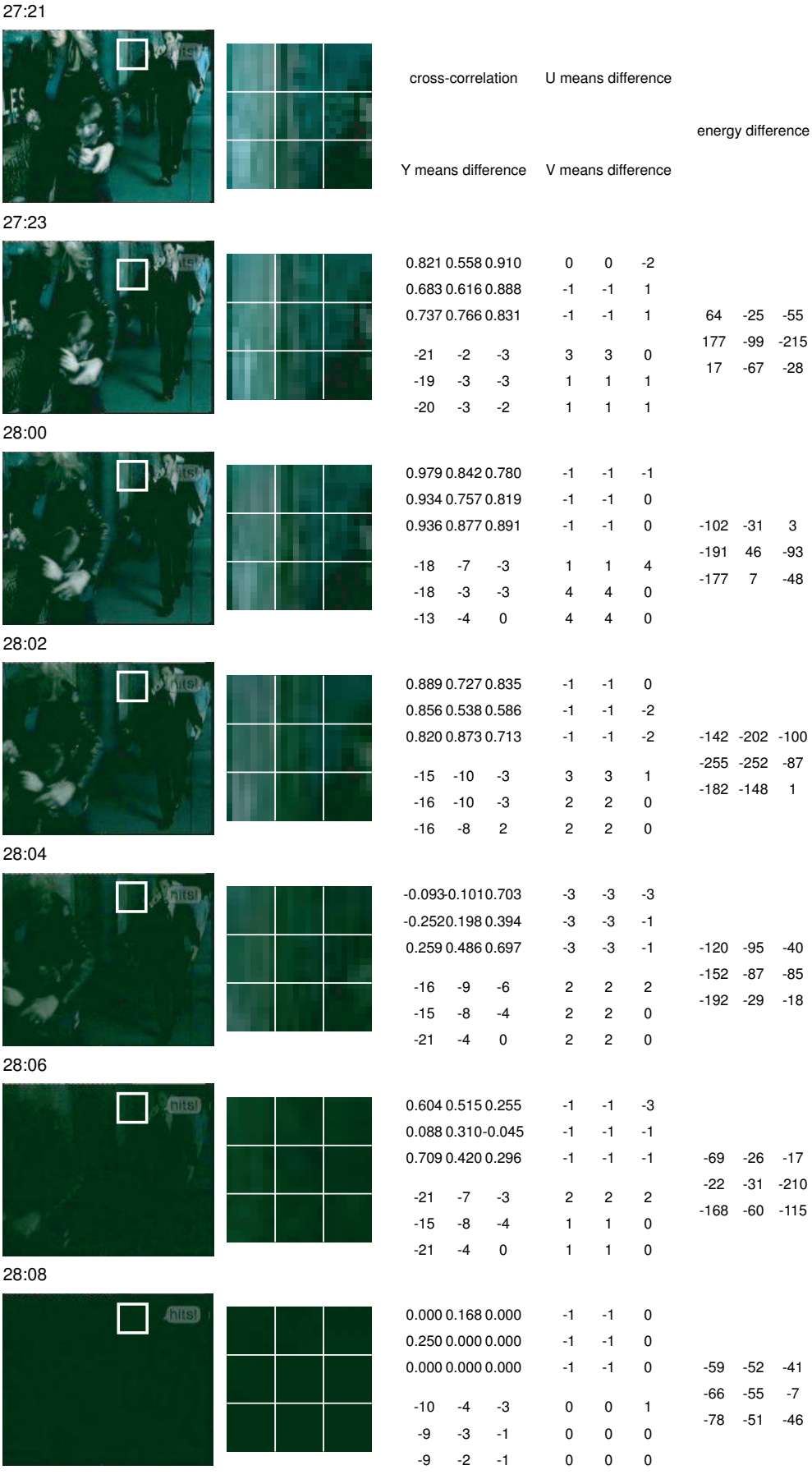


Fig. 4.22: Behaviour of the DCT statistics during a fade to black.

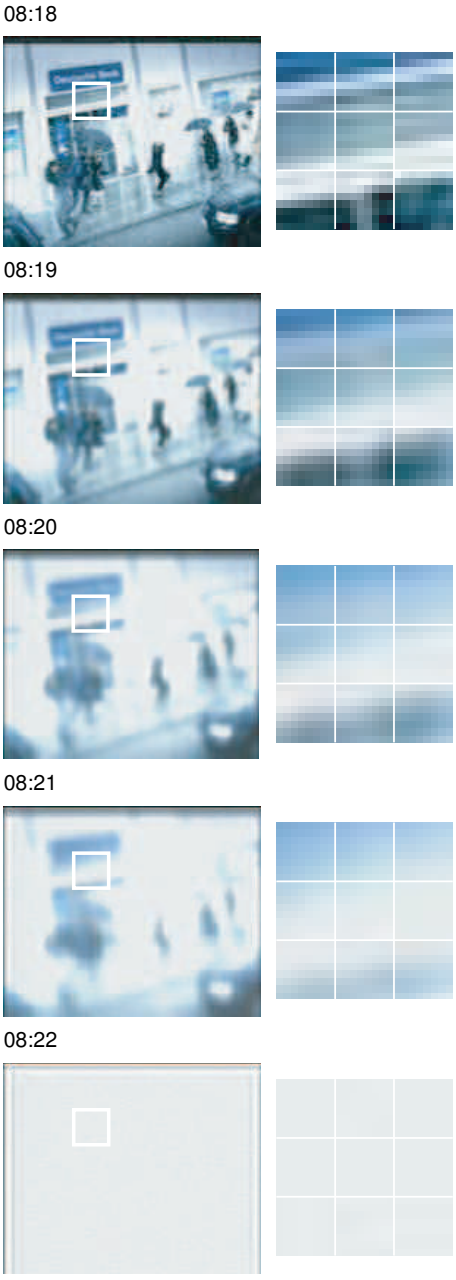


Fig. 4.23: Behaviour of the DCT statistics during a fade to white.

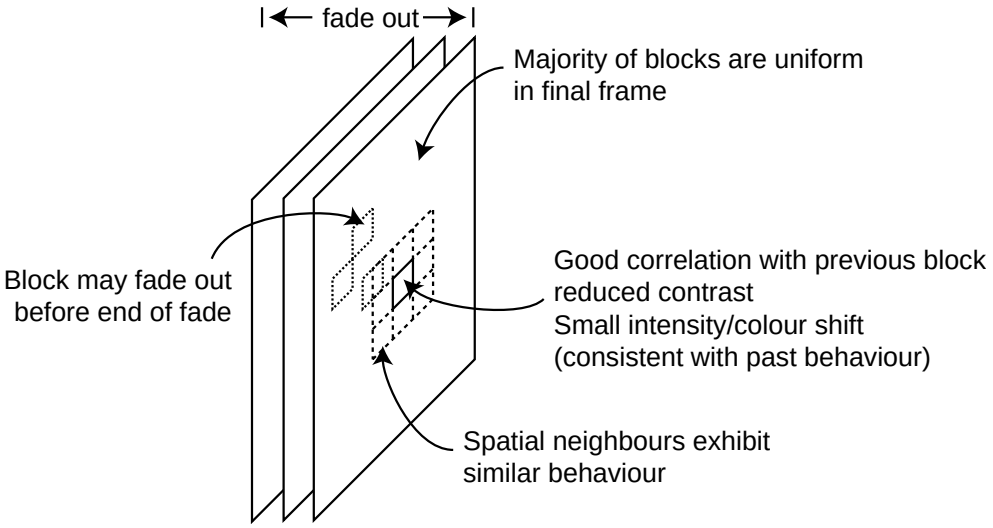


Fig. 4.24: Local region behaviour during a fade out.

2. Intensity and colour values will move consistently towards their final values (which may not necessarily be black or white).

3. Correlations may start off high, but will break down as detail disappears.

It is possible for a block to reach the end of its fading behaviour before the end of the fade. Indeed a block may not show fading behaviour at all if it starts off as uniform with the same colour as the final frame (like the shadow areas of the sequence in figure 4.22). A frame that starts off uniform but a different colour will still show changing intensity and colour but not detail, as it has none to begin with.

Algorithm

Our fade detection algorithm looks for fade-like behaviour in a block and tracks this behaviour from frame to frame until it decides that a fade has taken place. There are many other events that can cause a block to exhibit fade-like behaviour such as motion or illumination changes. To counter this we add an additional constraint by examining each block's spatial neighbours and checking that they exhibit the same behaviour.

For each block we store data on whether the block looks like it is fading, which fade-like changes it is undergoing, and how long it has

been fading for. We will ignore the effect of fading on the U and V channels, as we find that the intensity channel is sufficient.

We initially examine each block i in isolation for three fade like behaviours: a decrease in E_i, a decrease in Y_i or an increase in Y_i. We ignore any changes in E_i if $E_I < 100$ to avoid the noise present in low detail blocks. We then examine the eight neighbours of block i and if any one of them is undergoing one of the changes that i is showing, we mark i as fading and store the types of change.

If the block was already marked as fading we check if it is undergoing the same types of changes as it was in the previous frame. If so then we increment the count of how many frames the block has been fading for, otherwise we mark the block as non-fading. If a fading block reaches a uniform state we keep incrementing the counter as long as the block stays static.

When we reach a frame with at least 80% uniform blocks we make a decision as to whether we are at the end of a fade. Two conditions must be met: More than half of the blocks must have been fading for more than two frames, and the ratio of fading blocks in the current frame to non-uniform blocks in the previous frame must be greater than 0.75. The starting frame is found by taking the median age of the fading blocks. Only requiring 80% uniformity means the algorithm can tolerate a certain amount of noise and also overlays such as channel logos which do not fade along with the rest of the image, such as in figure 4.30.

4.6 Evaluation of new algorithm

The new DCT-based algorithm was evaluated on the same testset as the other algorithms in section 4.4. It must be noted that this is the same video collection that was used in the design of the algorithm, so performance against another testset may be poorer.

4.6.1 Cuts

Unlike the metric-based shot change detection algorithms, it is difficult to produce a range of precision/recall values by merely varying a single parameter. Instead we can produce a single performance measure using the various parameter values as they were described in the previous

Fig. 4.25: An effect from the 'music1' sequence which is falsely detected as a cut.

section.

We compare our new algorithm with the best shot change detection algorithm from section 4.4, which is the local RGB histogram with the χ^2 metric and local thresholding over a window two frames wide to either side. Since the colour histogram technique offers a trade off between precision and recall we are able to select a threshold that gives the same level of recall as our new algorithm. To achieve a recall of 0.978 over the entire testset the threshold is set to 2.5354; if the metric is greater than 2.5354 times the average over the local window, we detect a cut.

Table 4.3 shows the results for both the new algorithm and colour histogram. Overall the new algorithm achieves better precision at similar levels of recall than the colour histogram algorithm.

The 'music1' and 'music4' sequences produce quite a high proportion of false hits. Many of the false cuts in the 'music1' sequence occur during fades or dissolves. There are also a few special effects that trigger false cuts, such as the sequence shown in figure 4.25. The colour histogram technique achieves a higher precision on the 'music1' sequence, but with a lower recall when using a threshold of 2.5354. If we adjust the threshold to bring the recall up to the same level the precision of the colour histogram falls to 0.709, narrowing the gap somewhat. In the 'music4' sequence the false hits also predominantly occur during fades. The 'news' sequence contains a sports report with footage from a football match which causes a number of false hits, particularly during close-up shots of the players while they are running. The combination of object movement, camera movement and motion blur cause problems for the new algorithm and the histogram technique alike.

	New algorithm				Colour histogram	
	hits/total	false	precision	recall	precision	recall
arri2	71/74	6	0.922	0.959	0.563	0.973
arri3	27/27	5	0.844	1.000	0.563	1.000
arri4	20/20	1	0.952	1.000	0.833	1.000
arri5	7/7	6	0.538	1.000	0.200	1.000
advert1	22/30	0	1.000	0.733	1.000	0.967
advert2	40/40	1	0.976	1.000	0.952	1.000
advert3	5/5	0	1.000	1.000	1.000	1.000
advert4	47/47	6	0.887	1.000	0.940	1.000
advert5	4/5	0	1.000	0.800	0.800	0.800
advert6	12/13	8	0.600	0.923	0.706	0.923
music1	82/83	39	0.678	0.988	0.794	0.928
music2	143/147	5	0.966	0.973	0.894	0.973
music3	157/159	5	0.969	0.987	0.888	0.994
music4	110/114	28	0.797	0.965	0.691	0.982
music5	178/179	14	0.927	0.994	0.978	0.994
music6	209/220	5	0.977	0.950	0.887	0.932
news	88/88	25	0.779	1.000	0.715	1.000
Frasier	279/279	12	0.959	1.000	0.663	1.000
All adverts	184/194	27	0.872	0.948	0.749	0.985
All music	879/902	96	0.902	0.975	0.864	0.968
All post	1134/1170	129	0.898	0.969	0.816	0.971
Reference	367/367	37	0.908	1.000	0.675	1.000
All	1501/1537	166	0.900	0.977	0.776	0.978

Tab. 4.3: Performance of the new DCT-based cut detection algorithm on the testset.

Fig. 4.26: A very short fade to white from the 'music4' sequence.

4.6.2 Fades

Table 4.4 shows the results of the new DCT-based fade detector against Lienhart's standard deviation of pixel intensities fade detector for fade outs only. Overall the new algorithm detects substantially more of the fades but at the cost of many more false positives. The majority of the newly detected fades are the fade-to-whites that are entirely missed by the other algorithm. Also detected are more of the short duration fades. The marking of fades can become slightly ambiguous with the very short fades, particularly with the 12.5 frames per second sequences. At this frame rate a short fade can last as little as two or three frames, such as the fade from the 'music4' sequence in figure 4.26.

The two falsely detected fades from the 'music1' sequence are a little ambiguous—they are both light effects where the shot starts with most of the frame washed out and the camera then moves to point straight into the light, further saturating the image and pushing it over the 80% uniform threshold that triggers the fade detector (figure 4.27). It could be argued that this is actually a fade, however in the marking up of the video we have chosen not to interpret it as such.

Another source of falsely detected fades are cuts to a mostly uniform frame, such as the falsely detected fade from the 'music3' sequence in figure 4.28. If the behaviour of the preceding frames has shown any fade-like behaviour then the nearly uniform fade can trigger the fade detector. Since the fade detector sets no age limit on fading blocks, the original fade-like behaviour may have been many frames before the nearly uniform frame.

Another source of mostly uniform frames that cause problems for the fade detector are cartoon style shots or logo shots, such as shown in figure 4.29. These have enough uniform blocks to trigger the fade detector.

	New algorithm		Lienhart's algorithm	
	hits/total	false	hits/total	false
arri2	1/1	6	0/1	1
arri3	1/1	0	1/1	0
arri4	1/1	0	1/1	0
arri5	8/8	1	1/8	0
advert1	0/0	0	0/0	0
advert2	0/0	0	0/0	0
advert3	0/0	0	0/0	0
advert4	0/0	0	0/0	0
advert5	0/0	0	0/0	0
advert6	0/0	0	0/0	0
music1	8/8	2	0/8	0
music2	4/4	0	3/4	0
music3	1/1	3	1/1	0
music4	25/28	0	18/28	0
music5	1/2	0	0/2	0
music6	1/1	0	0/1	0
news	0/0	1	0/0	0
Frasier	11/13	0	12/13	0
All adverts	10/10	1	3/10	0
All music	40/44	5	22/44	0
All post	51/55	12	25/55	0
Reference	11/13	1	12/13	0
All	62/68	13	37/68	0

Tab. 4.4: Performance of the DCT-based fade detector against Lienhart's fade detector. Only fade outs are recorded.

Fig. 4.27: A falsely detected fade from the 'music1' sequence.

Fig. 4.28: A cut to a mostly uniform frame in the 'music3' sequence, which is falsely detected as a fade.

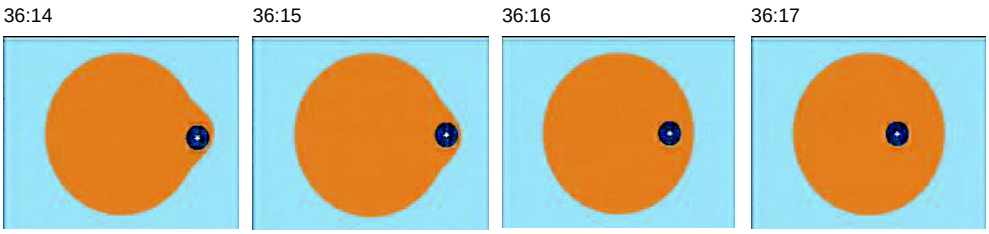


Fig. 4.29: A sequence of cartoon style frames from the 'arri2' sequence that triggers a falsely detected fade.

Fig. 4.30: A fade from the 'music4' sequence that is correctly detected despite the overlaid graphics.

There is a tradeoff in the amount of uniform blocks required to detect the final frame of a fade; if we increase the percentage of uniform blocks required then we can avoid falsely detected fades like that of figure 4.29, however the more lenient setting we use allows for the detection of fades in the presence of overlaid graphics such as the fade from the 'music4' sequence shown in figure 4.30, which is correctly detected.

4.7 Conclusions

In this chapter we have explored the application of shot change detection algorithms to the type of material encountered in a post-production environment, namely commercials and music videos. This is an area that has received very little attention in the published research on the temporal segmentation of video.

We have pointed out that this material is significantly different in nature to the 'conventional' material such as regular television programmes and news bulletins that are usually used in evaluating shot change detection algorithms. The material is highly dynamic with very rapid transi-

tions, large amounts of motion and is heavily manipulated, making the location and type of many transitions ambiguous.

Having evaluated a number of published shot change detection algorithms on samples of this material we found their performance disappointing, and instead designed a novel temporal segmentation algorithm tailored to handle this type of material. The resulting algorithm was designed to process the online proxy images stored by the *Cakes* system, and exploits the features of the M-JPEG format in order to reduce the required computation and avoid having to uncompress the video.

Our algorithm outperforms previous algorithms when detecting cuts and fades on post-production type material, and performs comparably when processing 'conventional' material. It is also fast, running at near four times real time on a 400MHz PC, due to its use of the DCT data.

However there are still many situations where no algorithm can successfully partition the video as a deeper semantic understanding of the content is required to identify the different elements and transitions. This task is difficult even for the experienced human observer.

5. Spatial segmentation

In this chapter we explore techniques for intra-shot segmentation—we assume that each video sequence is an uninterrupted sequence of frames with no transitions. Within the shot we want to identify the primary objects; objects that a human observer is interested in. We are interested in determining the location, shape and motion of these foreground objects.

There has been a lot of recent research in this area, much related to the new MPEG-4 format for video. Among its many other features, MPEG-4 allows a video sequence to be encoded in layers with arbitrarily shaped regions. For example, a person in a scene can be encoded in a separate layer from the background, allowing more bandwidth to be dedicated to the moving person than to the relatively static background. The MPEG-4 standard does not dictate a method for extracting these layers, only for encoding them. Therefore researchers and implementers have been free to explore and compare many different segmentation techniques.

Without a high level semantic understanding of the contents of an image it is very difficult to separate the foreground from the background of a static image—indeed the differentiation between what is foreground and background is somewhat subjective. We shall therefore make two vital assumptions about what constitutes a foreground object: that objects are rigid or semi-rigid, and that they move independently of the background. The rigidity constraint means that we can expect movement to be coherent across the extent of an object. These assumptions mean we can recast the problem of identifying foreground objects as one of identifying regions of coherent movement within a shot.

Segmenting moving objects draws from two areas of computer vision research—spatial segmentation and motion estimation. This chapter is organised as follows: first we will look at the different methods for describing and estimating motion within a video sequence. Then we look at static segmentation techniques which can be applied to a single image. We then look at techniques which draw from both these fields for segmenting moving objects from a video sequence.

5.1 Motion models and estimation

The apparent motion in a sequence of images is useful information for a number of purposes. In video coding and compression motion can be used to exploit temporal redundancy in the sequence, drastically reducing the amount of information that needs to be stored or transmitted. Registering two images can be useful, for example building a panorama from a number of overlapping images [88]. This requires that the relation of the images to each other be ascertained so that the overlapping regions can be aligned. This is similar to a major problem in computer vision, that of *stereo matching*. In stereo vision a scene is photographed from two (or more) different viewpoints, and it is necessary to identify the corresponding points in the images.

5.1.1 Motion models

There are several ways of modelling the motion between two images. The simplest is *optical flow*, where each pixel (x, y) has a unique motion vector (u, v) describing its translation from the first image to the next. Ideally, the intensity of each pixel in the second image, I_1, can be described from the first image, I_0, as

$$I_1(x + u, y + v) = I_0(x, y). \tag{5.1}$$

This is known as the constant intensity constraint, as it assumes that changes in pixel brightness are only caused by motion. Optical flow can also be described on a coarser level, where several pixels share a motion vector. Such regions of pixels are typically small blocks, such as 4 or 8 pixels wide. This reduces the amount of motion information, exploiting the fact that neighbouring pixels usually have coherent motion.

A generalisation of this is to use a subsampled grid of motion vectors, and some form of interpolation to calculate the individual pixel motion vectors, such as Szeliski and Coughlan's spline-based system [101].

Parametric motion models

Describing motion on a pixel level or even a reduced resolution grid still requires a lot of motion data. A lot of this information is redundant, as the motion of nearby pixels often shows a strong coherency. Therefore it

is sometimes preferable to describe the motion of a large region of pixels using a single model that describes the motion of pixels between frames I_0 and I_1 using a limited number of parameters.

The simplest parametric transform is a translation, where each pixel P in the region is translated by the same vector, thus each pixel's position in frame I_1 is modelled as

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = P + T, \text{ where } P = \begin{bmatrix} x \\ y \end{bmatrix}, \tag{5.2}$$

and T is the translation vector. Clearly this model is much more constrained than the general optical flow. If the video is of a 3-dimensional scene, it requires any moving objects to be rigid, flat, and moving parallel with the image plane.

More general 2-dimensional motion can be described using an affine transform of the form

$$P' = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot P + \begin{bmatrix} a_5 \\ a_6 \end{bmatrix} \tag{5.3}$$

This is sufficient to model translation, scaling, rotation and skewing of 2-dimensional elements. In terms of 3-dimensional objects this allows us to model planar objects that are not parallel to the view plane, so long as they are only translated on their own plane, and they can only be rotated about an axis normal to the image plane [24].

These constraints are necessary because these simple 2-dimensional transforms are unable to model the parallax effects that result when points at different depths in the image appear to move by different amounts. Adding further parameters gives the eight parameter *quasi-quadratic* model, where the displacements of each pixel are given by

$$\begin{aligned} u &= a_1 + a_2 x + a_3 y + a_8 xy + a_7 x^2 \\ v &= a_4 + a_5 x + a_6 y + a_7 xy + a_8 y^2 \end{aligned} \tag{5.4}$$

For notational convenience we can gather the parameters together into a vector $\theta = [a_1, \ldots, a_8]^T$ and write

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{bmatrix} \theta \tag{5.5}$$

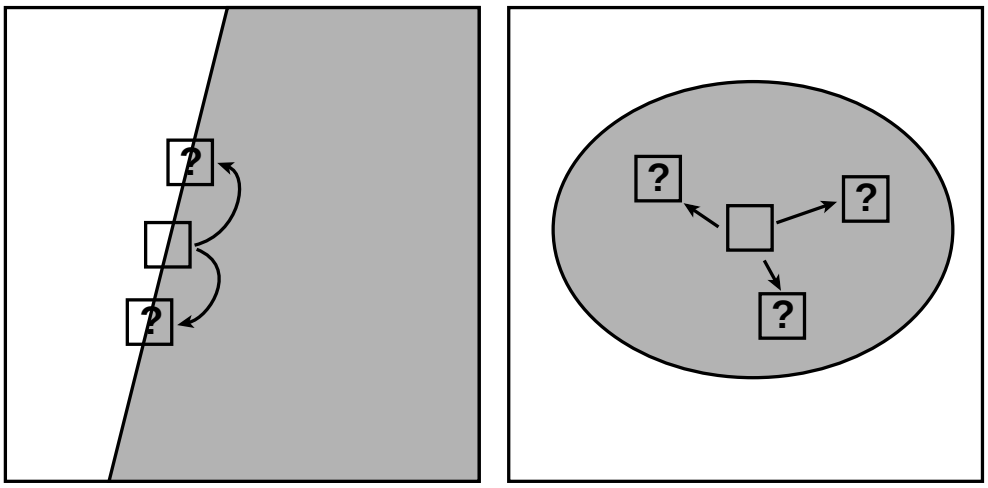


Fig. 5.1: Two examples of the aperture problem.

5.1.2 Estimating motion

Estimating the motion in a sequence is a well researched problem, and many techniques have been proposed. All motion estimation algorithms have to deal with the same problems:

Occlusion Motion in the scene can cause some areas to be occluded, and others to be revealed. These pixels have no corresponding pixels in the other frame, and so they have no well defined motion.

The aperture problem Where a portion of the image has a gradient in only one direction then the motion becomes ambiguous, as shown in figure 5.1.

Low detail areas Similarly, in low detail areas the motion is ambiguous. Also, when there is little detail, noise can become a problem.

Changing appearance/non-rigid objects If an object's appearance actually changes between frames, it can cause problems in determining the motion.

Block matching

Block matching is a straightforward way of estimating the optical flow. The second image is partitioned into small blocks and for each the algo-

rithm seeks the best matching block in the previous frame. The motion vector for each block is that required to translate the best matching block in I_0 to the position of the block in I_1.

Searching for matching blocks is a very compute intensive process, so the search can be limited to the local neighbourhood around the original block. However this may cause the algorithm to miss large displacements. A solution to this is to use a multiresolution pyramid, using the computed flow from the lower resolution levels to seed the search at the next resolution.

Block matching is popular for video compression as it can be implemented in hardware, and the matching process concentrates on reducing the error rather than reflecting the true motion in the scene.

The block matching algorithm leaves a large number of choices to the implementer, such as the metric used to compare blocks and the search strategy used. Kuhn et al. [48] compare a number of different implementations for use in MPEG-4 video encoding.

Differential methods

Differential methods compute per-pixel optical flow, and are based on the assumption that that points in the image maintain a constant brightness as they move through space and time. Therefore the total derivative of the intensity with respect to space and time is zero:

$$\frac{\delta I}{\delta x}\frac{\delta x}{\delta t} + \frac{\delta I}{\delta y}\frac{\delta y}{\delta t} + \frac{\delta I}{\delta t} = 0 \tag{5.6}$$

Rewriting with the flow components $u = \delta x/\delta t$ and $v = \delta y/\delta t$ gives us

$$-\frac{\delta I}{\delta t} = \frac{\delta I}{\delta x}u + \frac{\delta I}{\delta y}v \tag{5.7}$$

There is no single solution for u and v for this equation, so additional constraints are required in order to determine the flow.

Equation 5.7, known as the optical flow constraint equation, is the basis for two classic computer vision optical flow algorithms. Lucas and Kanade's technique [56] takes groups of neighbouring pixels and assumes that they share the same motion, giving a set of simultaneous equations that can be solved with a least squares method. Horn and Schunck's

method [39] makes an assumption about the smoothness of the flow field and uses an iterative method to try and minimise the derivative of the intensity.

Parameter estimation

If we use a parameterised model then there are are a number of techniques for estimating the parameters. If we already have per-pixel flow estimates for the pixels in the region then we can construct an overdetermined system of equations expressing the flow in terms of the model, and find a least squares solution to the model parameters.

We do not need flow estimates for every pixel in order to find a least-square estimate of the parameter. Since many pixel may have inaccurate flow estimates due to noise, lack of detail or the aperture problem as described above, it may be advantageous to use a much smaller number of flow estimates if we can rely on them being fairly accurate. One option is to use a robust least squares estimator [70], which ignores potential outliers.

Feature tracking is another technique; we estimate motion for a small number of points in the image, and use the motion of these points to estimate the model parameters. Feature tracking is usually done with block-matching; first a set of suitable features are identified in the first frame, and then a local neighbourhood search carried out in the next frame to find the best match. If the tracking is being carried out over a number of frames then the tracked motion can be filtered to eliminate probable bad matches. A Kalman filter can be used to model the motion of each feature, providing a prediction of where the feature will move in the next frame which can be used to guide the search [93].

A suitable feature for tracking should be a small region with high detail in both directions. Such features are sometimes called *corners*, as a corner is an ideal feature for tracking. Smith [95] provides a comprehensive survey of different feature detectors, and also proposes his own [94].

Parameter estimation is a well known problem in statistics and numerical computing, and there are a number of established methods for estimating a model's parameters. Sawhney and Ayer [82] use an expectation-maximization (EM) algorithm with a robust estimator, however the it-

erative nature of the algorithm makes it more compute intensive than a least squares solution.

Konrad and Dubois [47] propose a Bayesian approach, modelling motion as a Markov random field and using either simulated annealing or a relaxation technique to maximise the probability that the motion is correct. The technique requires a large number of iterations, and so has a high computational cost.

5.2 Spatial segmentation

Spatial segmentation is another well researched problem in computer vision. Segmenting a single image into meaningful units is a very difficult problem, and so it helps to be able to make as many assumptions about the image as possible. Single image segmentation has met most success in application specific tasks such as medical imaging where it is possible to make many assumptions about the images and there is a restricted domain of what can appear in the image.

General purpose segmentation has been less successful. If the algorithm has no knowledge of what may appear in the image and what types of object the user may be interested in it is forced to rely on low level image features to guide the segmentation. Many segmentation algorithms therefore try to segment an image into regions that are homogeneous with respect to some feature such as colour or texture. Motion can also be used as a feature if an optical flow field is available. Homogeneous regions in the image can be found by looking for transitions between regions, such as using an edge detector to find edges between regions of different colour or discontinuities in texture [57], and attempting to find contours that enclose the regions.

Clustering or classification techniques can be used to assign pixels to regions, particularly if the number and types of the regions is known beforehand. It is usually also necessary to add a constraint that pixels in the same region should all be connected. For example the Blobworld system [19] groups pixels using colour, texture and position features for each pixel. The use of position as a feature means that pixels will tend to be grouped in spatially coherent regions.

More common are partitioning approaches, where the image is partitioned into a number of connected regions through a process of splitting

and merging. Partitioning algorithms can take a top-down or bottom-up approach. The top-down approach starts with one region that encompasses the whole image. This region is then split into two, and the algorithm then continues by iteratively choosing a region and splitting it until some stopping criteria is reached. The means of splitting a region may be based on the pixel features such as colour or texture [26], or on a fixed geometry such as a quadtree.

Bottom-up algorithms start with small regions and iteratively grow them by adding pixels to each region. Pure region merging algorithms start with the image completely partitioned by many small regions, such as assigning one region per pixel. Other algorithms, such as the watershed algorithm [110], start with a small number of regions and merge neighbouring unassigned pixels into them. Both types of algorithm keep growing the regions until some stopping criteria is reached. Information from an edge-detector can be incorporated to guide the region growing process [89]. Bottom-up algorithms are easier to implement, as no decision has to be made about how to split a region. However if the final number of regions is small more iterations will be required. Region growing can be used after another technique has been used to seed the regions [22].

The criteria for splitting or merging regions are usually based on some model of the feature that is expected to be homogeneous over each region. The suitability of a given partition can then be measured as the error between the region models and the actual image data. For example, if regions are expected to be homogeneous in colour then a region can be modelled by its average colour and the error of the partition is then the difference in colour between each pixel in the region and the model. The object of each splitting or merging step is to produce a new partition that has a lower error.

A useful construct for a bottom-up region merging approach is a spanning tree [79, 80]. Figure 5.2 shows how such a tree is constructed, starting with a large number of small regions. The tree in figure 5.2 is in fact a binary partition tree, as each interior node has exactly two children. An advantage of constructing such a tree is that once it is constructed, a number of different partitions of the image can be quickly extracted as needed. For example if an n region partition is required, only the top n nodes are required. Salembier and Garrido [80] show how the spanning

tree can be used in many different image processing, segmentation and encoding tasks.

The partitions shown in figure 5.2 were built using a simple colour model operating in the YUV colour space; each region is modelled by an approximation to the median colour of the pixels in the region, and the L_2 distance between the models is used to decide the merging order. As can be seen, the extracted regions are not necessarily those we would desire from this image. We will explore the implications of building the spanning tree further in section 5.5.1.

5.3 Motion segmentation

As mentioned in the introduction to this chapter there have recently been many different motion segmentation algorithms proposed, all drawing to some extent on both motion estimation and spatial segmentation techniques.

5.3.1 Foreground separation

The simplest motion segmentation is to separate a moving foreground object from a static or quasi-static background. By quasi-static we mean the background is moving with a single motion that can be estimated and compensated for.

A basic approach is *change detection*, where each pixel is labelled as either foreground or background. A simple frame difference is sufficient to do this, but is highly susceptible to noise and will not work with a moving background. To handle a moving background, global motion estimation can be carried out on the scene and used to align the two images. Since we want to align the backgrounds we require a motion estimation technique that will provide a single motion model for the whole frame. A robust estimator is preferable, as it will ignore the motion of the foreground object.

A threshold is required to differentiate between truly changing pixels and noise. There is also the problem of low detail areas on the moving object, which will not trigger the threshold. To counter this, Aach and Kaup [9] introduce an adaptive threshold technique based on a Bayesian formulation of the problem. The algorithm adapts over the course of

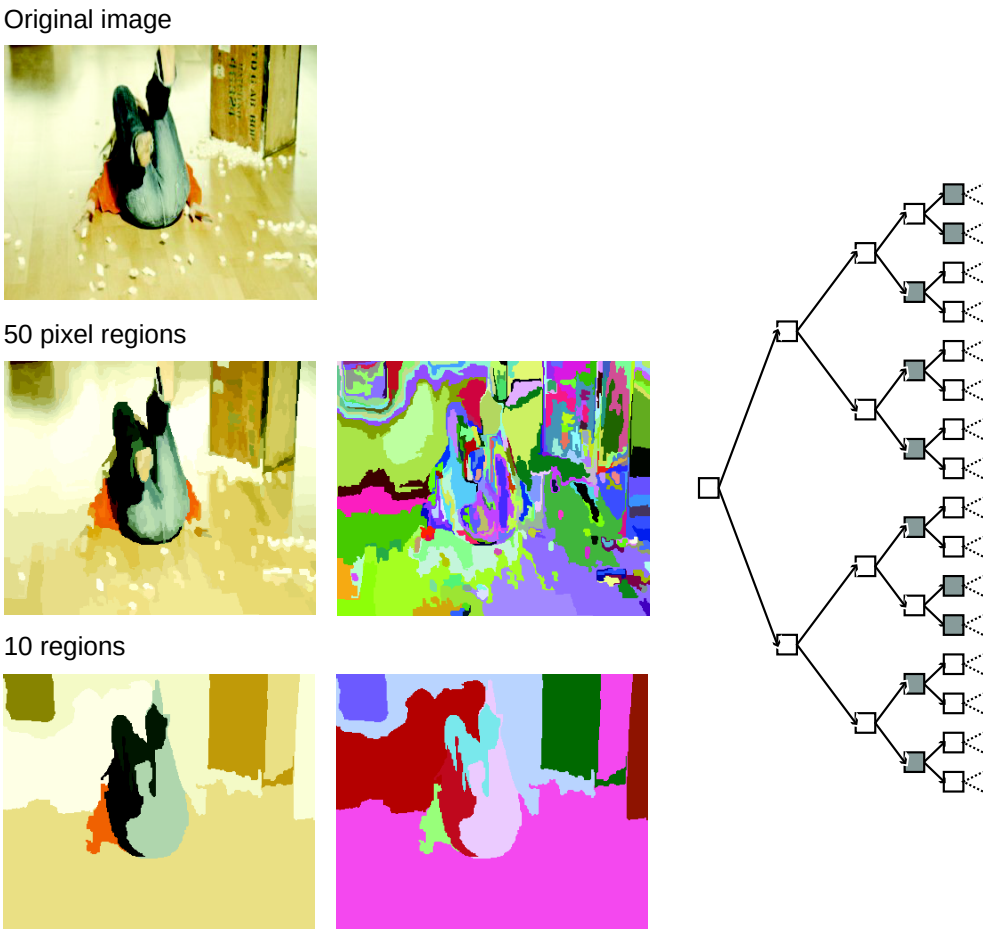


Fig. 5.2: A spanning tree constructed from an image using a simple colour model. Reconstructed images are shown for a segmentation with a minimum of 50 pixels per region and a segmentation with 10 regions, both constructed from the same spanning tree. The images on the left use the modelled colour for each region, while the images on the right use false colours.

a multi-frame sequence using the assumption that moving objects will form compact objects with smooth outlines while erroneously detected changes will appear as small scattered regions. Mech and Wollborn [61] extend this technique by adding a relaxation step, intended to simplify the outline of the object, and a memory, used to eliminate transient errors in the object mask. Neri et al. [67] present a similar approach, using morphological filters to carry out the regularisation.

5.3.2 Multiple model estimation

A drawback of change detection techniques is that they are unable to differentiate between multiple moving objects. They produce a binary mask where multiple objects are all labelled similarly as foreground pixels. If the masks of different objects are connected, they will appear as a single object.

An alternative is to try to estimate more than one motion model for the scene. This is a top-down approach, where we start with a single region encompassing the whole image, and estimate the dominant motion in the region. The dominant motion of the top region should be the camera motion, as we expect there to be more background pixels than there are pixels in any one object. Once the dominant motion is found, the pixels are separated into those which are modelled well by the dominant motion and those that are not. The region can thus be split into two—a region that is modelled by the estimated motion, and a region which is not. The process is then repeated for the region that did not fit the modelled motion. This can be repeated until all regions have good motion models, or until the regions are too small to estimate motion over. The motion models can then be refined by estimating the motion using only the pixels within each region [41].

Other authors have instead integrated support for multiple motion models directly into the motion estimation algorithm. For example, Sawhney and Ayer's E-M based motion estimation algorithm can be adapted to support multiple layers of motion [82]. After each iteration the optimal number of layers is estimated using a minimum description length principle. A drawback, however, is that the regions are not necessarily spatially coherent. Odobez and Bouthemy [71] use multiple parametric motion models and a Bayesian scheme for labelling each pixel

with the most suitable model. They too have a mechanism for adjusting the number or models, so coping with appearing or disappearing regions during a sequence.

5.3.3 Segmentation of the optical flow field

As mentioned in section 5.2 motion can be used as a low-level feature in a spatial segmentation algorithm, so another approach to motion segmentation is to compute the optical flow field for the frame and segment it into homogeneous regions. This requires a fairly accurate flow field, but estimated optical flow is often inaccurate around object boundaries. It also depends on the motion within each object being homogeneous, which is not necessarily the case for non-rigid objects such as people.

5.3.4 Integrated approaches

A major drawback of segmenting an image using motion alone is that although it is possible to determine the different moving regions the motion estimates are often poor around the edges of the objects, resulting in the extracted objects having inaccurate boundaries. In contrast the edges in a colour based segmentation are quite precise, but it is difficult to decide which regions are part of which objects. Therefore there have been a number of efforts to combine both spatial and motion segmentations.

Such techniques are typically initialised with a spatial segmentation of the first frame, usually based on homogeneous colour regions. The frame is oversegmented so that the contours of all the important regions will be preserved. A motion estimation step is then applied, and the calculated motion used to merge the spatial regions further into regions of homogeneous motion representing the independently moving regions of the scene. These regions are then motion compensated to predict their locations in the following frame, and these are used to seed the process for the next frame. This provides coherency of the regions between frames, and allows the tracking of objects throughout the sequence.

A number of systems have been proposed following this framework. We will focus on the COST 211 Analysis Module (COST AM) [11], which has been developed by the European COST 211 group for use in MPEG-4 and MPEG-7 applications, and is currently seen as the benchmark

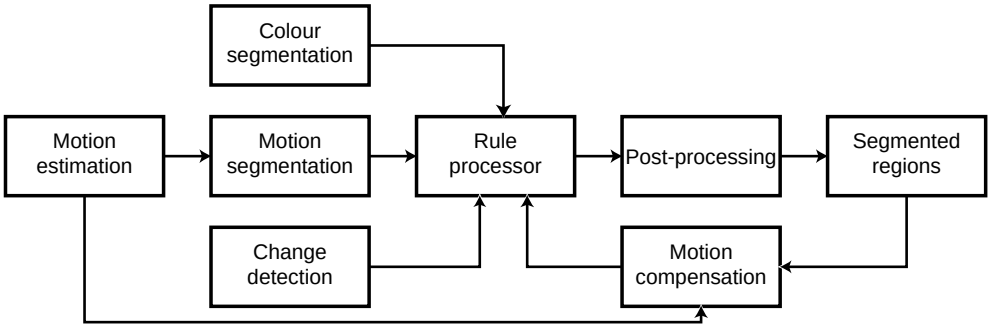


Fig. 5.3: Outline of the COST 211 Analysis Module.

against which other segmentation algorithms are compared.

5.4 The COST 211 Analysis Module

The COST 211 Analysis Module incorporates colour and motion based spatial segmentations and a change detection step, and uses a system of rules to combine these and identify the different moving regions. The output from the previous frame is used to assist analysing the current frame and to track regions throughout the sequence. The basic outline of the AM is shown in figure 5.3.

The model allows a fair degree of freedom for the implementer, who can choose between different algorithms for the colour and motion segmentations, and for the change detection. A pre-processing step can be added to compensate for global camera motion.

A similar framework is proposed by Herrman et al. [38]. Key differences between their approach and the COST AM are in the use of a different colour segmentation technique, and the addition of a shape analysis module. The aim of the shape analysis is to combine small regions into a larger region when the larger region has a simple coherent shape such as a rectangle. This helps to identify parts of occluded objects or larger objects that comprise of multiple colour regions.

5.4.1 The rule processor

The rule processor expects four inputs, which we will label following the notation of [11]. These are combined to produce the final segmentation

$\mathbf{R}^O$.

$\mathbf{R}^I$ The colour segmentation. The different regions of the colour segmentation should be of homogeneous colour. It is expected that the true contours of the objects in the image are represented by the edges of some of the regions in $\mathbf{R}^I$. The image should be oversegmented to ensure these contours are preserved.

$\mathbf{R}^M$ The motion segmentation. This is a segmentation applied on a flow field, and should consist of a small number of regions with coherent motion. Due to the inherent inaccuracy of motion estimation techniques around the edges of moving objects, it is expected that the boundaries of the regions will not be accurate.

$\mathbf{R}^{CD}$ The change detection mask. This is a binary mask indicating the moving and stationary parts of the image (after camera motion compensation). It is expected that the change detection mask will be a more reliable indicator of whether a region is moving or stationary than the motion segmentation.

$\mathbf{R}^{MC}$ If the segmentation $\mathbf{R}^O$ of the previous frame is available, each region is motion compensated to predict its position in the current frame, resulting in $\mathbf{R}^{MC}$. This is used to provide continuity between the segmentations of consecutive frames.

The aim of the rule processor is to produce a segmentation $\mathbf{R}^O$ of the principal objects in the scene using $\mathbf{R}^M$ to identify the different moving objects and $\mathbf{R}^I$ to provide accurate boundaries. A system of projections are used to match regions from the different segmentations using a projection operator. If $\mathbf{Y}_h$ is a region in $\mathbf{R}^Y$ then the projection operator $\mathcal{P}(\mathbf{Y}_h, \mathbf{R}^X)$ returns the region in $\mathbf{R}^X$ that has the greatest overlap with $\mathbf{Y}_h$. It is defined as

$$\mathcal{P}(\mathbf{Y}_h, \mathbf{R}^X) \doteq \mathbf{X}_{g'}, \text{ where } g' = \arg\max_g \left(a(\mathbf{X}_g \cap \mathbf{Y}_h)\right), \tag{5.8}$$

where $a()$ is a function that returns the area of a region in terms of pixels. The projection operator is used for each region $\mathbf{I}_i$ in $\mathbf{R}^I$ to find corresponding regions in $\mathbf{R}^M$ and $\mathbf{R}^{MC}$. Regions can be grouped using a group operator $\mathcal{G}(\mathbf{Y}_h, \mathbf{R}^X, g)$ which returns the set of regions $\mathbf{Y}_h$ in $\mathbf{R}^Y$ that project onto the same region $\mathbf{X}_g$ in $\mathbf{R}^X$. It is defined as

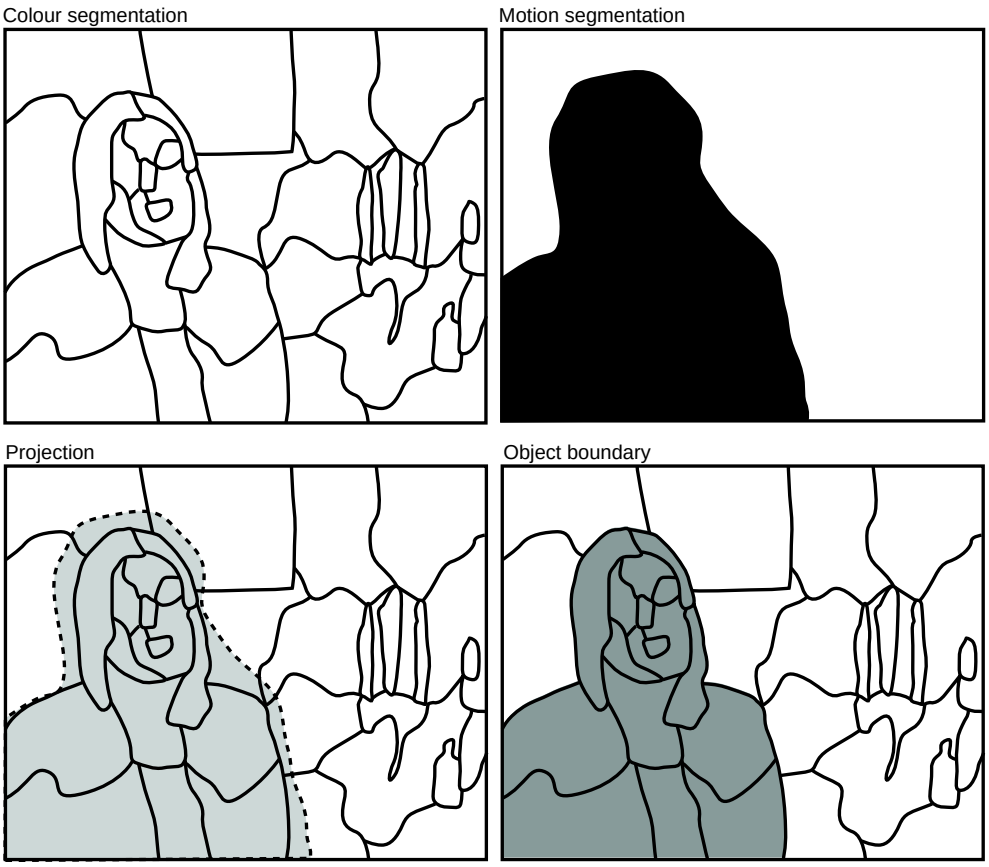


Fig. 5.4: Use of the projection operator to obtain an accurate object boundary using the colour and motion segmentations.

$$\mathcal{G}(\mathbf{Y}_h, \mathbf{R}^X, g) \doteq \mathbf{Y}_h : \mathcal{P}(\mathbf{Y}_h, \mathbf{R}^X) = \mathbf{X}_g. \tag{5.9}$$

Figure 5.4 shows a basic outline of how we can use the projection and grouping operators to find an accurate boundary for a moving object using $\mathbf{R}^I$ and $\mathbf{R}^M$. The colour and motion segmentations are produced independently, and then the motion region is projected onto the colour segmentation to identify the colour regions that are more than half covered by the motion region. These colour regions then provide an accurate boundary for the motion region.

A set of rules are applied to each pair of intersecting groups from $\mathbf{R}^M$ and $\mathbf{R}^{MC}$, i.e. for each pair (x, y) where $\mathcal{G}(\mathbf{I}_i, \mathbf{R}^M, x) \cap \mathcal{G}(\mathbf{I}_i, \mathbf{R}^{MC}, y) \neq \emptyset$. An additional set is defined to assist deciding what rules to apply. This

auxiliary set $\mathbf{B}_{x,y}$ for pair (x, y) is the set of regions from $\mathbf{R}^I$ that project onto group y in $\mathbf{R}^{MC}$ but *not* onto the group y in $\mathbf{R}^M$. Intuitively, the set $\mathbf{B}_{x,y}$ represents parts of the tracked object y that do not share the same motion as the rest of the object.

The following three rules are applied to each pair (x, y) of overlapping groups:

Tracking an object If $\mathbf{B}_{x,y} = \emptyset$ then we are tracking an object that projects onto a single motion region in the current frame. The object regions is given by the group $\mathcal{G}(\mathbf{I}_I, \mathbf{R}^{MC}, y)$ and the object's label from the previous segmentation can be propagated. This rule covers not only the tracking of moving objects, but also the background as it continues not to move. Also, if a previously segmented object starts or stops moving, it will be covered by this rule.

Newly exposed regions If the set $\mathbf{B}_{x,y}$ is not empty then we project the previously segmented object y (without motion compensation) onto the change detection mask $\mathbf{R}^{CD}$ for the previous frame. This tells us whether the object y was previously moving. If it was stationary ($\mathbf{R}^{CD}$ tells us that the region has not changed) then we have a part of a previously stationary region (such as the background) that has started to move. In this case we need to split the object y into two parts. The stationary part retains the label y while the newly moving part is given a new label.

Articulated motion If the set $\mathbf{B}_{x,y}$ is not empty but the object y *was* previously moving, then we have a part of a tracked object that has started to express a different motion from the rest of the object. This might be caused by an articulated object such as a person. In this case the part of the object with the new motion is split from the object to create a new object with a new label. The relationship between this new object and its parent can be recorded.

A post-processing step can be applied to handle any small extra regions that are produced as a result of motion compensation errors. Any regions with an area below a threshold are merged into one of their neighbours. On the initial frame there is no previous segmentation available, so

we simply label the entire frame as one initial background region—when objects start to move they will be detected and assigned new labels.

5.5 Implementation of the COST 211 AM

As mentioned above, implementation of the COST 211 Analysis Module allows a number of choices in the algorithms used for the different components. In this section we consider the alternatives and explain the choices we made for our implementation of the AM.

5.5.1 Colour segmentation

As discussed in section 5.2 the region merging approach to spatial segmentation has several advantages. In this case we desire an oversegmentation of the image in order to preserve the object boundaries. This leads to a fairly simple implementation of region merging using a spanning tree. We want our regions to be small and homogeneous in colour, so only simple rules are required in the merging process.

We use the algorithm described by Garrido, Salembier and Garcia [34] to generate a spanning tree from an image. The image is initially partitioned with one region per pixel, with each region's model being set to the colour of the pixel. A region adjacency graph is built, with links between each pixel and its four neighbours. Each link has a value defined by the *merging order* of the two regions it joins, and all the links are placed in a priority queue. The algorithm proceeds by taking the link at the top of the queue and merging the two regions it joins if they satisfy the *merging critereon*. A new region is created, with a model generated from the two child regions, and inherits the links of the children to their neighbours (which must have their merging orders updated to reflect the new region's model).

Our implementation is quite inefficient, especially when compared to the runtimes reported in [34]. This is due to our use of the container classes from the C++ Standard Template Library for managing the data structures, rather than building custom containers. In particular the ordered queue of links can become very large at the beginning of the process, when there is one region per pixel. The C++ STL implementation stores the queue as a red-black tree and a major overhead is finding and

removing links from this tree—each time two regions are merged all the links from the two regions must be removed from the tree and reinserted with new values. If there are many links in the tree with the same priority then the algorithm is reduced to doing a linear search through the links with the same value to locate the correct link for removal. Performance can be improved by adding a small random perturbation to the value whenever the merging order of a link is calculated. This makes most link values unique and allows them to be located in the tree in $O(\log n)$ time.

As mentioned before, the spanning tree algorithm requires us to specify a model, merging criterion and a merging order. In choosing these parameters our goals are to produce as coarse a segmentation as possible (with large regions) while preserving all the important boundaries in the image. We would like to avoid very small regions, as these will be difficult to motion compensate and will cause problems in the projection process.

Each region's model represents the colour of the whole region, which may range in size from one pixel to thousands. Good choices are the average or the median colour, as they are easy to compute and provide a good representation of the region's colour as long as the region is fairly homogeneous. This can be represented in any colour space such as the common RGB space, or the YUV space used by JPEG. The YUV space may have advantages for colour segmentation, due to its separation of the luminance and colour information.

The merging order determines the sequence in which regions are merged, and the objective here is to merge the regions that are most similar—to merge the pair of regions that will lead to the least degradation in the image after they are merged. It may also be desirable to use region size in the merging order, so as to guide the segmentation towards smaller or larger regions as desired. With a colour space such as YUV weights can be introduced into the merging order to guide the segmentation along luminance or colour boundaries.

The merging criterion is very important as it decides when the merging process stops, and can also introduce constraints into the process. If a minimum or maximum region size is required, it can be enforced by the merging criterion. Likewise a maximum acceptable error between a region and the original image can be enforced.

We test several configurations using the two source images shown in figure 5.5, at 352×288 resolution. One image has fairly clear colours

Fig. 5.5: Two source images for colour segmentation.

and boundaries while the other is quite dark and will be more difficult to segment. Any size constraints on regions are of course dependent upon the original resolution of the image.

Figure 5.6 shows the results of using only a minimum region size to produce the segmentation. The *RGB* colour space is used, the model is the median colour of the region, and the L_2 distance is used as the merging order.

With 50 pixel regions almost all the boundaries in the image are preserved; however the resulting regions are of course very small, and often an awkward long and thin shape as they follow contours in the image. Areas of texture, such as the floor and crate in the left-hand image are also segmented along the lines of texture. As we increase the minimum region size we start to lose detail. Using 1000 pixel regions gives very good segmentations of some parts of the image (such as the clothing in both images), but significant parts start to be lost, such as the man's face and the woman's right hand, which have been merged into the background as they are too small. A good compromise would appear to be in the 300–500 pixel range, where the important boundaries are preserved yet the regions are large enough to be useful. However even with the larger regions the shapes can be awkward as the regions are still following contours, such as on the woman's legs.

An alternative is to instead set a minimum merging order so that all links below this value are merged. This can be combined with a minimum region size by setting the merging criterion to be true if the merging order is below the threshold, or either of the regions is smaller than the

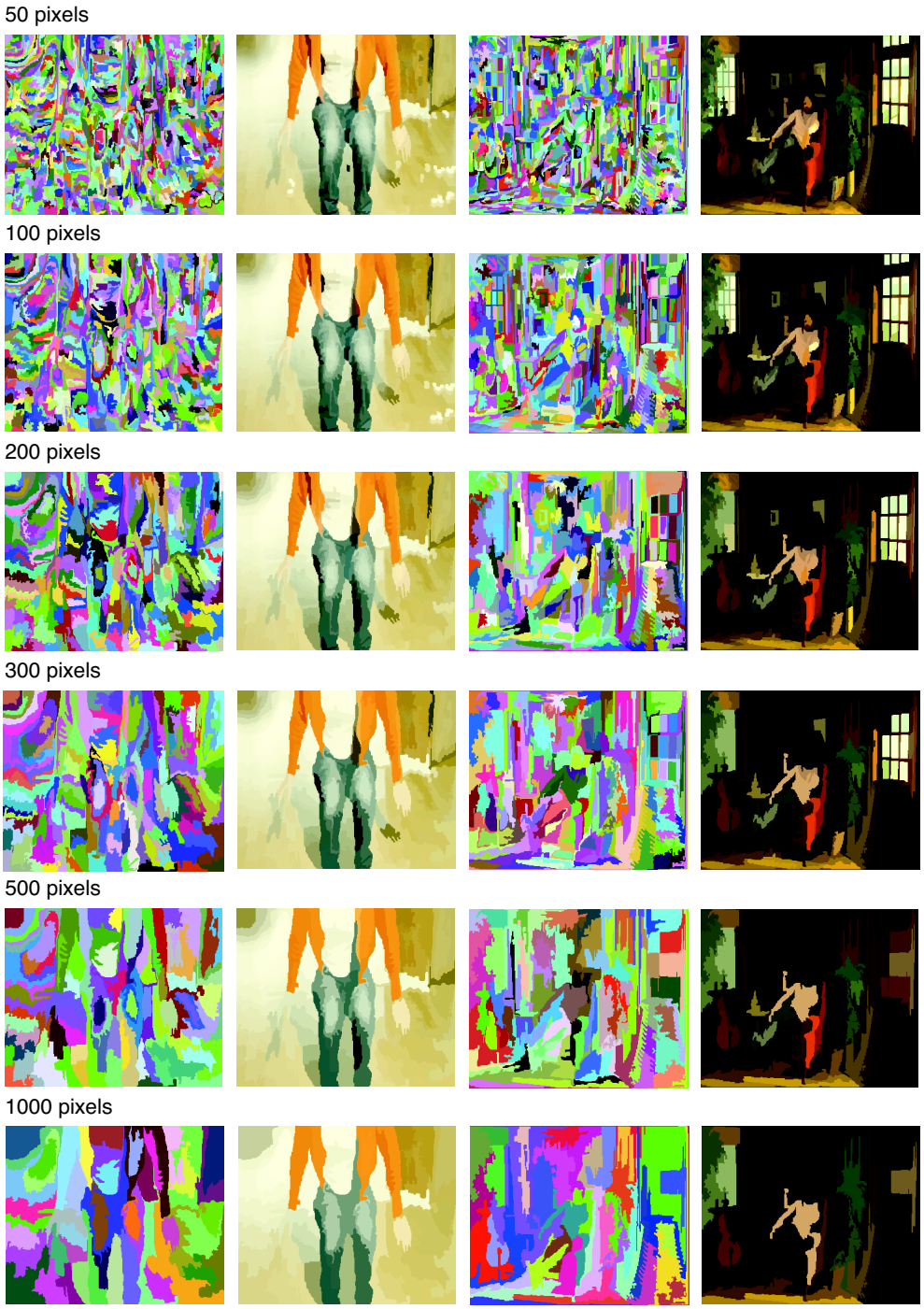


Fig. 5.6: *RGB* segmentations with different minimum region sizes.

minimum size. Figure 5.7 shows the results of this dual constraint approach, using two different thresholds for the L_2 distance between regions models. In this case we are working in the YUV colour space and the Y channel is given a weighting of 1/4 that of the U and V channels when calculating the distance between colours. This emphasises differences in chrominance over differences in intensity. This gives a good compromise, although if an object region is close to the colour of its background it can still be merged with it, such as the woman's right hand. This doesn't occur when using the minimum region size alone (providing the object is larger than this size), as merging will stop once the region reaches the set size. However large nearly uniform areas (which often occur in backgrounds) are segmented into a small number of large regions rather than many small regions, while the important object boundaries are preserved. We have found this approach, with a minimum region size of 300 pixels and a colour threshold of 50 to be the best configuration when working at this resolution.

5.5.2 Global camera motion

In the reference implementation of the Analysis Module [11] camera motion is modelled by an eight parameter parametric model. This models the scene as a flat plane that can be moved freely in 3-dimensional space. It is sufficient to model any camera movement, but will not account for parallax effects caused by differing depths in the image—however the amount of camera motion between consecutive frames is usually small enough that these effects are negligible.

The reference implementation uses Hötter and Thoma's regression technique [40], using only the pixels that have been marked as being part of the background in the previous segmentation.

Robust dominant motion estimation

We instead use the robust M-estimation technique proposed by Sawhney and Ayer [82, 83]. The use of a robust estimator makes this technique less sensitive to noise or regions that have been incorrectly labelled as part of the background.

The motion is represented by eight parameters $\theta_1, \ldots, \theta_8$ in a parameter vector θ. The flow at each pixel $P_i(x, y)$ is computed as in equa-

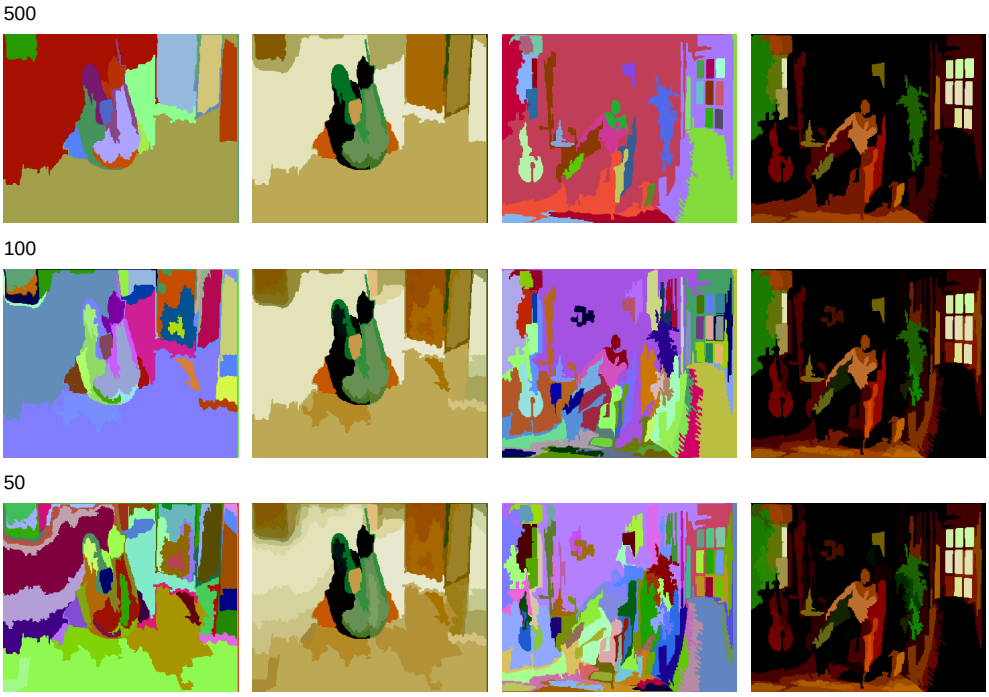


Fig. 5.7: Segmentation using a minimum region size and different minimum merging orders. Three different thresholds are used while the minimum region size is held at 300 pixels.

tion 5.5. We will define this as the function $\mathbf{u}(P;\theta)$, which takes a pixel's position and the parameter vector, and returns the flow at that pixel. The *residual* at each pixel is the error when the previous frame, I_0, is warped according to the flow and subtracted from the current frame I_1; the residual r_i for pixel i is

$$r_i = I_1(P_i) - I_0(P_i - \mathbf{u}(P_i;\theta)). \tag{5.10}$$

Conventionaly, to find the best model for the motion in the frame we wish to minimise $\sum_i |r_i|$. Rather than minimising r directly, we use a robust estimator $\rho(r;\sigma)$, which requires a scale factor σ and is defined as

$$\rho(r;\sigma) = \log\left(1 + \frac{1}{2}\frac{r^2}{\sigma^2}\right). \tag{5.11}$$

This is the *Lorentzian* function; other functions can also be used [83]. Thus the equation we wish to minimise is

$$\min_\theta \sum_i \rho(r_i;\sigma), \text{ where } r_i = I_1(\mathbf{p}_i) - I_0(\mathbf{p}_i - \mathbf{u}(\mathbf{p}_i;\theta)). \tag{5.12}$$

We minimise this equation using an iterative Gauss-Newton method. Each iteration step attempts to improve upon the current estimate of the parameters $\theta^{(m)}$ by moving along the descent direction $\delta\theta^{(m)}$. A new estimate of the solution is then given by

$$\theta^{(m+1)} = \theta^{(m)} + \alpha\delta\theta^{(m)}. \tag{5.13}$$

The descent direction is given by

$$\delta\theta^{(m)} = -\mathbf{H}^{-1}(\theta^{(m)})\mathbf{g}(\theta^{(m)}), \tag{5.14}$$

where $\mathbf{H}(\theta^{(m)})$ is the first order approximation to the Hessian of the error function of (5.12), a matrix given by

$$\mathbf{H}_{kl} = \sum_i \frac{\partial^2\rho}{\partial r_i^2}\frac{\partial r_i}{\partial\theta_k}\frac{\partial r_i}{\partial\theta_l}, \tag{5.15}$$

and $\mathbf{g}(\theta^{(m)})$ is the gradient, a vector given by

$$\mathbf{g}_k = \sum_i \frac{\partial\rho}{\partial r_i}\frac{\partial r_i}{\partial\theta_k}. \tag{5.16}$$

We can find $\delta\theta$ as the solution to the system of 8 linear equations

$$\sum_l \mathbf{H}_{kl}\delta\theta_l = -\mathbf{g}_k, \;\; k,l = 1,\ldots,8 \tag{5.17}$$

which expands to

$$\sum_l \left(\sum_i \frac{\partial^2 \rho}{\partial r_i^2} \frac{\partial r_i}{\partial \theta_k} \frac{\partial r_i}{\partial \theta_l} \right) \delta\theta_l = -\frac{\partial \rho}{\partial r_i^2} r_i \frac{\partial r_i}{\partial \theta_k}. \tag{5.18}$$

We can substitute the second derivative of $\rho(r)$ with its secant approximation, $\ddot{\rho}(r) = \dot{\rho}(r)/r$, which has the advantage of being positive everywhere. For the Lorentzian function this gives

$$\frac{\dot{\rho}(r)}{r} = \frac{2}{2\sigma^2 + r^2} \tag{5.19}$$

The derivative of r with respect to the model parameters can be expanded as [83]:

$$\frac{\partial r}{\partial \theta} = \frac{\partial \delta\mathbf{u}}{\partial \theta} \frac{\partial r}{\partial \delta\mathbf{u}} = \mathbf{M}^T \nabla I_1, \tag{5.20}$$

where $\mathbf{M}$ is the matrix in the quasi-quadratic model of equation 5.5 and ∇I_1 is the intensity gradient of the current image. Combined with equation 5.18 we can now compute the descent direction of the parameters, $\delta\theta$. A line minimisation [78] is performed along this direction to find the value of α and the estimate of the parameters are then updated according to equation 5.13.

The robustness of the technique to outliers comes from the use of the robust estimator $\rho(r)$. In equation 5.18 $\dot{\rho}(r)/r$ is a weighting for each pixel; pixels with high residuals are considered to be outliers, and have less influence over the solution. The detection of outliers is dependent upon the scale factor σ. In early iterations the estimate of the model parameters is likely to be poor so the residuals will be high, and σ should be chosen high so all pixels are included. As the solution improves, σ should be lowered so that pixels that do not fit the emerging model are discounted. We choose σ automatically, making it dependent upon the current residuals by defining it as

$$\sigma = 1.4826 \; median_i \; |r_i|. \tag{5.21}$$

The scale factor of 1.4826 required because we are taking the median of the absolute values [83]. By using the robust estimator and calculating the scale factor this way we can tolerate almost 50% of the pixels being outliers. A multiresolution pyramid is used so that larger displacements can be handled, with the solution for each level is projected to the next and used as the initial estimate.

Figure 5.8 shows the estimated camera motion for a sample sequence, using consecutive frames and also frames nine frames apart. Three levels from the multiresolution pyramid are used with six iterations of the algorithm at each level. As can be seen in the figure, the technique correctly identifies and estimates the background motion. The pixels of the moving man are discounted by the robust estimator, and the high residuals caused by them when the camera motion is compensated can be seen in the frame difference of the warped image. If we already had a segmentation of the man then we could provide a foreground/background mask so those pixels would be ignored anyway; however the algorithm copes well even without such a mask.

5.5.3 Optical flow estimation

As seen in section 5.1, there are numerous ways of estimating motion in a scene. In this case we require a general optical flow field that we can subject to spatial segmentation, so we can discount those methods that estimate parametric models for regions as we do not want to make assumptions about where the regions are, and a single parametric model will suppress the object motions.

In the reference implementation of the Analysis Module [11] a hierarchical block matching algorithm due to Bierling is used [15]. The block motions are then interpolated to give a per-pixel flow field. We decided to compare several optical flow algorithms before selecting one for use in our AM implementation.

As well as Bierling's hierarchical block matcher we will examine the Lucas and Kanade algorithm [56], the Horn and Shunck algorithm [39], Szeliski's spline-based flow [101] and a second block matching method implemented by the Intel OpenCV library [8]. The implementations of the Lucas and Kanade and the Horn and Shunck algorithms are also from the OpenCV library, while the others were implemented based on

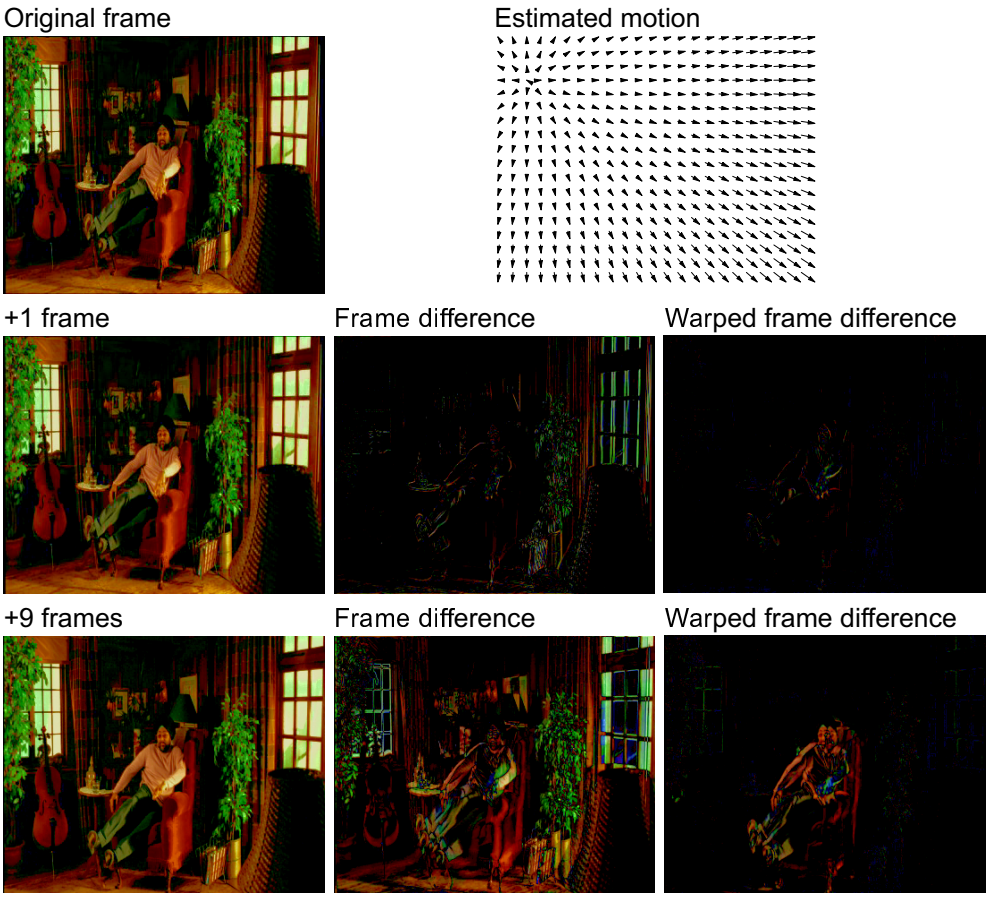


Fig. 5.8: Dominant camera motion estimation for two different intervals in a sequence.

the respective publications.

Figure 5.9 shows five different optical flow techniques applied to two frames from a sequence (the selected frames are three frames apart in the sequence—this interval means there is appreciable motion between the frames). The images are 352×288 resolution and are 'clean'—they have not been compressed at any point. Both the Lucas and Kanade and the Horn and Shunck techniques produce an overall correct flow field, but with appreciable noise. In particular the low detail areas are susceptible and there is evidence of the aperture effect around the edges of the window on the right, particularly with the Lucas and Kanade method. The first block matching method is hierarchical block matching as implemented by the Intel OpenCV library. The flow is on the most part consistent, but low detail areas again cause problems (in particular in the lower right corner) and there is again evidence of the aperture problem. An obvious problem is around the edges, where the correct flow should flow out of the frame, but instead matches are found within the frame, reversing the direction of the flow. The second block matching technique is an implementation of hierarchical block matching as described by Bierling [15]. It suffers greatly from noise, and is also limited in that it only handles integer offsets.

The final two examples are Szeliski's spline-based technique [101] using different sizes of spline patch. The use of 16×16 pixel patches produces a clean flow field; the smoothing effect of the larger patches may result in the loss of detail, but it helps to maintain consistent flow in the areas that otherwise cause problems. Using 8×8 patches produces overall correct flow, but with more noise. Both patch sizes still show some difficulties around the borders of the image.

Figure 5.10 shows the same techniques applied to an image more typical of what would be found on an online proxy server. The resolution is reduced (176×144) and the image has been heavily JPEG compressed; all the images are preprocessed with a gaussian blur before the optical flow calculation in order to reduce the JPEG artefacts. The results are in line with the previous example, although there is noticeably less noise present in the flow computed by the Lucan and Kanade and the Horn and Shunck methods. This may be because the magnitude of the motion is smaller in this example (it appears amplified in the figure because the images are at 1/4 of the resolution, and thus the pixels are larger).

Fig. 5.9: Different optical flow techniques.

Fig. 5.10: Different optical flow techniques with a low resolution and heavily compressed *Cakes* proxy.

Based upon these examples we have chosen to use Szeliski's spline-based method, as it produces a smoother and more consistent flow field. The noise present in the other techniques will cause problems when we try to segment the flow field in the next section. A drawback of the spline-based technique is that it is iterative, and not as fast as some of the others. The results here are produced using two levels from a multiresolution pyramid and 20 iterations per level, and complete in an acceptable amount of time.

The flow generated using the spline-based technique is generally smooth except near the borders where large erroneous flow vectors can be generated. To remove these we apply a relaxation of the flow field from the area inside the border into the spline patches on the border, removing the erroneous flow vectors.

It should be noted that the optical flow calculation takes place after the camera motion compensation, so the background motion will not be present. Figure 5.11 shows the flow computed after camera motion compensation; there is very little flow computed in the background, and the actual motion of the moving objects relative to the scene is much clearer. However the interpolation used in the camera motion compensation causes some blurring, leading to further errors in the flow estimation.

5.5.4 Motion segmentation

The reference implementation [11] segments the flow field in a very similar way to the colour image, by region merging with a spanning tree, except a two component motion model is used instead of a three component colour model, and the L_2 distance between models is used as the merging order.

Figure 5.12 shows the results of using this approach to segment the flow field into six regions. The example on the left uses a dense flow field, with one sample per pixel. This has produced very small regions around the larger flow elements and one misshaped (but still connected) region. Since we are using the spline-based optical flow algorithm the dense field is actually generated from a coarser mesh of control vertices using a set of spline basis functions. The example on the right of figure 5.12 shows the result of applying the segmentation on the grid of control vertices themselves (which are spaced 16 pixels apart). This gives similar results to the dense flow except the smallest possible size of a region is larger.

Fig. 5.11: Optical flow computed with the spline-based technique after camera motion compensation and border relaxation.

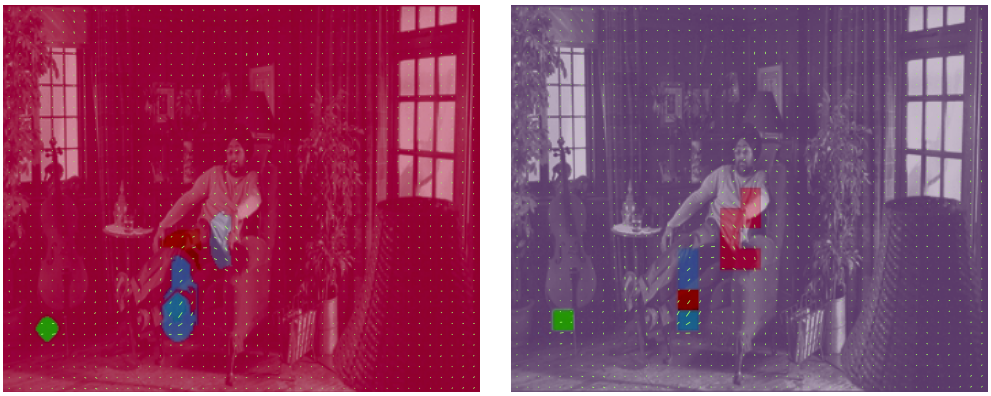


Fig. 5.12: Segmentations of optical flow into six regions using a spanning tree with a simple two parameter model. Dense flow (left) and sparse flow (right).

In both cases the regions produced are very small, and although they are mostly located in the area of interest much of the moving man has been merged into the background region.

A tendency towards small regions is not the only failing of using such a simple model. Tuncel and Onural [107] point out another weaknesses, namely that it can only handle 2-dimensional translational motion of objects. They propose instead that the spanning tree be built using a six-parameter parametric model. As can be seen from the optical flow in the example, the motion of the man is not rigid translation.

Tuncel and Onural propose using a six parameter model which can handle 2-dimensional affine transforms (any combination of translation, scaling, rotation and skew). The model parameters for a region are found by a least squares fit of the model to the optical flow in the region. A least squares fit needs a sufficient number of samples to work, so in cases where the region is too small for a least squares solution we initialise the model with translational motion using the average of the optical flow vectors within the region.

The merging order is calculated by finding the model for the union of the two regions and finding the error of this model compared to the actual flow. The errors of the models of the two existing regions are subtracted from the error of the new region, in effect giving a measure of how much worse the unified region will represent the flow compared to the existing situation. More formally, the error of a region R_i is

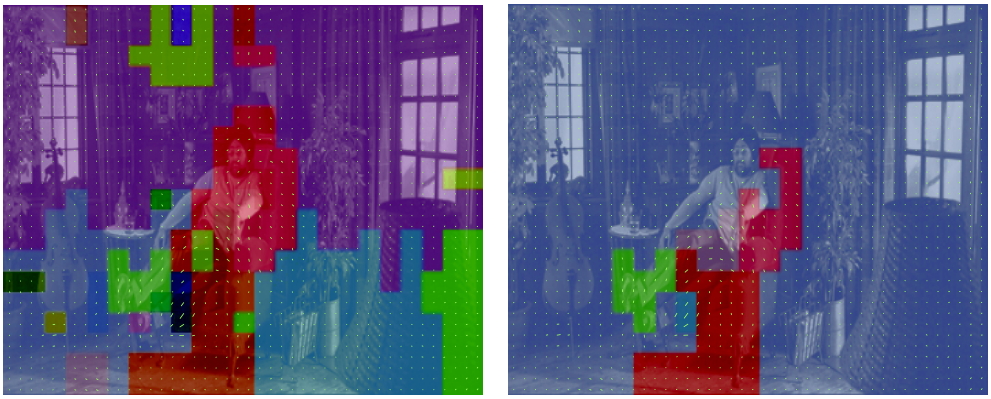


Fig. 5.13: Flow segmented with a six parameter model. The flow is segmented into 30 regions (left) and six regions (right).

$$E_{R_i} = \sum_{(x,y)\in R_i} \| \mathbf{v}(x,y) - \mathbf{w}_{R_i}(x,y) \|^2, \tag{5.22}$$

where $\mathbf{v}(x,y)$ is the optical flow at (x,y) and $\mathbf{w}(x,y)$ is the flow computed by the model at (x,y). The increase in error resulting from merging regions R_i and R_j is

$$d_{ij} = E_{R_i \cup R_j} - E_{R_i} - E_{R_j}. \tag{5.23}$$

This value can also be negative, as it is possible that a least squares fit of the model over the unified region will produce a model that better represents both regions. If we set the merging order of each link in the spanning tree to be $-d_{ij}$ then each merging step will merge the two regions that result in the smallest increase in error. Figure 5.13 shows the result of using this model and merging order, segmenting the flow into 30 and six regions. In the 30 region segmentation many of the remaining regions are in the area of interest, and background regions have been merged together at an early stage (into a region with a near stationary model). When the segmentation is reduced to six regions the motion of the man is still segmented, although with a highly inaccurate boundary.

We suspected that since the merging order is summing the error over the entire region, it may be biased against larger regions as these may have larger errors. We performed the same segmentations using the mean squared error, where the error of each region is divided by the size of the

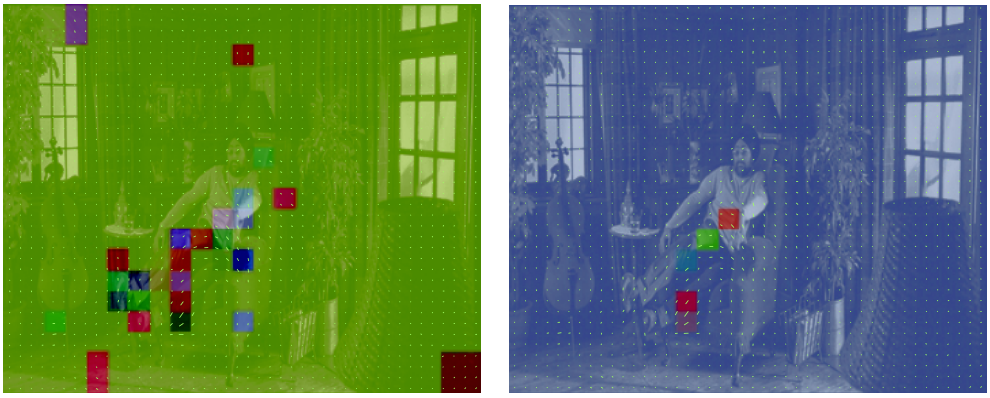


Fig. 5.14: Flow segmented using the six parameter model, but with mean squared error.

region. However the results, shown in figure 5.14, were much poorer.

Tradeoffs must be made in choosing the resolution of the optical flow and the segmentation. In order to get smooth and consistent flow with the spline based method we chose a fairly coarse grid (16×16 pixel patches). Segmenting this coarse grid leads to very inaccurate boundaries, as can be seen figure 5.14. It also limits the smallest size of moving regions that we can hope to detect—however all optical flow techniques have difficulties with small regions. If we instead use 8×8 pixel patches to compute the flow, allowing smaller regions and more accurate boundaries, the greater inaccuracy and noise in the computed flow causes a poor segmentation, as shown in figure 5.15. When the segmentation reaches 30 regions the main area of interest is still covered by some of the regions, but once the segmentation is reduced to 6 regions only one region remains in the area of interest, covering only a small part of it. Because the flow is not as smooth as when using larger regions it is more difficult to model a larger region using the six parameter model. The true motion of the man is complex and cannot easily be modelled with the six parameter model either, but the smoothing caused by the larger spline patches allows us to segment more of the man. Attempting to segment a dense per-pixel flow field suffers from the same problems, and becomes computationally infeasible due to the large number of regions, as we have to recompute several models every time two regions are merged.

An important issue when using a region merging segmentation is deciding at which point to halt the merging process. Various options were

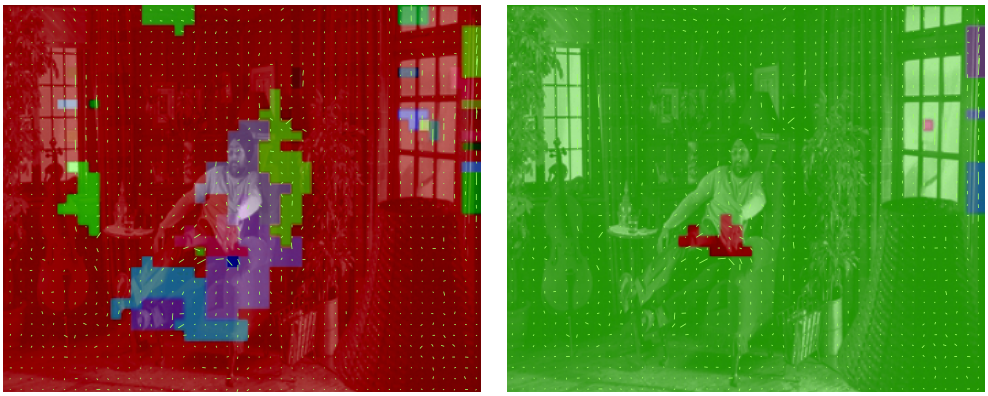


Fig. 5.15: Spline-based flow using 8×8 pixel patches.

discussed in section 5.2. The problem of halting the motion segmentation is more difficult than for the colour segmentation; the level of colour segmentation is not crucial, as long as it sufficiently oversegments the image to preserve the boundaries. The motion segmentation however determines the number and extent of different moving objects that will be tracked in the final segmentation.

If we decide that we want a fixed number of regions we can simply stop the segmentation when that number is reached. For example if we assume that the shot contains one principal moving object then we can fix the motion segmentation to produce two regions per frame. If there is more than one moving object in the frame then some are going to be missed, but a bigger problem is that the segmentation may not necessarily identify the same object in each frame, instead producing a segmented object that appears to move wildly about the frame.

Another approach is to use the increase in the error at each merging stage to determine when to stop. This value is readily available as it is also used for the merging order. A fixed threshold is unsuitable as the values can vary largely from shot to shot. Figure 5.16 shows plots of the increase in error at each of the last 16 merging steps for six example shots. The top four shots have one significant area of motion—in each case the moving object is a person, so the object motion cannot necessarily be well described by a single affine model. The lower left shot contains one principal object but also has a lot of background motion. The lower right shot has two moving objects.

We can see from the graphs that a fixed threshold is not suitable as

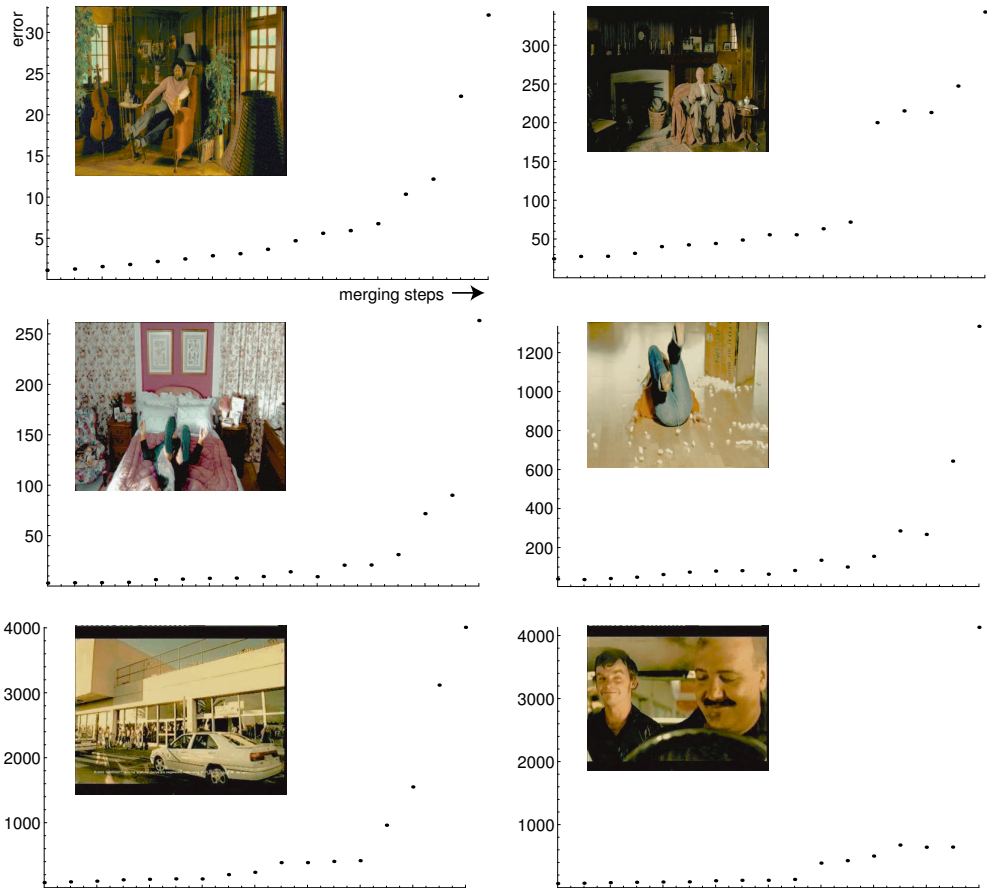


Fig. 5.16: Increase in error for each merging step.

the range of values varies enormously. Instead we should try to identify a discontinuity in the sequence where the errors suddenly get significantly larger than they were before. For some shots there appears to be an obvious point at which the errors become much larger, but for others it is not so clear.

In the graphs of figure 5.16 the discontinuity is where there is the largest difference between the error values produced by merging steps. Figure 5.17 shows the results when we stop the merging at the largest difference in errors. In the two middle shots the number of regions is correctly estimated, and the segmented regions are correctly positioned on the objects although the boundaries are highly inaccurate. The upper two shots have overestimated numbers of regions; in the first an extra region is positioned on the man, reflecting that the motion cannot be adequately described by the affine model. However the upper right shot has a number of spurious regions in the background, caused by erroneous optical flow estimates in the low detail areas. The lower two shots suffer from the same problem; in the lower left shot there is a spurious third region, but in the lower right shot the error in the flow is so great that it appear as the only moving region.

As has been mentioned already, the motion between consecutive frames is often quite small, and can easily be dominated by noise in the image. The lower right shot contains two moving regions, but has been through multiple transmission and compression processes and contains significant noise. When consecutive frames are used to compute the flow the motion is quite small and the noise causes significant errors, particularly in low detail areas and near the edges of the image. When the motion segmentation pass is applied (using the prior knowledge that there are two moving objects) only one is correctly detected because an area of erroneous flow has been incorrectly segmented as the second object. However if we increase the temporal gap to two frames when computing the flow, as shown in figure 5.18, then the apparent motion is larger, and the moving regions can be identified against the noise.

5.5.5 Change detection

The change detection mask is used in the rule processor to help differentiate between different cases as described in section 5.4.1. The reference

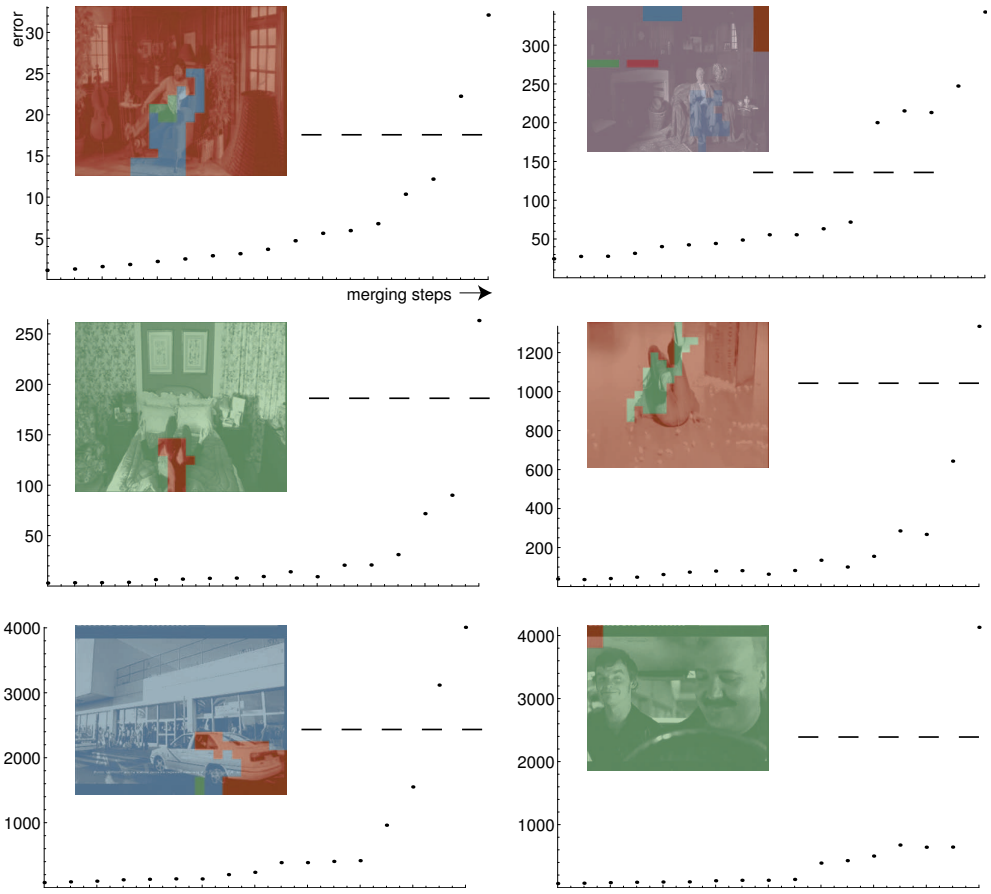


Fig. 5.17: Stopping merging at the largest error increase.

Fig. 5.18: Using a larger gap between frames means the apparent motion is not swamped by noise.

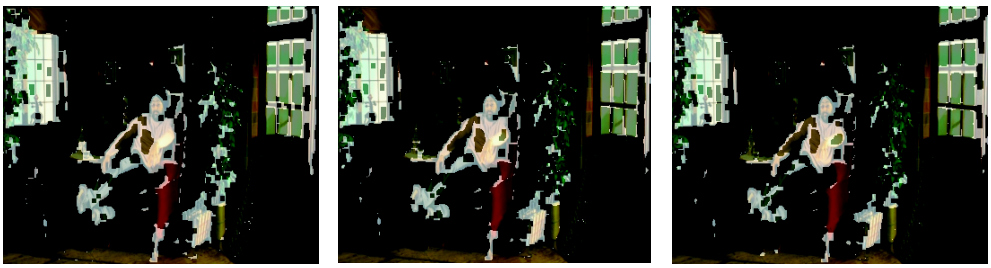


Fig. 5.19: Change detection masks computed from three consecutive frames (after camera motion compensation).

implementation of the Analysis Module [11] uses the change detection algorithm described by Mech and Wollborn [61].

This change detection algorithm seeks to avoid the problems of fixed threshold algorithms by using an adaptive threshold and a relaxation step to remove noise from boundaries and the background, and holes from objects. It is heavily based on the algorithm described by Aach and Kaup [9], who provide a more detailed derivation of the how the adaptive thresholds are chosen.

We have implemented the adaptive threshold algorithm of [9] along with a temporal memory and smoothing by a morphological closing operator, both as described in [61]. The memory helps provide stable object regions by remembering which pixels were changed in the past. If a segmentation of the previous frame is available then every pixel belonging to a moving region is marked as changed for the current frame—however to avoid error propagation we only carry over pixels that have also been detected as changed in the last n frames, where n is a fairly small number of frames such as 5.

Figure 5.19 shows change detection masks computed for three consecutive frames of our example sequence. Much of the man is labelled as changed in each frame, although the dark regions of his legs are not. Also some parts of the background have been marked as changed, in particular high detail areas such as the magazine rack and the edges of the windows. This is due to the camera motion compensation; after the camera motion is estimated, one frame is warped to the reference frame of the other using bilinear interpolation. The interpolation causes a blurring of the image and so causes differences between the two frames (although much less significant than if the camera motion compensation were not

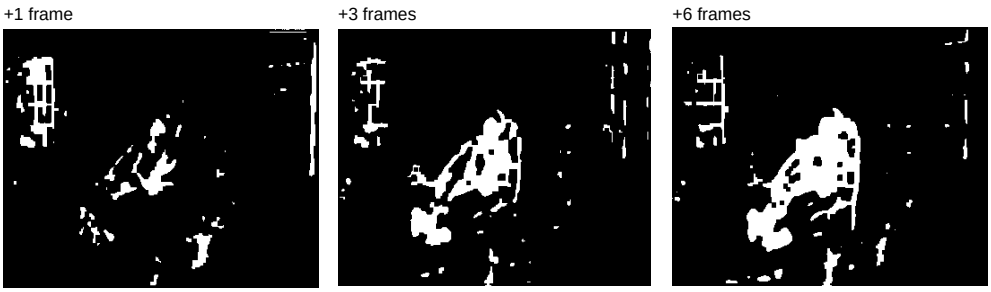


Fig. 5.20: Change detection masks computed for frames one, three and six frames apart.

carried out at all).

This exhibits a common problem with analysing motion when working with full frame rate video (25 frames per second in this case)—unless the scene is highly dynamic then the motion between consecutive frames is usually very small. In both the motion segmentation and change detection for this particular scene we have seen that it is difficult to differentiate between true motion and noise or errors introduced by the camera motion compensation. For example figure 5.20 shows the change detection masks computed for different frame intervals on the example sequence (with camera motion compensation). When the interval is larger the motion of the actual moving objects is larger than the noise, and the change detection algorithm can adapt better to differentiate between the two.

When we use the change detection mask to determine if a region has changed a threshold of 10% of the pixels in the regions is used.

5.6 Results

In this section we test our implementation of the Analysis Module on the example sequence to study its performance. As mentioned in the above section the different components of the AM allow a large degree of freedom in their implementation and configuration, and our implementation differs in many respects from the reference implementation.

5.6.1 The Analysis Module: Step by step

Figure 5.21 shows a breakdown of the application of the Analysis Module to the initial four frames of a sequence. The input sequence is 352×288 pixels and the colour segmentation is done in the YUV space, stopping when all regions have at least 300 pixels, and the motion segmentation produces six regions. Note that in this example the relaxation of the flow field at the borders has not been applied, so the motion segmentation often produces small regions at the borders of the image caused by erroneous flow. This configuration is by no means optimal, but will serve to illustrate how the Analysis Module works and how it is affected by the different components.

The initial segmentation at the start of the sequence is a single background region encompassing the whole of the frame. After the camera motion compensation the optical flow is computed and the motion segmentation finds three significant regions and three very small regions. These are projected onto the colour segmentation, and the three small motion regions disappear as they do not cover the majority of any one region in the colour segmentation. We now have three regions: the background, and two regions on the man's legs. All three regions overlap with the single region in $\mathbf{R}^{MC}$, so the AM rules will be applied to each of them in relation to the single background region.

When the two smaller motion regions are compared with $\mathbf{R}^{MC}$ the set $\mathbf{B}$ is not empty and the change detection mask indicates that the background region was stationary in the previous frame, so rule 2 of the AM is invoked. The change detection mask indicates that the two motion regions have changed in the current frame so the two regions are detected as new motion under rule 2 of the AM and labelled as new objects. When the larger motion region is processed the set $\mathbf{B}$ is again non-empty and the background stationary, but this time the motion region is also stationary so the other clause of rule 2 is invoked to track the stationary background.

On frame 3 the motion segmentation stage again finds two regions around the moving figure, however with different boundaries from those detected in the previous frame. Looking at the final segmentation for frame 3 we see that the boundaries of these two regions are preserved but merged together into a larger region, which is perhaps not the expected behaviour. What has actually happened is that the two regions have

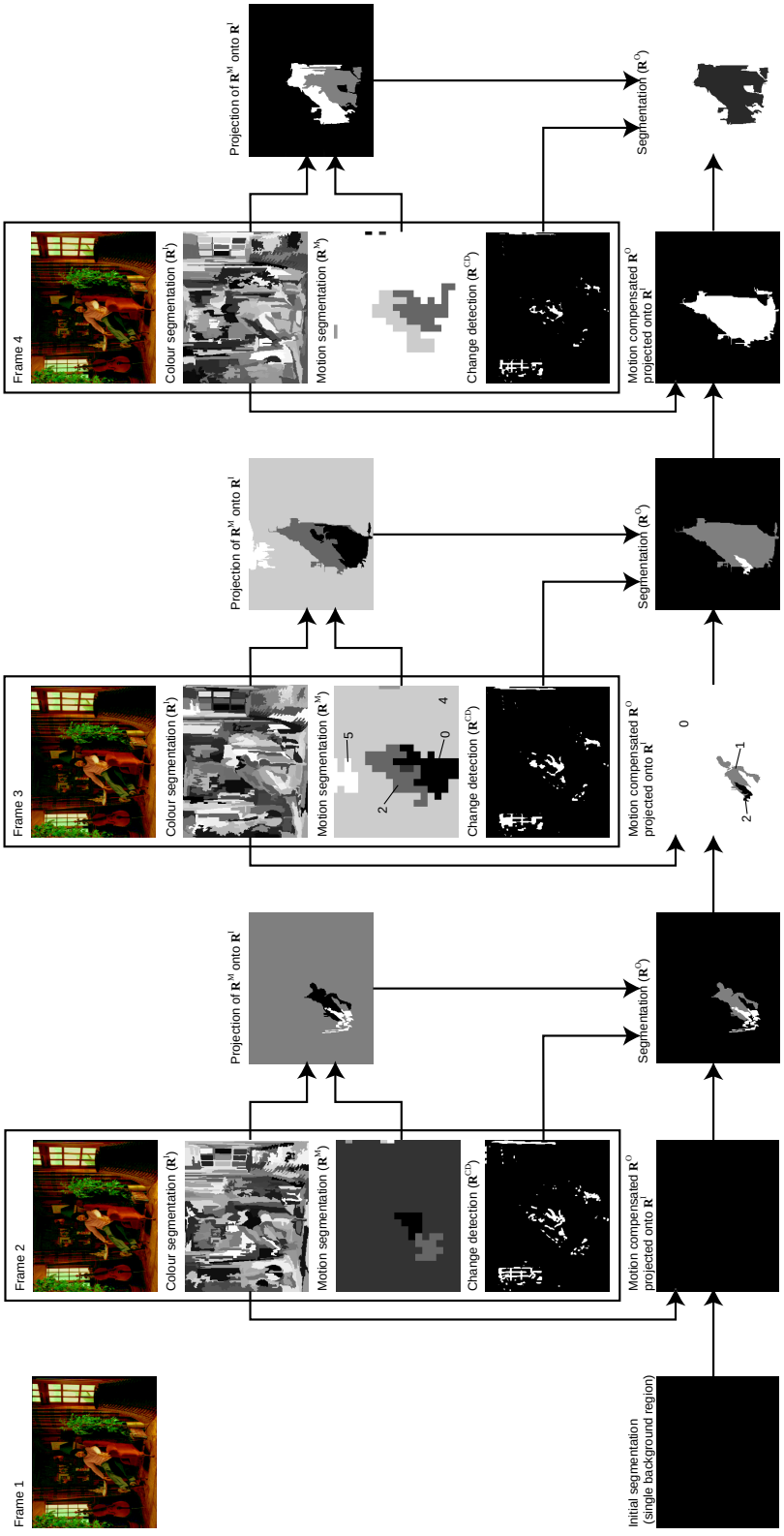


Fig. 5.21: Breakdown of the Analysis Module applied to a four frame sequence.

been labelled more than once.

Initially the two regions are compared with the background region in $\mathbf{R}^{MC}$ with which they both overlap. This results in both being detected as newly moving regions and given new labels under rule 2. However both regions also overlap with region 1 of $\mathbf{R}^{MC}$ and so are relabelled as tracking region 1 under rule 3 when compared against it. The small region 2 from $\mathbf{R}^{MC}$ is tracked under rule 1 as it is entirely contained within a single motion region. The erroneous motion region 5 is relabelled as part of the background under rule 2.

5.6.2 Adapting the colour and motion segmentations

We now adapt the colour and motion segmentation stages as suggested in the earlier sections. The colour segmentation uses both region size and contrast in the merging criterion, maintaining the minimum regions size at 300 pixels but allowing much larger regions in near uniform areas. The motion segmentation produces a variable number of regions using the largest error difference as the stopping criteria for the merging process. We use every third frame from the shot so that the motion is evident above any noise.

Figure 5.21 shows the intermediate steps when processing the same shot with these parameters. The motion segmentation usually identifies one, but sometimes two, moving regions positioned on the man, and these are projected and tracked through the sequence. However the inaccurate boundaries of the motion segmentation are carried through to the output segmentation. If the moving object was more homogeneous in colour then this would be alleviated somewhat as the colour segmentation would produce larger regions and a more accurate object boundary would be obtained from the projection; unfortunately this is not the case in this shot.

Figure 5.23 shows the extracted moving objects using this configuration; although the boundary varies greatly from frame to frame, the object is partly recognised and segmented. This compares favourably with the reference implementation of the Analysis Module (available from the COST group's web page [6]), which does not identify the man at all and is instead distracted by erroneous movement around the window area

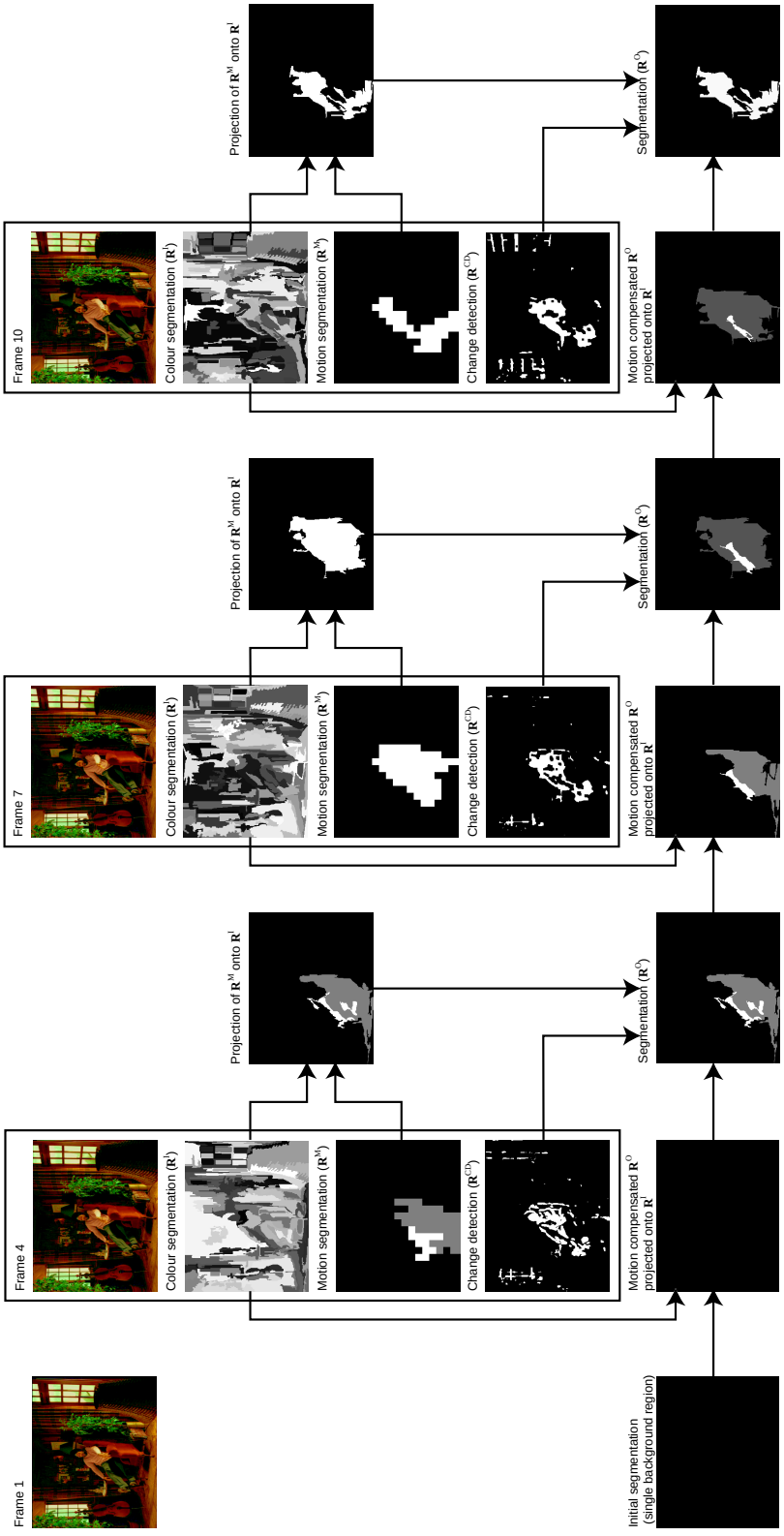


Fig. 5.22: Applying the adapted Analysis Module.

Fig. 5.23: Extracted objects from the apated Analysis Module.

Fig. 5.24: Segmentations from the reference implementation of the Analysis Module.

(figure 5.24).

5.7 Conclusions

Our aim in this chapter was to identify and isolate the principal foreground objects in a single shot from a video sequence. This is a very difficult problem that has attracted the attention of many researchers in the past, and will remain a major problem in computer vision for sometime to come.

We had only limited success in this task; we can identify and segment part of the foreground object in many cases, but not reliably enough to be of practical use. Even when the object is identified the outline of the segmented region varies wildly from frame to frame and so parts of the object are continually lost and reinstated.

Our approach involved taking an existing framework (the COST 211 Analysis Module) and examining each of the individual components to see how they could be adapted or tuned to improve the performance of the system. Although we succeeded in improving the performance slightly over the reference implementation the results are still far from ideal.

A number of problems contribute to this disappointing performance. The first is that the initial constraint we put on objects, that they have coherent motion over their spatial extent, is not suitable for a large num-

ber of objects, in particular people. This results in a moving person being identified as a constantly changing collection of objects, with each frame segmented into a different number of regions with different boundaries.

Another significant problem is the quality of the optical flow field which is used to perform the motion segmentation step. We found there was a marked tradeoff between resolution and noise when computing the optical flow field; methods that generated a high resolution flow field were very susceptible to noise. Using the lower resolution spline-based method eliminated much of this noise due to the smoothing it introduced. This smoothing also helps the motion segmentation phase as the smoothed flow is more likely to be well modelled by the affine model. However this technique severely affects the accuracy of the borders of the identified regions, and is still susceptible to noise, particularly with video that has low detail areas and a significant amount of noise introduced by analogue transmission and digital compression.

6. Presentation

This chapter is concerned with the presentation of a summary of a video sequence, using the information provided by the techniques of the previous two chapters. As explained in chapter 3, we have decided on a storyboard representation as the best way of concisely summarising the contents of a video sequence.

In a traditional storyboard each shot of a sequence is represented by a single frame, which is annotated to show the principal objects and motions in the shot, and also convey the *blocking*, or layout. We will assume that the sequence has been temporally segmented into shots, and that for each shot we have a sequence of bitmapped masks for each segmented object. We also assume that the estimated camera motion is available to us. Our aim is now to devise a system for automatically generating a 2-dimensional storyboard to be displayed on a screen or page, and from which a human can quickly understand the content of the original sequence.

The sequences of object masks produced by the technique in chapter 5 are not of sufficient quality for our purposes in this section. The object masks used in the examples in this chapter have been produced by manually labelling regions from a colour segmentation produced as described in section 5.5.1, with the exception of the car mattes in figures 6.7, 6.9 and 6.11, which were drawn by a *flame* artist.

6.1 Camera motion

Camera movement is ubiquitous in video—moving the camera adds dynamism and interest to a shot. Camera movement manifests itself as movement over the entire frame, affecting both the background and moving objects in the scene.

The camera can be described by its *extrinsic* parameters, namely its position and orientation in 3-dimensional space, and its *intrinsic* parameters, which control how the world is projected onto the film plane; we are typically only interested in one intrinsic parameter, the focal length, as the others are fixed by the camera. Although the camera can have

seven degrees of freedom in this parameter space it is rare for all the parameters to be varied at once. Camera movements are instead usually restricted to a set of well used and understood movements [16].

Tracking or dollying This is movement of the camera itself parallel to the ground plane. This motion can be parallel to the camera plane, or pushing into or pulling out from the scene.

Crane Vertical motion of the camera, typically achieved by mounting the camera on a crane or even helicopter.

Pan and tilt Horizontal and vertical rotation of the camera. In a pan the camera rotates around the vertical axis, and in a tilt around the horizontal axis (parallel to the film plane).

Zoom The camera itself does not move, but the focal length of the lens is adjusted, changing the magnification of the scene.

Different movements can be combined, for instance a pan may be required when tracking in order to keep the principal objects within the frame. If the camera is mounted on a crane or a steadycam then it can trace more arbitrary paths through space. The same applies to a handheld camera but with significant noise due to camera shake.

Changes in the camera's position and orientation can cause very similar results. In particular a track and a pan appear similar, as does zooming into or tracking into a scene. The differences are caused by perspective; in a zoom or pan the position of the camera remains the same so the perspective is unchanged. However if the camera moves then the perspective changes, which is manifested as a parallax effect where the foreground appears to move relative to the background.

6.1.1 Previous work

Sudhir and Lee [98] analyse the optical flow field to determine the dominant camera motion from the seven degrees of freedom. The flow field is separated into horizontal and vertical components and these are tested to determine whether they are *singular* or not (whether they disappear at the centre of the image). A system of rules is used to classify the motion. They acknowledge the difficulty in differentiating between actual camera

movement and changes in orientation, however they suggest that actual camera motion such as tracking will produce a greater variance in the magnitude of flow in the direction of motion. This model is later refined by Xiong and Lee [113] by separating the frame into eight regions and applying a system of rules based on the magnitude of the horizontal and vertical motion in each region.

Akutsu and Tonomura [10] use concepts from the field of computerised tomography to infer camera motions by analysing cross-sections taken from a video sequence treated as a 3-dimensional volume.

6.1.2 Camera motion in storyboards

Figures 6.1 and 6.2 show storyboards drawn for the film *Jurassic Park* (1993) that feature camera movements. The camera movement is indicated by an arrow either outside of the frame or overlapping the border, indicating that this is a camera motion and not an object motion. The arrows are also annotated with the type of motion. In the case of motion along the camera's view axis arrows are drawn at the corners pointing either into or out of the frame.

Figure 6.3 shows an alternative way of representing a long camera motion, where the viewpoint moves completely from one part of the scene to another. The frame is elongated and three separate drawings, showing the contents of different parts of the shot, are rendered within it.

6.1.3 Interpreting camera motion from global motion

As part of the motion segmentation process described in the previous chapter we performed a global motion compensation step, which involved estimating an eight parameter model of the background motion of the scene—in effect the 2-dimensional motion in the image caused by the movement of the camera.

We can use the estimated model to generate a flow field, however the generated field will be an approximation limited by the ability of the model to represent the actual flow in the image. The eight parameter model we use is sufficient to represent the apparent 2-dimensional motion in the frame, but not the parallax effect caused by points of different

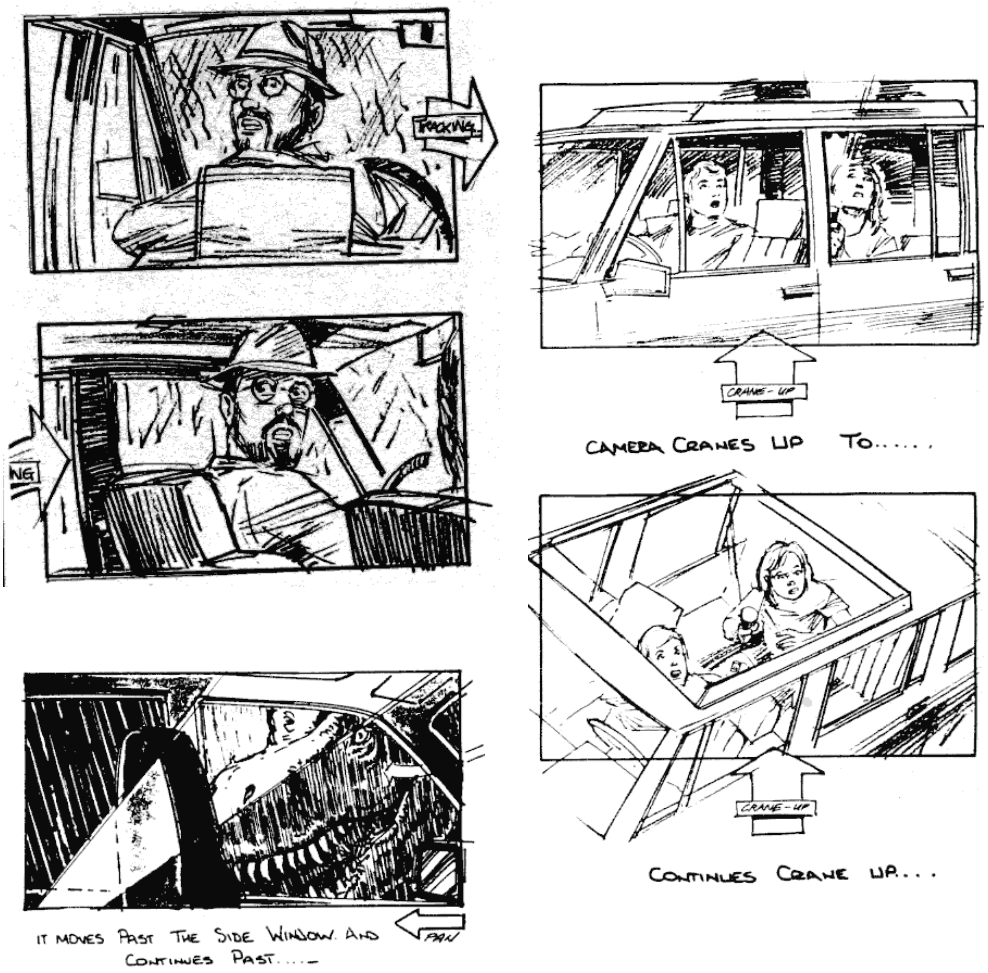


Fig. 6.1: Tracking, panning and a crane movement from the storyboards of *Jurassic Park* (1993) [87].

Fig. 6.2: Storyboards from *Jurassic Park* (1993) [87], showing camera movements into and out of the frame.

Fig. 6.3: Another method of portraying a pan [37].

depth. Thus we will not be able to use the variance of the flow magnitude to differentiate between tracks and pans.

However it is not essential that we be able to do so. The traditional role of the storyboard is as a planning and previsualisation tool used to guide production. Although the storyboard artist will indicate the anticipated camera motions they do so only in vaugue terms. The actual camera motions are decided by the director and cinematographer on the set during the process of *blocking* each shot, where the final actions of the camera and objects in the scene are planned before actually being shot. In our case we are not producing a storyboard as a preproduction aid, but rather as a quickly understandable visual summary of what has already been produced. For this purpose we argue that the difference between a panning or tracking shot is not important—that the camera is moving from right to left is, and so the contents of the image appear to move from left to right. If the viewer is interested in the finer points of the camera motions and perspective in a shot then they require more information than can be sensibly presented in the storyboard format, and would be better off viewing the shot itself.

Rather than deconstruct the components of a camera motion, or calculate the camera parameters explicitly, we prefer to present the apparent 2-dimensional motion caused in the frame by the camera motion. We do so by using the estimated model parameters to calculate the flow at each corner of the frame. A zoom or track into or out of the scene will produce vectors either all pointing into or out of the frame, and these arrows can be used to annotate a storyboard frame by placing arrows at the corners of the frames as done in the hand drawn storyboards of figure 6.2. Motion parallel to the film plane will produce corner vectors with consistent direction, and can be annotated with a single arrow outside the frame.

Figure 6.4 shows two camera motions annotated using the corner motion vectors. The frame on the left is part of a pan. The directions of the arrows are basically consistent, although note the inward bias and the small differences in magnitude between the two sides of the frame—these are caused by the differing depths of the scene at the edges and the way the camera lens projects the scene onto the image (lenses with shorter focal lengths exaggerate this effect more than longer lenses). For a motion such as this we could choose to label the frame with a single arrow, which would be sufficient to indicate the direction of motion, or

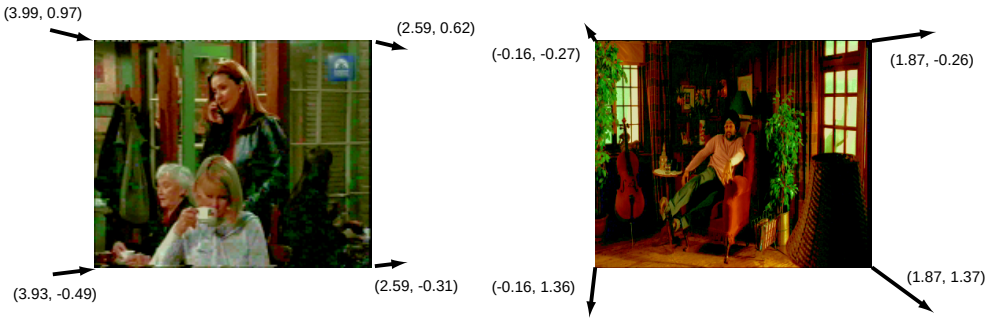


Fig. 6.4: Annotation of two camera movement using corner flow vectors.

to label the four corners, which give a cue to an experienced viewer that the motion is in fact a pan rather than a horizontal track.

This technique also handles more complex camera motions in a straightforward and intuitive way. Complex camera movements produce vectors of different magnitude at the corners, such as the off-centre zoom in the sequence on the right in figure 6.4. The motion in this sequence is a zoom (the perspective remains fixed) but the centre of expansion is in the upper left of the image. A regular lens could not produce this effect but it could have been created by cropping a larger frame, or though a non-uniform scaling of the image.

Annotating the corners with arrows gives a quickly understood indication of the motion. The outward direction of the arrows indicates that the image is expanding, the directions indicate the centre of expansion (at the intersection of the vectors), and the differing sizes show that the motion is greater towards the right of the frame. The other techniques described earlier would not be able to classify this motion, as the focus of expansion is not at the centre of the image and so the flow will not behave as expected in the fixed partitions of the frame.

While using the corner motions to annotate the frame provides a convenient way to indicate a single camera motion, they do not provide an obvious way to show a frame that has a sequence of different motions. Such compound motions are quite common in post-production type material, for instance where a mobile camera is following a moving (and often dancing) person. We have different options in such a situation; one is to break the shot into a sequence of consecutive camera movements and indicate each with a separate storyboard frame. Another option is

to choose a single motion to display, such as the motion which covers the largest number of frames or produces the largest overall movement.

6.2 Object outlines

The output from the motion segmentation system is a sequence of bitmap masks of each detected object. We can produce a vector outline of the bitmapped object mask; this requires less storage and produces a smoother contour, reducing the noise of the bitmapped mask, and will aid the presentation of the object's motion throughout the shot.

We use the technique of Schneider [84, 85] to produce a closed piecewise cubic Bézier curve around the contour of the object mask. The process begins by running an edge detector on the initial mask (we use the SUSAN kernel [94]) to produce a bitmapped outline. The outline is then divided into segments by looking for corners, where there is a discontinuity in the outline. These will become discontinuities in the Bézier curve, and are found by examining the angle between each pixel on the boundary and its neighbours n pixels away on either side; if the angle is below a threshold then the pixel is treated as a corner. The boundary segments between corners are treated independently for the rest of the curve fitting process.

An intial curve is fit to each boundary segment, as shown in figure 6.5; since the end points of the segment are known and the tangents at the end points can be found from the bitmapped boundary (using a least squares fit in the end points' neighbourhood), the only unknown that must be estimated is the lengths, α_1 and α_2, of the tangents which decide where the two internal control points are placed. This is done by a chord length parameterisation of the curve which allows us to associate each point on the bitmapped boundary with a point on the curve. An error function can then be defined as the sum of squared differences between each point on the bitmapped boundary and its corresponding point on the curve. An algebraic manipulation of the Bézier curve definition yields expressions for α_1 and α_2 which allow us to place the two internal control points [84].

If the resulting curve is a poor fit it can be subdivided into two segments. The point of maximum error between the curve and the bitmapped boundary is chosen to insert a new control point (reducing the error for that point to zero). The tangents for the new control point

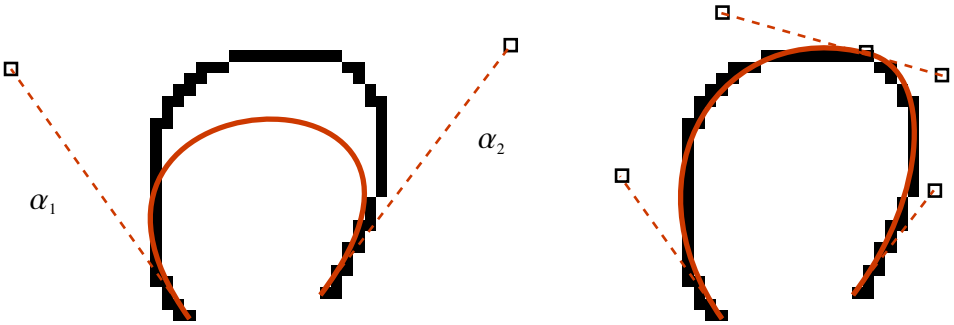


Fig. 6.5: Fitting of a Bézier curve to the outline of a bitmapped region.

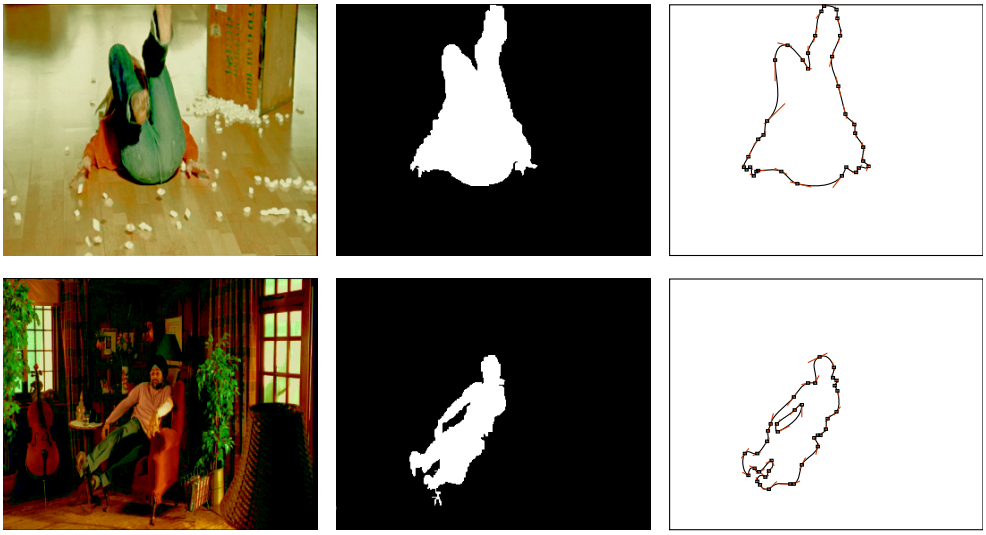


Fig. 6.6: Bézier outlines fit to segmented objects.

are estimated (the tangents at the join between the new curve segments must be parallel to preserve the smoothness of the curve) and the fitting process is repeated independently for each segment.

Figure 6.6 shows Bézier outlines generated for two segmented objects. The Bézier curve represents the complex outline of the object using a small number of control points that require little storage. As this is a vector representation it can be transformed without any degradation in quality.

6.2.1 Moving outlines

During a shot objects can move and change shape—in fact they must move as the segmentation algorithm only detects moving objects—and we can track the moving object mask with our Bézier outline. Given the bitmapped boundaries of two consecutive frames we can estimate a model of the motion required to transform the first boundary into the second. We use a 2-dimensional affine transform of the form

$$P' = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} P + \begin{bmatrix} a_5 \\ a_6 \end{bmatrix}. \tag{6.1}$$

The use of an affine transform rather than using per-pixel flow helps preserve coherency between the two outlines. We estimate the parameters of the transform by a downhill simplex method [78]. An error function is defined as the sum of the distances between each transformed pixel on the first boundary and the closest pixel on the second boundary. The simplex method attempts to minimise this function by adjusting the parameters of the transform.

The resulting transform is applied to the control points of the Bézier curve that was fit to the first frame and the resulting curve refined to fit the outline of the second frame. This produces a coherent animated Bézier outline rather than the collection of independent outlines that would result from applying the curve fitting process independently to each frame.

Figure 6.7 shows a sequence of masks with the corresponding bezier outlines. In the final frame the original outline is shown in red—the outline in the last frame has the same control points in the same relative positions as the original outline. Extra points can be added with each frame if required, but points are only removed if they actually merge. Although the wing mirror on the right of the car is absorbed into the body of the silhouette during this sequence, the control points that were required to define it in the original frame remain.

The masks in figure 6.7 were drawn by hand, and are smooth and consistent from frame to frame. However the masks produced by automatic segmentation are not as clean, and can have a significant amount of noise. For example figure 6.8 shows the outlines of masks produced from three frames of a sequence by labelling regions produced by a colour segmentation. As can be seen, the outline wobbles from frame to frame

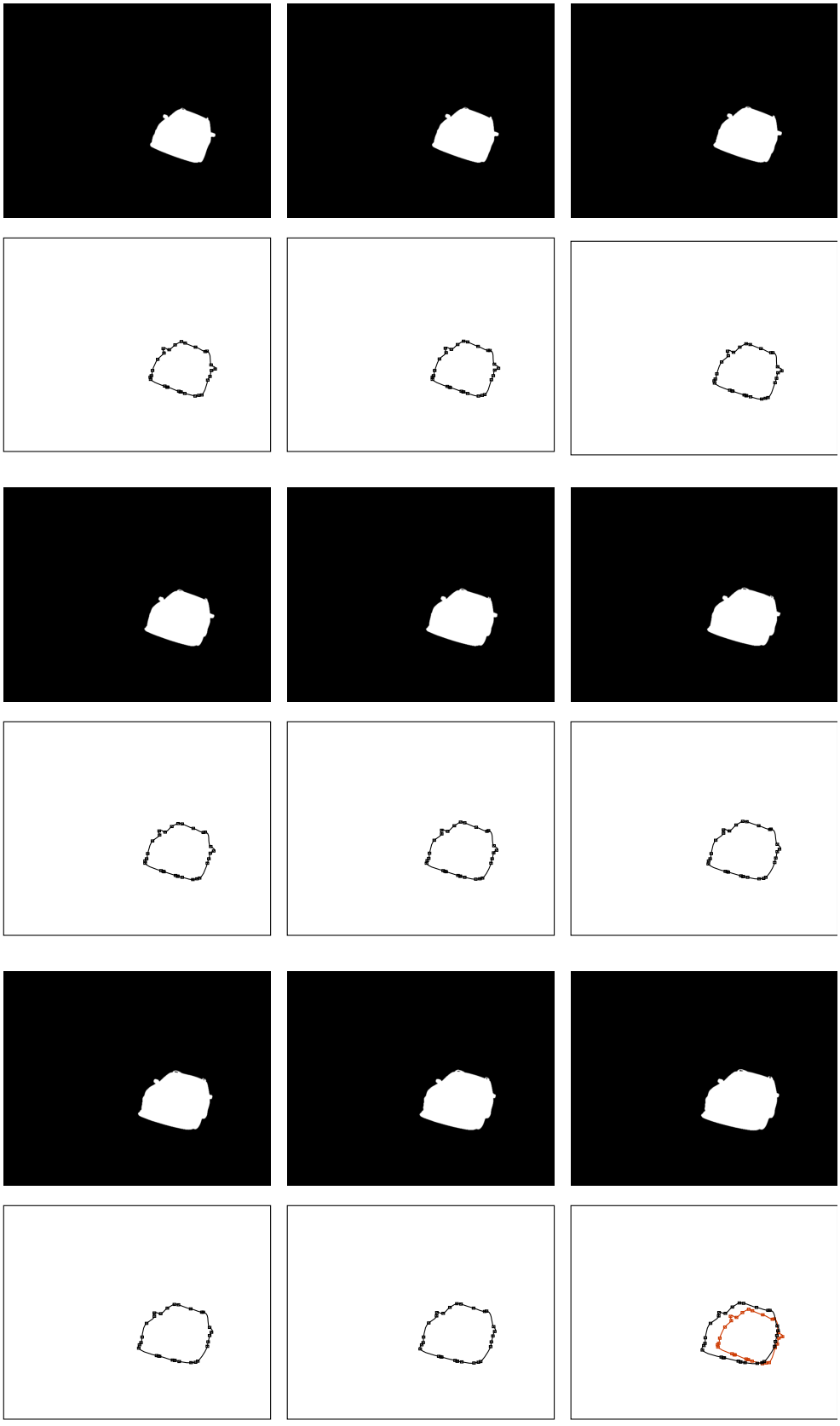


Fig. 6.7: A Bézier outline following a sequence of frames.

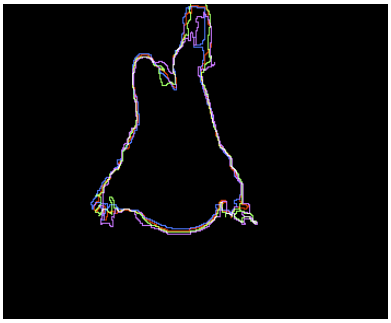


Fig. 6.8: Boundaries of the object masks produced using colour segmentation for four consecutive frames.

are significant, particularly around the moving parts of the object; this is caused because the colour segmentation is independent for each frame, and so the boundaries of the colour regions are not necessarily consistent from frame to frame. This causes problems for the motion estimation because if the shape varies too much then there will not be an affine transform that takes the mask from one frame to the next. The simplex solver can then show degenerate behaviour, often collapsing the entire shape to a point on the boundary, as this is a local minimum in the error function.

An alternative method for warping the outline to the next frame is to use active contours, also known as *snakes*. Active contours are a technique for fitting an outline to features in an image. An energy function is defined combining how well the curve approximates the image features with constraints on the curve itself, such as smoothness and continuity. By minimising the energy function, the curve adapts to fit the image features.

We use Williams and Shah's greedy algorithm to implement active contours [112]. The energy function we wish to minimise is

$$E = \int (\alpha_s E_{cont} + \beta_s E_{curv} + \gamma_s E_{image})\ ds, \tag{6.2}$$

the integral over the length of the curve of three functions that define the cost in terms of continuity, curvature and image features. Each function is multiplied by a weight which can vary over the length of the curve. In practice we discretise this equation by applying it to the control

points of the Bézier outline.

The image cost E_{image} is what causes the contour to follow the image. The image cost function is often based on the intensity gradient at that point on the curve, attracting the contour towards edges in the image. Since we are using a binary mask we instead define the image cost as the distance from the edge of the mask.

The continuity cost E_{cont} is intended to stop the points bunching up on the contour, which was a problem with earlier active contour algorithms [112]. This cost is minimised when the points are evenly spaced around the contour. If $\overline{d}$ is the mean distance between neighbouring points on the contour then we define E_{cont} for point i as $\overline{d} - |\mathbf{p}_i - \mathbf{p}_{i-1}|$. The curvature cost for point i is defined as $|\mathbf{p}_{i-1} - 2\mathbf{p}_i + \mathbf{p}_{i+1}|^2$. The weight β can be set to zero on corner points so that they are not penalised for their high curvature.

The algorithm is iterative. On each iteration, for each point on the contour, the cost functions is evaluated for each pixel in the 3×3 pixel neighbourhood around the point. The values of the three cost functions are normalised to the range $[0, 1]$ within the neighbourhood, and the control points at the end of each curve segment are moved to the point in the neighbourhood that minimises the cost function.

The contour is initialised with the control points of the previous frame, and iteration continues until the contour converges. The points on the active contour are then taken as control points for a new Bézier outline, tangents at the control points are recomputed and the curve refined as in the previous section.

The relative values of the weights are important, affecting the trade-off between a smooth, evenly spaced contour, and one that actually follows the edge of the mask but can become degenerate, with control points bunching up on the contour. This is because simply moving each Bézier control point to the closest point on the new boundary does not necessarily reflect the true motion of the mask. We found values of $\alpha = 0.5, \beta = 0.5$ (for non-corner points) and $\gamma = 1.2$ to produce a good balance; the image function has the highest weighting, forcing the contour to follow the mask. However there is no combination of weights that will work in all circumstances. Figure 6.9 shows an example of an incorrect contour—one of the control points on the wing mirror has 'jumped' across the gap. This causes problems for the curve fitting procedure as

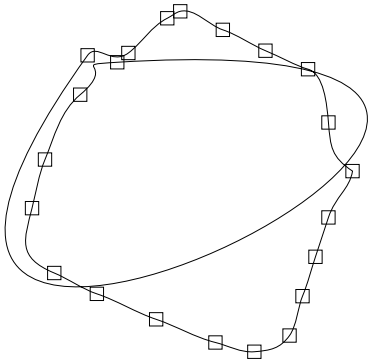


Fig. 6.9: An adaptive contour where a control point has 'jumped' across a narrow part of the mask.

it expects the curve to continue in the direction of the tangent at each control point, not to go 'backwards'. Such problems are avoided using the affine transform technique as the motion is calculated on a global rather than a local level.

The active contours can handle the noisier motion of the automatically generated masks, such as shown in figure 6.10. The affine transform method failed on this sequence, but the active contour manages to track the outline. However the motion of the tracked contour is not as smooth as would be expected from the affine transform. Small detail areas are lost, and there is significant movement of the points along the contour itself from frame to frame. For example figure 6.11 shows the paths of the control points in the car sequence when tracked with the affine transform and with the active contours. With the active contours the points follow fairly smooth and consistent paths and their positioning on the contour remains consistent, whereas with the active contours there in considerable noise in the paths.

6.3 Rendering

Although outlines can be a powerful visual cue, they are not always sufficient to actually convey what an object is. We wish to further enhance our simple vector representation with information from the original frame.

Fig. 6.10: A sequence tracked using active contours.

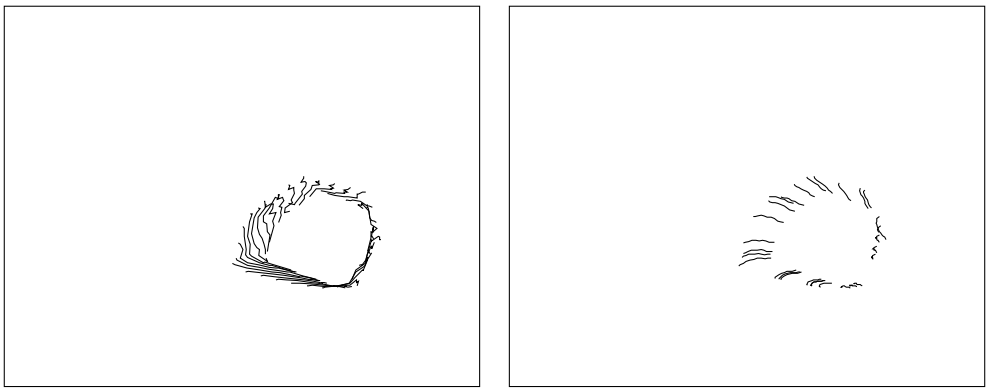


Fig. 6.11: The paths of the control points in a sequence tracked with active contours (left) and an affine transform (right).

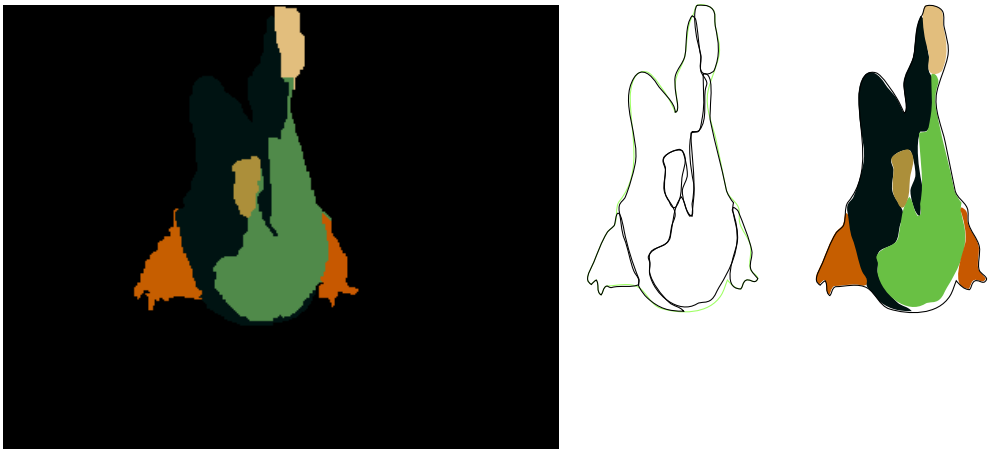


Fig. 6.12: Colour segmenting the interior of an object.

We can find the main coloured sections of an object using the spanning tree technique introduced in section 5.5.1. Figure 6.12 shows how a colour segmentation can be applied to an object area; the segmentation tree is initialised using only pixels within the object area. The merging then progresses using a distance threshold as the merging criterion—this threshold is set high, resulting in a small number of larger regions with high contrast between them. Any small regions are discarded, the remaining large regions have Bézier boundaries fit to them, and these regions are used to colour the interior of the object boundary. The interior regions do not exactly match the extracted outline of the whole object, as the interior regions are also approximations, so the boundary curve of the object is used to clip the interior regions so they do not spill out of the object.

Additional detail can be conveyed by highlighting edges within the object. The original object is processed with an edge detector to find high contrast lines. Chains of connected boundary pixels are found and Bézier curves are fit to them. These curves are drawn in black over the colour filled regions to provide additional highlights and detail to the object.

The background can be treated in a similar manner. It is important to represent the background, as this is the context of the extracted objects in the shot. If one of the background regions is substantially bigger than the others we can use it as the background colour for the whole frame, and

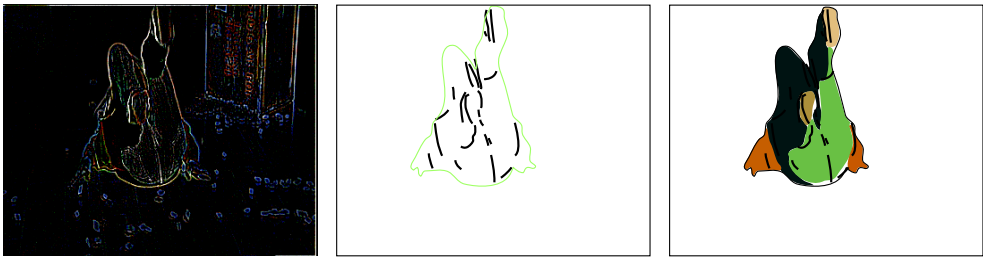


Fig. 6.13: Using an edge detecctor to find additional detail lines within the object.

place the other background regions on top. Figure 6.14 shows how the different layers are composited to form a final image, shown in figure 6.15. To further make the distinction between the moving foreground object and the background clear we can lighten the background, de-emphasizing it and focusing attention on the foreground object.

6.3.1 Moving objects

Object motion is one of the most important pieces of information to be presented in the storyboard. Figure 6.16 shows storyboard frames from the film *Jurassic Park* which include annotations of object motion using arrows. The final frame indicates complex movement—more complex than can be detected with our 2-dimensional analysis techniques.

Another technique used to indicate motion in 2-dimensional drawings is speedlines, shown in figure 6.17. These are lines originating from the object and travelling in the direction opposite to the object's motion. A variation is repeated contours, where the trailing contour is repeated several times in the object's wake. Masuch [59] synthesised speedlines and repeating contours to indicate motion in line drawings generated from moving 3-dimensional models.

Masuch suggests using the control points of a 3-dimensional model as potential starting points for speedlines; similarly we can use the control points of the contour of the moving object. The path of the speedlines can be taken from the actual paths of the control points as the object contour is tracked through the sequence. We take only the paths that travel outside the object (so only the rear of the object generates speedlines), thin out any lines that are clustered together, and smooth the lines.

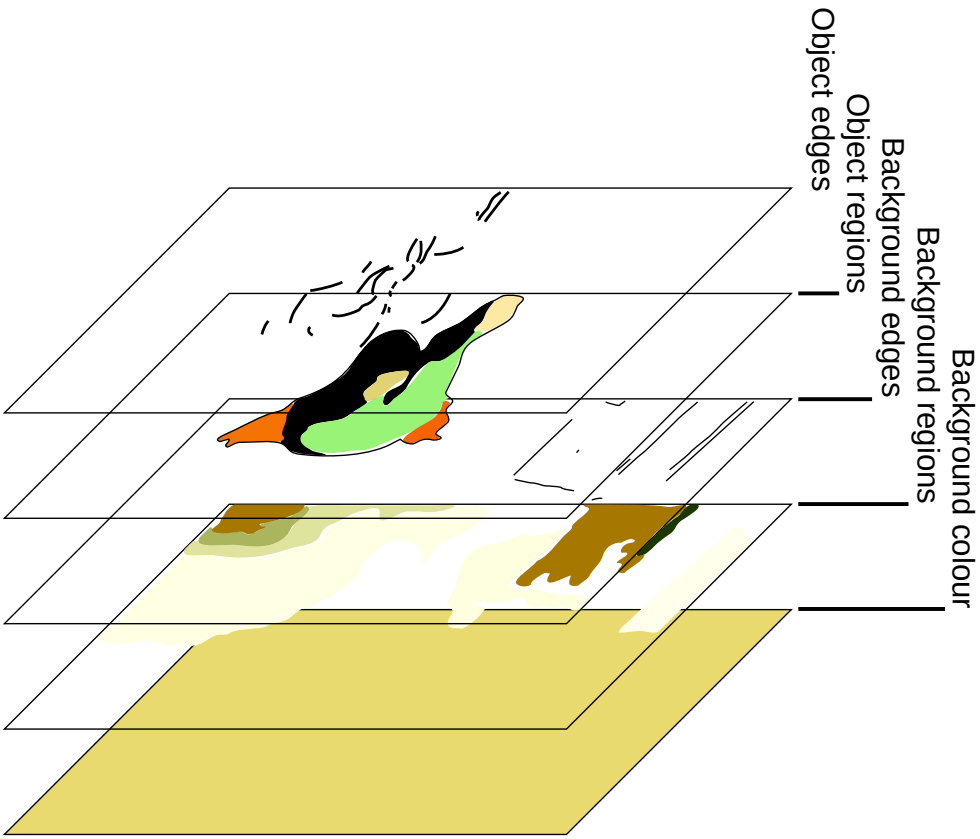


Fig. 6.14: The different layers superimposed to form the vector representation of a frame.

Fig. 6.15: The layers compositied together, and with the background de-emphasised.

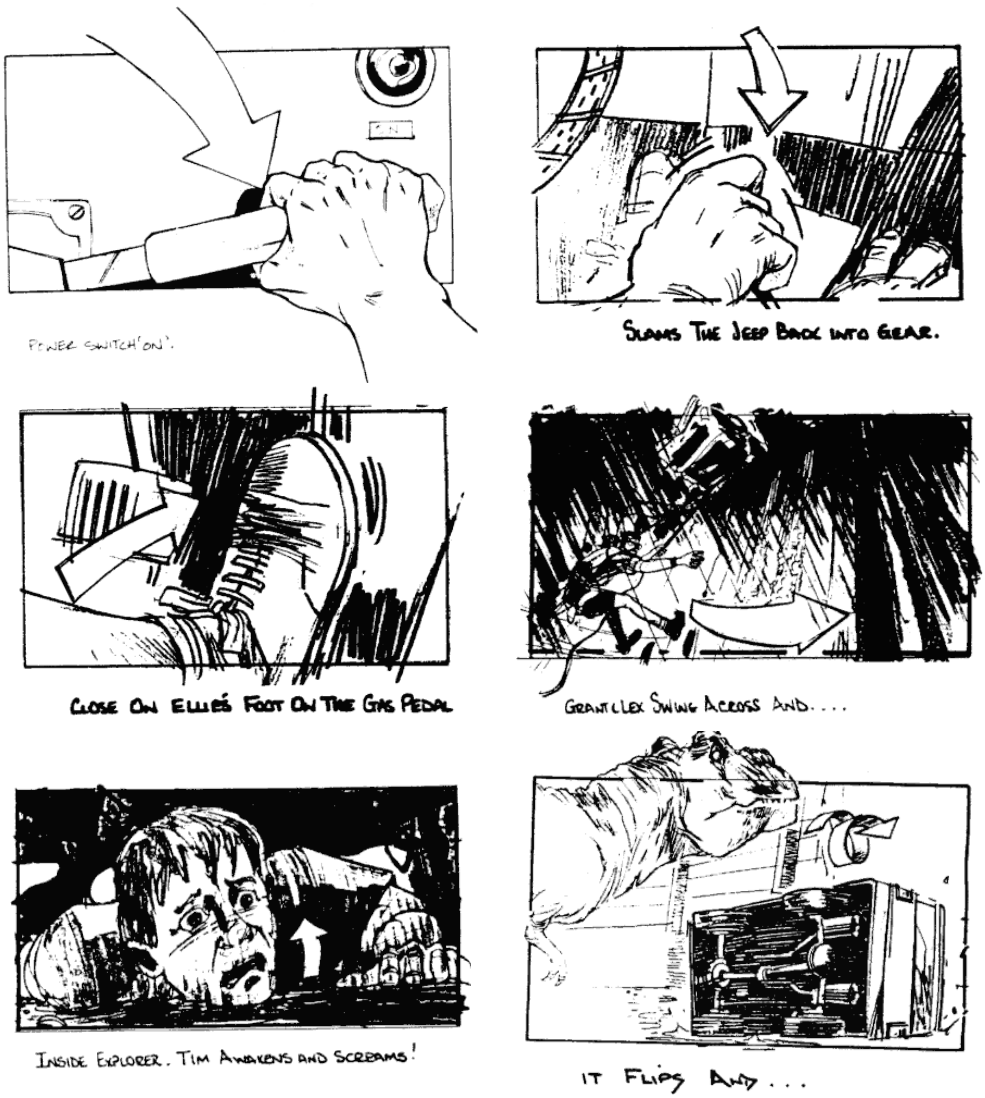


Fig. 6.16: Object motion annotations from the *Jurassic Park* (1993) storyboards [87].

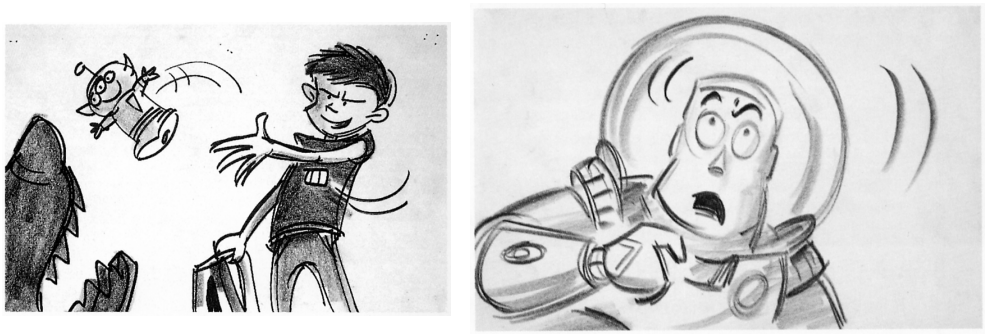


Fig. 6.17: Object motion annotated with speedlines and repeated contours, from the storyboards from *Toy Story* (1995) [49].

The lines are additionally clipped so that they do not actually touch the object, but rather begin slightly behind it; the lower left frame of figure 6.18 shows the resulting effect.

Repeated contours can also be produced from the Bézier outline, as shown in the lower right frame of figure 6.18. A portion of the contour is extracted, using only segments where the normal of the curve points in the direction of the paths of the tracked control points (within a given tolerance). This extracted segment is then repeated at intervals along the direction of the control point's path.

Both these techniques are useful for indicating object motion in a more subtle way than annotating the object with arrows; they do not attract attention, yet effectively communicate the direction of motion. One drawback is that they both, speedlines in particular, can convey a false impression of speed. In the example in figure 6.18 the car is in fact moving quite slowly whereas the speedlines give the impression that it is moving at high velocity.

6.4 Conclusions

We have presented here a technique for producing a summary of a single shot in a video sequence as a 2-dimensional frame, containing a representation of the contents of the shot and annotated with the principal object and camera movements. By applying the technique to each shot in a larger sequence a storyboard-style summary of the sequence can be produced.

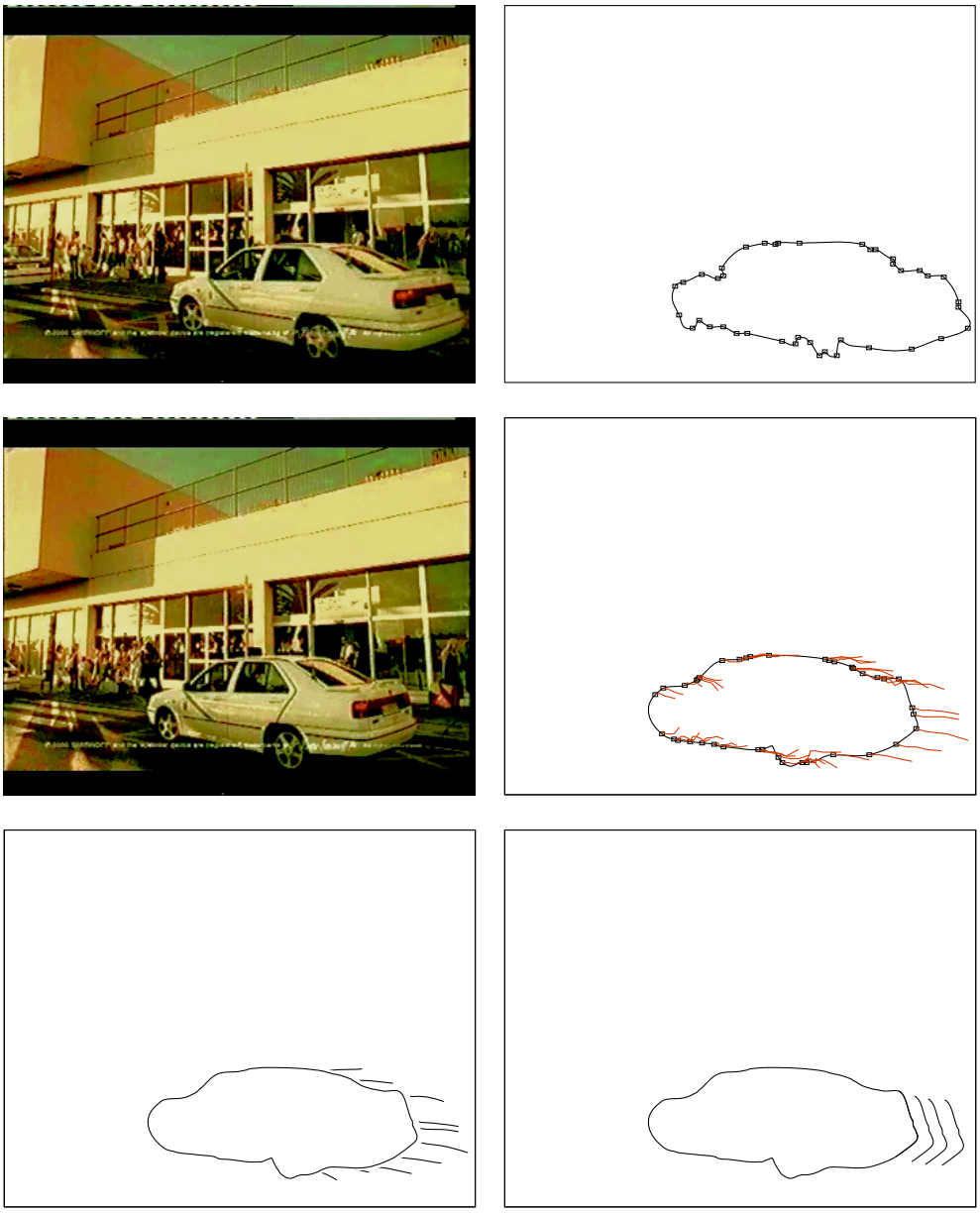


Fig. 6.18: Using speedlines and repeated contours to indicate the motion of an extracted contour.

Fig. 6.19: Final renderings with speedlines and active contours.

This 2-dimensional summary has several advantageous properties. It can be quickly viewed and thus convey the contents of the sequence in much less time than would be required to view the sequence itself. As it is 2-dimensional and static it can be viewed on a screen or printed on paper, so requiring no extra software or equipment. The task of semantic understanding of the scene is offloaded onto the user, who is far better equipped for the task than the computer.

That the output is in vector form rather than bitmapped also has advantages. The frames can be manipulated and transformed without loss, and additional information can be encoded into the data, such as by storing the different segmented objects in different layers. A considerable amount of compression is also realised. At their original resolution (352×288) the frames here require 300 kilobytes of storage, or roughly 70 kilobytes if JPEG compressed. The vector summary of the frame shown in figure 6.19 requires 60 kilobytes as an Acrobat PDF file, or only 17 kilobytes if stored in the compressed SVG format.

There are also drawbacks. The successful annotation of objects and their motion requires a fairly reliable segmentation as input; better than is currently produced by the segmentation scheme described in chapter 5. Reasonably simple object or camera motions can be represented using these annotations, but they are not suitable for complex motion. Unfortunately a large amount of material encountered in our chosen domain, post-production, contains such complex camera and object motion. The motion of people in particular is complex and difficult to represent.

The rendered summary frames have an interesting visual style. The look is somewhat cartoon-like, due to the outlining of detail and the use of speedlines. It is also reminiscent of the style of Richard Linklater's film *Waking Life* (2001), in which live action footage was rotoscoped, redrawn and recoloured (figure 6.20). *Waking Life* required a large amount of manual effort—roughly 250 hours of animator time for each minute on screen [14]. This suggests that the techniques introduced here may have an application in the production of such a style of animation.

DeCarlo and Santella [21] have produced similar looking images using a different technique. They produce a stylised image from a single photograph, also using colour segmentation and edge detection to generate a vector representation. The level of detail is varied across the image to highlight the areas of visual interest; these are identified by tracking the

Fig. 6.20: Scenes from *Waking Life* (2001).

eye movements of a human observer as they look at the image.

The technique could be extended to produce *animatics*; simple animations describing the shots. By using a vector animation format such as Macromedia's *Flash* the scene could be animated with the object and camera motions and stored in considerably less space than the original video would require.

7. Conclusions

Content based image and video analysis is a rapidly developing field that has many exciting applications. As the capacity of computers to store and process large amounts of video material increase we will continue to see large developments in this field. Digital storage and transmission of video are already standard practice and it is only a matter of time until content based indexing and analysis become part of the standard toolset used to manage digital video.

Despite the many recent advances in image analysis, automatic semantic understanding of images and video is still beyond our reach. Indexing and query systems still have a low level pixel-based approach to visual data, and although these systems can still be very useful in many circumstances there is still a large subset of image retrieval tasks for which they are not suitable.

In parallel to research into automatic indexing, querying and classification there has also been ongoing work on processing visual data to aid human understanding of it. The focus of this work has not been to give the computer a complete understanding of visual material, but to process it in ways so that a human user can manage much larger amounts of data. This leaves the burden of semantic understanding with the user, who is far better equipped for the task than the computer.

In this work we have investigated what techniques can assist a specific group of users—people working in video post-production. Having decided that current indexing and query based systems are not sufficiently sophisticated to handle many of their tasks we have instead focused on using video summarisation to aid users in common tasks; in short letting them deal with large amounts of video without actually having to watch all of it.

To this end we have developed a technique for generating a 2-dimensional summary of a video clip, showing the sequence of shots in the clip and the principal objects and motions of each shot, including the apparent camera motion. This summary is presented in the form of an annotated storyboard, which will be familiar to anyone who has worked with storyboards before, or even to anyone who has read a comic book.

This summary can be quickly viewed, either on screen or on paper, and understood by a human user in a fraction of the time it would take to view the original clip. In this way the summary can assist the process of searching for material, either allowing a search to be carried out quicker, or allowing more clips to be reviewed in the same length of time.

This is a way in which computer analysis of stored video material can be of practical assistance in the management of large online collections of material. Other approaches, such as those proposed for content based image retrieval, are hindered by putting the burden of semantic understanding of content on the computer, which is poorly equipped for the task. Dealing with such systems can be frustrating for users when the computer doesn't appear to understand the task. Summarisation, on the other hand, leaves such high-level processing to the user and instead assists them in this task by allowing them to process material faster.

7.1 Contribution

Our technique is a novel approach to video summarisation. Although several storyboard type summarisation techniques have been presented before, using keyframes selected from shots, or panoramas generated from moving cameras, ours is unique in using the visual language of the artist-drawn storyboard to convey information.

The research of the individual parts of the system also resulted in smaller contributions in different areas. The evaluation of temporal segmentation algorithms in Chapter 4 spurred the development of new algorithms for the detection of cuts and fades in highly dynamic material. These algorithms show improved performance over existing techniques on dynamic material while retaining good performance on more conventional material and runtime efficiency by working directly on compressed video streams.

Our work on motion segmentation (Chapter 5) used an existing framework, the COST 211 Analysis Module, and concentrated on improving its performance by manipulating its components. By using different algorithms for motion estimation and segmentation we improve the performance over the reference implementation. Our implementation is better at differentiating the principal moving objects of the scene from noise, and more consistent at tracking objects from frame to frame.

In rendering the extracted data into the storyboard form (Chapter 6) we draw on a number of disparate techniques from graphics and computer vision. Annotations such as speedlines have been previously used in rendering synthetic scenes where motion data is readily available, while we use them to present real scenes. The rendering of the storyboards can also be seen as a form of non-photorealistic rendering, where the computer simulates the style of a human artist.

7.2 Evaluation

The final images produced are very satisfactory, concisely representing a shot in a single frame. However generating these images requires a significant amount of manual intervention. Unfortunately although the system works in principal, the individual components are not robust enough for automatic operation on entire sequences.

Although the temporal segmentation algorithm is an improvement on previous techniques it can only achieve perfect recall and precision on a small number of examples; however its overall performance is good. More limiting is that it currently only detects two types of transitions: cuts and fades. Commercials and music videos can contain a dazzling range of transitions. While some complex edits are well beyond automatic detection, dissolves and wipes should be achievable, and our system doesn't detect these.

The major weakness of the whole system is the poor performance of the motion segmentation step, which only produces acceptable results in a small minority of cases; this prevents the practical use of the system at present. The general framework of the motion segmentation stage is only capable of identifying relatively large objects which have coherent motion independent from the background motion. This presents problems with complex objects, such as people, or in scenes with complex background motion such as a crowd. General motion segmentation of non-domain specific video is still an open and unsolved problem, but if an acceptable technique is discovered it can be dropped into the system as a replacement for the current algorithm.

The rendering subsystem is highly dependent on the information provided to it by the preceding stages. It does an acceptable job of rendering scenes where the motions are straight forward, but is not so suitable for

conveying more complex motions. Complex camera moves are difficult both to identify and to represent in a single storyboard frame. Complex movement of articulated objects, such as people, is also difficult to represent in a clear manner.

7.3 Future work

Content based image and video analysis is a large and developing field, and there are many directions for further work.

Further research that would directly benefit this work would be advances in low level image processing, in particular image segmentation and motion estimation. Motion estimation algorithms are highly susceptible to noise, and attempts to reduce the influence of noise by integrating motion over an area result in reduced resolution and less accurate motion vectors. Motion estimation algorithms have difficulty with object edges, where there is a discontinuity in the flow field. Motion estimation is also very computationally intensive. The development of robust, efficient motion estimation techniques would greatly benefit this work and others.

Image segmentation also suffers problems with noise. In many cases boundaries that are obvious to the human eye are not really present at the pixel level due to noise and blurring. The ubiquitous use of lossy compression techniques only increases this problem. The development of segmentation algorithms based on research into the how the human visual system works will be of great use in content based image analysis.

As we have pointed out, we have not yet reached a point where we can say that computers can represent visual information at a semantic level. There is a huge amount of work still to be done in building computer representations of visual data that mirror out own understanding, but the potential payoffs of such work are huge.

More steps also have to be taken in applying content based image analysis to practical problems. Many research systems are of little use outside the lab, yet there are many practical situations where currently available techniques could be applied to greatly aid users in visual data management tasks.

On the presentation level we believe this work highlights an interesting direction in non-photorealistic rendering. Most work in this field has

concentrated on trying to recreate the physical artifacts of human art; the effect of brush strokes on canvas or pencil on paper. There has been less work on emulating the way a human artist chooses to represent images; what is left in, what is left out, and what is modified in the transition between the physical world and the artist's depiction. The cartoon style of drawing emphasises this as the physical marks (lines, areas of constant colour are easy to recreate.

Our summarisation framework has potential uses beyond producing 2-dimensional storyboards. As mentioned in chapter 6 other representations of the original video clip could be produced, such as vector format animatics. The same breakdown of a sequence could be used to present it in different formats; as a static 2-dimensional storyboard, as an animated storyboard, or as an animatic. Information gathered during the analysis process, such as object characteristics and camera movements could be indexed and used to support a query or browsing interface.

Appendix

A. Pseudocode

A.1 Cut detector

For each frame pair $f-1, f$

For each interior block i

Compute differences $\Delta E_i = |E_{f,i} - E_{f-1,i}|$, $\Delta Y_i = |Y_{f,i} - Y_{f-1,i}|$,
$\Delta UV_i = |U_{f,i} - U_{f-1,i}| - |V_{f,i} - V_{f-1,i}|$

Compute cross-correlation $C_{f,i}$

If $E_{f-1,i} < T_{uniform}$ **and** $E_{f,i} < T_{uniform}$

If $\Delta Y_i > T_Y$ **or** $\Delta UV_i > T_{UV1}$

$C_{f,i} = 0$

else

$C_{f,i} = 1$

else if $E_{f,i} < T_{uniform}$ **and** $E_{f-1,i} > T_{uniform}$ **and** $\Delta E_i > T_E$

$C_{f,i} = 0$

Mark block as type 2

else if $E_{f-1,i} < T_{uniform}$ **and** $E_{f,i} > T_{uniform}$ **and** $\Delta E_i > T_E$

$C_{f,i} = 0$

Mark block as type 2

Compute mean and standard deviations of past behavious
of C_i and ΔUV_i: $\overline{C_i}, \sigma_{C_i}, \overline{\Delta UV_i}, \sigma_{\Delta UV_i}$

If $C_{f,i} < 0.8$ **and** $|\Delta UV_i - \overline{\Delta UV_i}| > 2\sigma_{\Delta UV_i}$ **and** $\Delta UV_i > T_{UV2}$

Mark block as type 2

else if $C_{f,i} >= 0.3$ **and** $\sigma_{C_i} < 0.25$ **and** $|C_{f,i} - \overline{C_i}| < 1.5\sigma_{C_i}$

Mark block as type 1

else if $C_{f,i} - \overline{C_i} < -0.5$

Mark block as type 2

If ($C_{f,i} > 0.5$ **and** $C_{f-1,i} < 0.3$) **or** $C_{f,i} - C_{f-1,i} > 0.3$

Mark block as type 3

Compute counts of type 1, 2, and 3 blocks: $n_{f,1}, n_{f,2}, n_{f,3}$

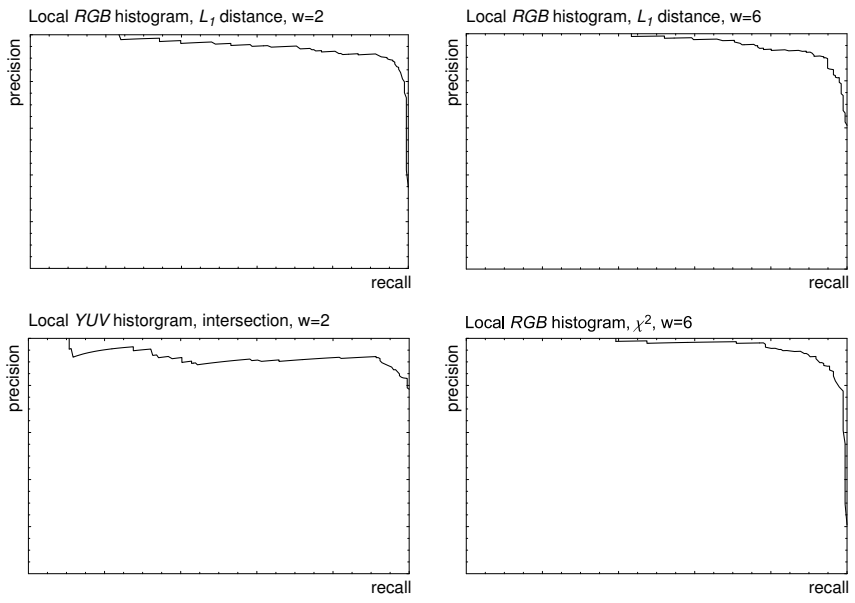
If previous frame marked as potential cut
$a = (n_{f-1,3} + n_{f-2,3})/2$
$b = (n_{f,3} + n_{f-1,2})/2$
If $b > 1/4$ of blocks **and** $n_{f,3} > a$ **and** $n_{f,3} > n_{f,1}$
Frame f is a cut

Compute mean and standard deviation of n_2: $\overline{n_2}, \sigma_{n_2}$
If $n_{f,2} > n_{f,1} + n_{f,2}$ **and** $|n_{f,2} - \overline{n_2}| > \sigma_{n_2}$ **and** $n_{f,2} > 5/4 \times n_{f-1,2}$
Mark frame as potential cut

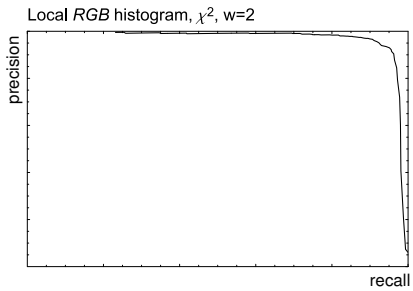
The thresholds we use are: $T_{uniform} = 100, T_Y = 48, T_{UV1} = 24, T_E = 500$, and $T_{UV2} = 12$.

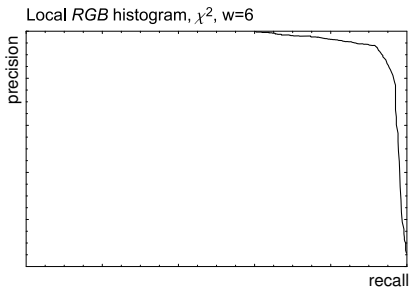
B. Evaluation of cut detection algorithms

			Recall @ 90% precision w=2	w=6	Precision @ 90% recall w=2	w=6
Global	RGB	L1	0.874	0.883	0.885	0.898
		Chi	0.892	0.920	0.890	0.917
		Intersection	0.874	0.883	0.885	0.898
		K-s	0.836	0.829	0.831	0.851
	YUV	L1	0.922	0.888	0.907	0.876
		Chi	0.883	0.898	0.878	0.875
		Intersection	0.922	0.888	0.907	0.876
		K-s	0.827	0.855	0.871	0.849
	Luv	L1	0.750	0.786	0.816	0.832
		Chi	0.770	0.847	0.862	0.805
		Intersection	0.750	0.786	0.816	0.832
		K-s	0.689	0.762	0.691	0.682
	Y	L1	0.850	0.824	0.838	0.833
		Chi	0.838	0.841	0.833	0.839
		Intersection	0.850	0.824	0.838	0.833
		K-s	0.676	0.781	0.707	0.725
Local	RGB	L1	0.944	0.939	0.917	0.922
		Chi	0.915	0.926	0.910	0.927
		Intersection	0.913	0.923	0.906	0.904
		K-s	0.883	0.872	0.890	0.872
	YUV	L1	0.918	0.862	0.915	0.889
		Chi	0.923	0.905	0.907	0.904
		Intersection	0.926	0.898	0.922	0.883
		K-s	0.918	0.878	0.904	0.889
	Luv	L1	0.824	0.804	0.880	0.859
		Chi	0.886	0.865	0.894	0.868
		Intersection	0.804	0.778	0.880	0.851
		K-s	0.746	0.811	0.874	0.848
	Y	L1	0.752	0.763	0.864	0.842
		Chi	0.819	0.856	0.850	0.870
		Intersection	0.717	0.804	0.874	0.864
		K-s	0.740	0.752	0.872	0.846
Pixel difference			0.776	0.689	0.869	0.835
pixel inner product			0.000	0.000	0.041	0.040
Ftest			0.597	0.660	0.162	0.177
Ford 1			0.133	0.149	0.211	0.233
Ford 2			0.597	0.625	0.225	0.252
Zabih			0.528	0.235	0.610	0.515
DCT inner product			0.000	0.000	0.209	0.135

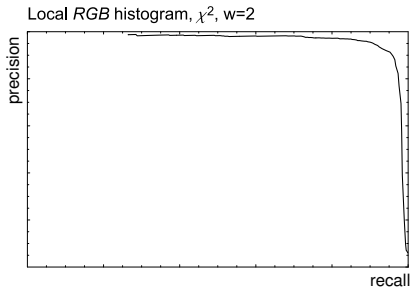


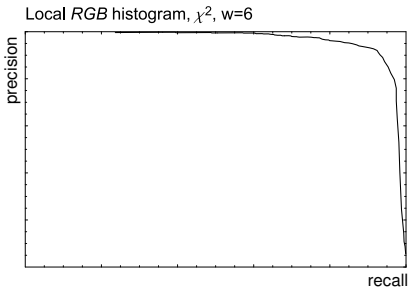
			Recall @ 90% precision		Precision @ 90% recall	
			w=2	w=6	w=2	w=6
Global	RGB	L1	0.914	0.873	0.924	0.871
		Chi	0.918	0.886	0.935	0.893
		Intersection	0.914	0.873	0.924	0.871
		K-s	0.874	0.835	0.837	0.769
	YUV	L1	0.916	0.859	0.917	0.850
		Chi	0.885	0.858	0.887	0.836
		Intersection	0.916	0.859	0.917	0.850
		K-s	0.837	0.796	0.791	0.778
	Luv	L1	0.822	0.796	0.775	0.747
		Chi	0.834	0.778	0.807	0.760
		Intersection	0.822	0.796	0.775	0.747
		K-s	0.785	0.762	0.652	0.653
	Y	L1	0.895	0.799	0.895	0.834
		Chi	0.895	0.844	0.895	0.842
		Intersection	0.895	0.799	0.895	0.834
		K-s	0.856	0.791	0.782	0.749
Local	RGB	L1	0.957	0.919	0.960	0.931
		Chi	0.963	0.935	0.968	0.942
		Intersection	0.958	0.924	0.959	0.927
		K-s	0.947	0.921	0.955	0.922
	YUV	L1	0.955	0.901	0.958	0.901
		Chi	0.952	0.923	0.958	0.921
		Intersection	0.954	0.896	0.959	0.896
		K-s	0.943	0.898	0.948	0.898
	Luv	L1	0.926	0.891	0.932	0.892
		Chi	0.942	0.893	0.946	0.893
		Intersection	0.935	0.894	0.922	0.894
		K-s	0.925	0.887	0.926	0.891
	Y	L1	0.930	0.890	0.935	0.896
		Chi	0.928	0.901	0.947	0.902
		Intersection	0.939	0.904	0.941	0.902
		K-s	0.931	0.902	0.935	0.902
Pixel difference			0.963	0.929	0.943	0.915
pixel inner product			0.000	0.000	0.049	0.048
Ftest			0.588	0.637	0.159	0.219
Ford 1			0.089	0.125	0.236	0.249
Ford 2			0.698	0.689	0.222	0.222
Zabih			0.736	0.656	0.808	0.799
DCT inner product			0.000	0.000	0.092	0.093



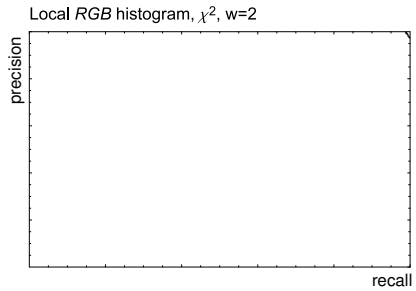


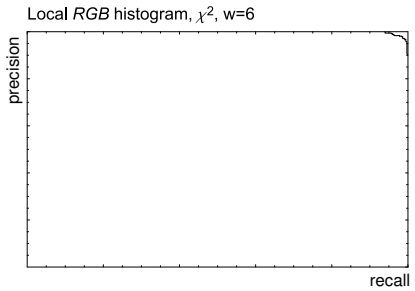
			Recall @ 90% precision		Precision @ 90% recall	
			w=2	w=6	w=2	w=6
Global	RGB	L1	0.919	0.881	0.921	0.882
		Chi	0.916	0.890	0.924	0.892
		Intersection	0.919	0.881	0.921	0.882
		K-s	0.875	0.838	0.847	0.814
	YUV	L1	0.913	0.859	0.912	0.856
		Chi	0.885	0.861	0.880	0.842
		Intersection	0.913	0.859	0.912	0.856
		K-s	0.845	0.806	0.797	0.799
	Luv	L1	0.824	0.799	0.787	0.770
		Chi	0.842	0.804	0.805	0.771
		Intersection	0.824	0.799	0.787	0.770
		K-s	0.786	0.761	0.635	0.671
	Y	L1	0.877	0.800	0.881	0.833
		Chi	0.875	0.842	0.873	0.832
		Intersection	0.877	0.800	0.881	0.833
		K-s	0.832	0.788	0.776	0.770
Local	RGB	L1	0.954	0.920	0.950	0.927
		Chi	0.959	0.933	0.956	0.933
		Intersection	0.954	0.916	0.949	0.922
		K-s	0.940	0.908	0.943	0.909
	YUV	L1	0.951	0.903	0.947	0.904
		Chi	0.950	0.920	0.947	0.917
		Intersection	0.950	0.897	0.950	0.896
		K-s	0.944	0.890	0.937	0.891
	Luv	L1	0.927	0.879	0.922	0.886
		Chi	0.934	0.886	0.931	0.889
		Intersection	0.924	0.876	0.916	0.887
		K-s	0.922	0.865	0.916	0.883
	Y	L1	0.920	0.875	0.915	0.883
		Chi	0.919	0.891	0.928	0.893
		Intersection	0.930	0.888	0.926	0.892
		K-s	0.922	0.867	0.919	0.886
Pixel difference			0.945	0.894	0.925	0.896
pixel inner product			0.000	0.000	0.047	0.047
Ftest			0.597	0.638	0.158	0.215
Ford 1			0.094	0.129	0.231	0.249
Ford 2			0.694	0.677	0.225	0.245
Zabih			0.668	0.585	0.766	0.742
DCT inner product			0.000	0.000	0.085	0.086





			Recall @ 90% precision		Precision @ 90% recall	
			w=2	w=6	w=2	w=6
Global	RGB	L1	0.997	0.989	1.000	0.985
		Chi	1.000	0.997	1.000	1.000
		Intersection	0.997	0.989	1.000	0.985
		K-s	0.975	0.962	1.000	0.951
	YUV	L1	0.995	0.989	0.997	0.979
		Chi	0.997	1.000	1.000	1.000
		Intersection	0.995	0.989	0.997	0.979
		K-s	0.951	0.946	0.985	0.932
	Luv	L1	0.959	0.946	0.985	0.968
		Chi	0.981	0.978	0.991	0.988
		Intersection	0.959	0.946	0.985	0.968
		K-s	0.926	0.917	0.971	0.932
	Y	L1	0.997	0.989	0.997	0.979
		Chi	1.000	0.997	1.000	0.997
		Intersection	0.997	0.989	0.997	0.979
		K-s	0.973	0.954	0.997	0.951
Local	RGB	L1	1.000	0.997	1.000	0.994
		Chi	1.000	1.000	1.000	1.000
		Intersection	1.000	0.997	1.000	0.994
		K-s	1.000	0.995	1.000	0.997
	YUV	L1	1.000	0.997	1.000	0.988
		Chi	1.000	1.000	1.000	1.000
		Intersection	1.000	0.997	1.000	0.988
		K-s	1.000	1.000	1.000	0.988
	Luv	L1	0.995	0.992	1.000	0.994
		Chi	0.995	0.995	1.000	1.000
		Intersection	0.997	0.995	1.000	0.994
		K-s	0.995	0.990	1.000	0.997
	Y	L1	0.996	0.989	0.991	0.979
		Chi	0.997	1.000	0.991	0.994
		Intersection	0.995	0.994	0.982	0.988
		K-s	0.995	0.986	0.991	0.979
Pixel difference			0.995	0.981	0.988	0.971
pixel inner product			0.000	0.000	0.053	0.015
Ftest			0.744	0.777	0.151	0.221
Ford 1			0.203	0.295	0.300	0.361
Ford 2			0.723	0.765	0.173	0.203
Zabih			0.880	0.828	0.882	0.819
DCT inner product			0.000	0.000	0.225	0.127

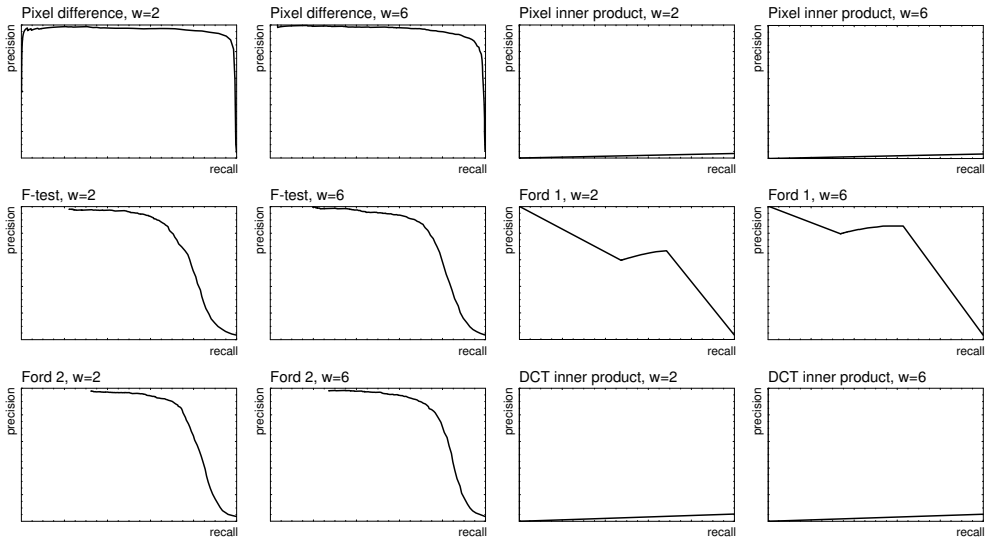




			Recall @ 90% precision		Precision @ 90% recall	
			w=2	w=6	w=2	w=6
Global	RGB	L1	0.940	0.907	0.949	0.907
		Chi	0.940	0.906	0.954	0.909
		Intersection	0.940	0.907	0.949	0.907
		K-s	0.893	0.863	0.885	0.849
	YUV	L1	0.939	0.891	0.946	0.890
		Chi	0.921	0.894	0.936	0.893
		Intersection	0.939	0.891	0.946	0.890
		K-s	0.871	0.838	0.848	0.832
	Luv	L1	0.858	0.834	0.806	0.802
		Chi	0.885	0.839	0.874	0.806
		Intersection	0.857	0.834	0.806	0.802
		K-s	0.821	0.798	0.633	0.707
	Y	L1	0.920	0.850	0.925	0.863
		Chi	0.915	0.878	0.923	0.869
		Intersection	0.920	0.850	0.925	0.863
		K-s	0.876	0.827	0.830	0.796
Local	RGB	L1	0.969	0.940	0.968	0.941
		Chi	0.972	0.947	0.971	0.948
		Intersection	0.967	0.938	0.964	0.946
		K-s	0.958	0.929	0.962	0.929
	YUV	L1	0.969	0.935	0.964	0.927
		Chi	0.968	0.946	0.965	0.938
		Intersection	0.969	0.937	0.969	0.930
		K-s	0.961	0.923	0.958	0.922
	Luv	L1	0.947	0.914	0.945	0.912
		Chi	0.956	0.916	0.956	0.918
		Intersection	0.954	0.921	0.947	0.913
		K-s	0.944	0.908	0.944	0.906
	Y	L1	0.945	0.902	0.945	0.902
		Chi	0.940	0.909	0.949	0.909
		Intersection	0.949	0.914	0.949	0.912
		K-s	0.943	0.895	0.946	0.897
Pixel difference			0.963	0.925	0.940	0.912
pixel inner product			0.000	0.000	0.031	0.031
Ftest			0.630	0.673	0.123	0.154
Ford 1			0.119	0.168	0.234	0.253
Ford 2			0.702	0.697	0.169	0.177
Zabih			0.701	0.628	0.741	0.679
DCT inner product			0.000	0.000	0.049	0.048

Tab. B.1: Precision/recall graphs for the whole test set.

Tab. B.2: Precision/recall graphs for the whole test set (*cont.*).



Tab. B.3: Precision/recall graphs for the whole test set (*cont.*).

Bibliography

[1] Smoke & Mirrors Ltd., http://www.smoke-mirrors.com/.

[2] Unique-ID Ltd., http://www.unique-id.com/.

[3] Discreet Logic. http://www.discreet.com/.

[4] *The Guardian*. 20 May, 2002.

[5] *The Observer*. 3 June, 2001.

[6] The European COST 211 Group. http://www.iva.cs.tut.fi/COST211/.

[7] Independent JPEG Group. http://www.ijg.org/.

[8] Intel Open Source Computer Vision Llibrary. http://developer.intel.com/software/products/opensource/libraries/cvfl.%htm.

[9] Til Aach and André Kaup. Bayesian algorithms for adaptive change detection in image sequences using markov random fields. *Signal Processing: Image Communication*, 7(2):147–160, 1995.

[10] Akihito Akutsu and Yoshinobu Tonomura. Video tomography: An efficient method for camerawork extraction and motion analysis. In *ACM Multimedia '94*, pages 349–356, 1994.

[11] A. Aydin Alatan, Levent Onural, Michael Wollborn, Roland Mech, Ertem Tuncel, and Thomas Sikora. Image sequence analysis for emerging interactive multimedia services—the European COST 211 framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):802–813, November 1998.

[12] F. Arman, R. Depommier, A. Hsu, and M.-Y. Chiu. Content-based browsing of video sequences. In *ACM Multimedia '94*, pages 97–103, 1994.

[13] Farshid Arman, Arding Hsu, and Ming-Yee Chiu. Image processing on encoded video sequences. *Multimedia systems*, 1(5):211–219, 1994.

[14] Richard Baimbridge. Unreal world. *Wired*, 9(2), February 2001.

[15] M. Bierling. Displacement estimation by hierarchical blockmatching. In *Visual Communications and Image Processing '88*, volume 1001 of *Proceedings of SPIE*, pages 942–951. SPIE, 1988.

[16] David Bordwell and Kristin Thompson. *Film art: An introduction*. McGraw-Hill, 1990.

[17] John S. Boreczky and Lawrence A. Rowe. Comparison of shot boundary detection techniques. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Still Image and Video Databases IV*, volume 2670 of *Proceedings of SPIE*, pages 170–179, 1996.

[18] John S. Boreczky and Lynn D. Wilcox. A hidden markov model framework for video segmentation using audio and image features. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 1998*, pages 3741–3744. IEEE, 1998.

[19] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference on Visual Information Systems*. Springer-Verlag, 1999.

[20] Scott Craver, Boon-Lock Yeo, and Minerva Yeung. Multilinearization data structure for image browsing. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 155–166. SPIE, 1999.

[21] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM Transactions on Graphics*, 21(3):769–776, July 2002. Proceedings of Siggraph '02.

[22] Yining Deng, B. S. Manjunath, and Hyundoo Shin. Color image segmentation. In *IEEE Computer Society Conference on Computer*

Vision and Pattern Recognition, volume 2, pages 446–451. IEEE, 1999.

[23] Ajay Divakaran, Kadir A. Peker, and Huifang Sun. Video summarization using motion descriptors. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retrieval for Media Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 517–522, 2001.

[24] Jean-Luc Dugelay and Henri Sanson. Differential methods for the identification of 2D and 3D motion models in image sequences. *Signal Processing: Image Communication*, 7(1):105–127, March 1995.

[25] Dennis Dunn and William E. Higgins. Optimal gabor filters for texture segmentation. *IEEE Transactions on Image Processing*, 4(7):947–964, July 1995.

[26] Tomio Echigo and Shun-ichi Iisaku. Unsupervised segmentation of colored texture images by using multiple GMRF models and a hypothesis of merging primitives. *Systems and Computers in Japan*, 31(2):29–39, 2000.

[27] A. Müfit Ferman and A. Murat Tekalp. Editing cues for content-based analysis and summarization of motion pictures. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 71–80, 1998.

[28] W. A. C. Fernando, C. N. Canagarajah, and D. R. Bull. Video segmentation and classification for content based storage and retrieval using motion vectors. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 687–698, 1999.

[29] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, September 1995.

[30] Ralph M. Ford. Fuzzy-logic approach to digital video segmentation. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 360–370, 1998.

[31] Ralph M. Ford. A quantitative comparison of shot boundary detection metrics. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 666–676, 1999.

[32] D. A. Forsyth. Benchmarks for storage and retrieval in multimedia databases. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retrieval for Media Databases 2002*, volume 4676 of *Proceedings of SPIE*, pages 240–247, 2002.

[33] Ullas Gargi, Rangachar Kasturi, and Susan H. Strayer. Performance characterization of video-shot-change detection methods. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(1):1–13, February 2000.

[34] Luis Garrido, Philippe Salembier, and David Garcia. Extensive operators in partition lattices for image sequence analysis. *Signal Processing*, 66(2):157–180, April 1998.

[35] Arun Hampapur, Ramesh Jain, and Terry Weymouth. Digital video segmentation. In *ACM Multimedia '94*, pages 357–364, 1994.

[36] Seung Hoon Han and In So Kweon. Shot detection combining bayesian and structural information. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retrieval for Media Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 509–516, 2001.

[37] John Hart. *The Art of the Storyboard*. Focal Press, 1999.

[38] Stephan Herrmann, Hubert Mooshofer, Harald Dietrich, and Walter Stechele. A video segmentation algorithm for hierarchical object representations and its implementation. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1204–1214, December 1999.

[39] B. K. P. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, August 1981.

[40] Michael Hötter and Robert Thoma. Image segmentation based on object oriented mapping parameter estimation. *Signal Processing*, 15:315–334, 1988.

[41] Michal Irani, Benny Rousso, and Shmuel Peleg. Computing occluding and transparent motions. *International Journal of Computer Vision*, 12(1):5–16, January 1994.

[42] Bruce F. Kawin. *How Movies Work*. University of California Press, 1992.

[43] Changick Kim and Jenq-Neng Hwang. An integrated scheme for object-based video abstraction. In *ACM Multimedia '00*, pages 303–311, 2000.

[44] Hyeokman Kim, Jinho Lee, Jae Heon Yang, Sanghoon Sull, W. M. Kim, and S. Moon-Ho Song. Visual rhythm and shot verification. *Multimedia Tools and Applications*, 15(3):227–245, December 2001.

[45] Vikrant Kobla, Daniel DeMenthon, and David Doermann. Special effect edit detection using videotrails: A comparison with existing techniques. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 302–313, 1999.

[46] Vikrant Kobla, David Doermann, and King-Ip (David) Lin. Archiving, indexing, and retrieval of video in the compressed domain. In *Proc. of the SPIE Conference on Multimedia Storage and Archiving Systems*, volume 2916 of *Proceedings of SPIE*, pages 78–89. SPIE, 1996.

[47] Janusz Konrad and Eric Dubois. Bayesian estimation of motion vector fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):910–927, September 1992.

[48] Peter M. Kuhn, Georg Diebel, Stephan Herrmann, Andreas Keil, Hubert Mooshofer, Andreé Kaup, Robert Mayer, and Walter

Stechele. Complexity and PSNR-comparison of several fast motion estimation algorithms for MPEG-4. In *Applications of Digital Image Processing XXI*, volume 3460 of *Proceedings of SPIE*, pages 486–499, 1998.

[49] John Lasseter and Steve Daly. *Toy Story: The Art and Making of the Animated Film*. Hyperion, 1995.

[50] Seong-Whan Lee, Young-Min Kim, and Sung Woo Choi. Fast scene change detection using direct feature extraction from MPEG compressed videos. *IEEE Transactions on Multimedia*, 2(4):240–254, December 2000.

[51] Rainer Lienhart. Comparison of automatic shot boundary detection algorithms. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 290–301, 1999.

[52] Rainer Lienhart. Reliable dissolve detection. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retieval for Multimedia Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 219–230. SPIE, 2001.

[53] Rainer Lienhart, Silvia Pfeiffer, and Wolfgang Effelsberg. Video abstracting. *Communications of the ACM*, 40(12):55–62, December 1997.

[54] Yi Lin, Mohan S. Kankanhalli, and Tat-Seng Chua. Temporal multi-resolution analysis for video segmentation. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Media Databases 2000*, volume 3972 of *Proceedings of SPIE*, pages 494–505, 2000.

[55] Joan Llach and Philippe Salembier. Analysis of video sequences: Table of contents and index creation. In *International Workshop on Very Low Bitrate Video (VLBV '99)*, October 1999. Kyoto.

[56] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *DARPA Image Understanding Workshop*, pages 121–130, 1981.

[57] W. Y. Ma and B. S. Manjunath. Edge flow: A framework of boundary detection and image segmentation. *IEEE Transactions on Image Processing*, 9(8):1375–1388, August 2000.

[58] Peter J. Macer, Peter J. Thomas, Nouhman Chalabi, and John F. Meech. Finding the cut of the wrong trousers: Fast video search using automatic storyboard generation. In *ACM Multimedia '96*, pages 303–304. ACM, 1996.

[59] Maic Masuch. Speedlines: Depicting motion in motionless pictures. In *Siggraph '99 Sketches and Applications*, page 277, 1999.

[60] T. McGee and N. Dimitrova. Parsing tv programs for identification and removal of non-story segments. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Image and Video Databases VII*, volume 3656 of *Proceedings of SPIE*, pages 243–251. SPIE, 1999.

[61] Roland Mech and Michael Wollborn. A noise robust method for 2D shape estimation of moving objects in video sequences considering a moving camera. *Signal Processing*, 66(2):203–217, 1998.

[62] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. Robust and efficient shape indexing through curvature scale space. In *Proc. British Machine Vision Conference*, pages 53–62, 1996.

[63] Jeho Nam and Ahmed H. Tewfik. Combined audio and visual streams analysis for video sequence segmentation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 1997*, 1997.

[64] Jeho Nam and Ahmed H. Tewfik. Dynamic video summarization and visualization. In *Multimedia '99*, pages 53–56, 1999.

[65] M. R. Naphade, R. Mehrotra, A. M. Ferman, J. Warnick, T. S. Huang, and A. M. Tekalp. A high-performance shot boundary detection algorithm using multiple cues. In *IEEE International Conference on Image Processing (ICIP)*, pages 884–887. IEEE, 1998.

[66] Chahab Nastar, Matthias Mitschke, Christophe Meilhac, and Nozha Boujemaa. Surfimage: a flexible content-based image retrieval system. In *ACM Multimedia '98*, pages 339–344, 1998.

[67] A. Neri, S. Colonnese, G. Russo, and P. Talone. Automatic moving object and background separation. *Signal Processing*, 66(2):219–232, 1998.

[68] Chong Wah Ngo. *Analysis of spatio-temporal slices for video content representation*. PhD thesis, Hong Kong University of Science and Technology, 2000.

[69] Chong-Wah Ngo, Ting-Chuen Pong, and Hong-Jiang Zhang. On clustering and retrieval of video shots. In *ACM Multimedia '01*, pages 51–59, 2001.

[70] Hieu T. Nguyen, Marcel Worring, and Anuj Dev. Robust motion based segmentation in video sequences. Technical Report 4, Intelligent Sensory Information Systems, University of Amsterdam, 1998.

[71] Jean-Marc Odobez and Patrick Bouthemy. Direct incremental model-based image motion segmentation for video analysis. *Signal Processing*, 66(2):143–155, 1998.

[72] JungHwan Oh and Kien A. Hua. An efficient technique for summarizing videos using visual contents. In *IEEE International Conference on Multimedia and Expo*, pages 1167–1170, 2000.

[73] A. Pentlands, R. W. Picard, and S. Sclaroff. Photobook: content based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, June 1996.

[74] Silvia Pfeiffer, Rainer Lienhart, Stephan Fischer, and Wolfgang Effelsberg. Abstracting digital movies automatically. *Journal of Visual Communication and Image Representation*, 7(4):345–353, December 1996.

[75] Dulce Ponceleon, Savitha Srinivasan, Arnon Amir, Dragutin Petkovic, and Dan Diklic. Key to effective video retrieval: Ef-

fective cataloging and browsing. In *ACM Multimedia '98*, pages 99–107, 1998.

[76] Sarah Porter, Majid Mirmehdi, and Barry Thomas. Detection and classification of shot transitions. In Tim Cootes and Chris Taylor, editors, *British Machine Vision Conference 2001*, pages 73–82, 2001.

[77] Thomas Porter and Tom Duff. Compositing digital images. In *SIGGRAPH 1984*, pages 253–259, 1984.

[78] William H. Press, Saul A. Teukolsky, William T Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[79] P. Salembier and F. Marqués. Region-based representations of image and video: Segmentation tools for multimedia services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1147–1167, December 1999.

[80] Philippe Salembier and Luis Garrido. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *IEEE Transactions on Image Processing*, 9(4):561–576, April 2000.

[81] Simone Santini and Ramesh Jain. The "El Niño" image database system. In *Multimedia Systems '99: International Conference on Multimedia Computing and Systems*, pages 524–529. IEEE, 1999.

[82] Harpreet S. Sawhney and Serge Ayer. Compact representations of video through dominant and multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):814–830, August 1996.

[83] Harpreet S. Sawhney, Serge Ayer, and Monika Gorkani. Model-based 2D & 3D dominant motion estimation for mosaicing and video representation. In *Fifth International Conference on Computer Vision*, pages 583–590. IEEE, 1995.

[84] Philip J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988.

[85] Philip J. Schneider. An algorithm for automatically fitting digitized curves. In Andrew S. Glassner, editor, *Graphics Gems*, pages 612–626. Academic Press, 1990.

[86] Ishwar K. Sethi and Nilesh Patel. A statistical approach to scene change detection. In Wayne Niblack and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases III*, volume 2420 of *Proceedings of SPIE*, pages 329–338, 1995.

[87] Don Shay and Jody Duncan. *The Making of Jurassic Park*. Boxtree, 1993.

[88] H.-Y. Shum and R. Szeliski. Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2):101–130, February 2000.

[89] Andreas Siebert. Segmentation based image retrieval. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 14–24, 1998.

[90] John R. Smith. VideoZoom spatio-temporal video browser. *IEEE Transactions on Multimedia*, 1(2):157–171, June 1999.

[91] John R. Smith and Shih-Fu Chang. Visualseek: a fully automated content-based image query system. In *ACM Multimedia '96*, pages 87–98, 1996.

[92] Michael A. Smith and Takeo Kanade. Video skimming for quick browsing based on audio and image characterization. Technical Report CMU-CS-95-186, School of Computer Science, Carnegie Mellon University, 1995.

[93] S. M. Smith and J. M. Brady. Asset-2: Real-time motion segmentation and shape tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):814–820, 1995.

[94] S. M. Smith and J. M. Brady. Susan – a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997.

[95] Stephen M. Smith. Reviews of optic flow, motion segmentation, edge finding and corner finding. Technical Report TR97SMS1, Department of Clinical Neurology, Ofxord University, 1997.

[96] S. Moon-Ho Song, Tae-Hoon Kwon, Woonkyung M. Kim, Hyeokman Kim, and Byung-Do Rhee. On detection of gradual scene changes for parsing of video data. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 404–413, 1998.

[97] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann, 1996.

[98] G. Sudhir and John C. M. Lee. Video annotation by motion interpretation using optical flow streams. *Journal of Visual Communication and Image Representation*, 7(4):354–368, December 1996.

[99] Sanghoon Sull, Jung-Rim Kim, Yunam Kim, Hyun Sung Chang, and Sang Uk Lee. Scalable hierarchical video summary and search. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retrieval for Media Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 553–561, 2001.

[100] M. J. Swain and D. N. Ballard. Color indexing. *Internation Journal of Computer Vision*, 7:11–32, 1991.

[101] Richard Szeliski and James Coughlan. Spline-based image registration. Technical Report CRL 94/1, Digital Equipment Corporation Cambridge Research Lab, 1994.

[102] H. Tamura, S. Mori, and T. Yamawaki. Texture features corresponding to visual perception. *IEEE Transaction on Systems, Man, and Cybernetics*, 8(6):460–473, June 1978.

[103] Yukinobu Taniguchi, Akihito Akutsu, and Yoshinobu Tonomura. PanoramaExcerpts: Extracting and packing panoramas for video browsing. In *ACM Multimedia '97*, pages 427–436. ACM, 1997.

[104] Laura Teodosio and Walter Bender. Salient video stills: Content and context preserved. In *ACM Multimedia '93*, pages 39–46, 1993.

[105] Candemir Toklu and Shih-Ping Liou. Automatic key-frame selection for content-based video indexing and access. In Minerva M. Yeung, Boon-Lock Yeo, and Charles A. Bouman, editors, *Storage and Retrieval for Media Databases 2000*, volume 3972 of *Proceedings of SPIE*, pages 554–563, 2000.

[106] M. S. Toller, P. H. Lewis, and M. S. Nixon. Video segmentation using combined cues. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 414–425, 1998.

[107] Ertem Tuncel and Levent Onural. Utilization of the recursive shortest spanning tree algorithm for video object segmentation by 2-D affine motion modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(5):776–781, August 2000.

[108] Shingo Uchihashi, Jonathan Foote, Andreas Girgensohn, and John Boreczky. Video manga: Generating semantically meaningful video summaries. In *ACM Multimedia '99*, pages 383–392, 1999.

[109] Nuno Vasconcelos and Andrew Lippman. Statistical models of video structure for content analysis and characterization. *IEEE Transactions on Image Processing*, 9(1):3–19, January 2000.

[110] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991.

[111] Jie Wei, Mark S. Drew, and Ze-Nian Li. Illumination-invariant video segmentation by hierarchical robust thresholding. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases VI*, volume 3312 of *Proceedings of SPIE*, pages 188–201, 1998.

[112] Donna J. Williams and Mubarak Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Understanding*, 55(1):14–26, January 1992.

[113] Wei Xiong and John Chung-Mong Lee. Efficient scene change detection and camera motion annotation for video classification. *Computer vision and image understanding*, 71(2):166–181, August 1998.

[114] Li-Qun Xu, Jian Zhu, and Fred Stentiford. Video summarization and semantics editing tools. In Minerva M. Yeung, Chung-Sheng Li, and Rainer W. Lienhart, editors, *Storage and Retrieval for Media Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 242–252. SPIE, 2001.

[115] Boon-Lock Yeo and Bede Liu. Rapid scene analysis on compressed video. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(6):533–544, December 1995.

[116] Boon-Lock Yeo and Minerva M. Yeung. Retrieving and visualizing video. *Communications of the ACM*, 40(12):43–52, December 1997.

[117] Minerva M. Yeung and Boon-Lock Yeo. Video visualization for compact presentation and fast browsing of pictorial content. *IEEE Transactions on Circuits and systems for Video Technology*, 7(5):771–785, October 1997.

[118] Hong Heather Yu and Wayne Wolf. A hierarchical multiresolution video shot transition detection scheme. *Computer Vision and Image Understanding*, 75(1):196–213, July 1999.

[119] Ramin Zabih, Justin Miller, and Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. In *ACM Multimedia '95*, pages 189–200, 1995.

[120] HongJiang Zhang, Atreyi Kankanhalli, and Stephen W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28, 1993.