AUTOMATIC NON-LINEAR VIDEO EDITING FOR HOME VIDEO COLLECTIONS

VAISHNAVI RAJGOPALAN

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE

DEGREE OF MASTERS IN APPLIED SCIENCE (SOFTWARE ENGINEERING)

AT

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

APRIL 2010

Your file Votre référence
ISBN: 978-0-494-67129-0
Our file Notre référence
ISBN: 978-0-494-67129-0

Abstract

Automatic Non-linear Video Editing for Home Video Collections

Vaishnavi Rajgopalan

The video editing process consists of deciding what elements to retain, delete, or combine from various video sources so that they come together in an organized, logical, and visually pleasing manner. Before the digital era, non-linear editing involved the arduous process of physically cutting and splicing video tapes, and was restricted to the movie industry and a few video enthusiasts. Today, when digital cameras and camcorders have made large personal video collections commonplace, non-linear video editing has gained renewed importance and relevance. Almost all available video editing systems today are dependent on considerable user interaction to produce coherent edited videos. In this work, we describe an automatic non-linear video editing system for generating coherent movies from a collection of unedited personal videos. Our thesis is that computing image-level visual similarity in an appropriate manner forms a good basis for automatic non-linear video editing. To our knowledge, this is a novel approach to solving this problem.

The generation of output video from the system is guided by one or more input keyframes from the user, which guide the content of the output video. The output video is generated in a manner such that it is non-repetitive and follows the dynamics of the input videos. When no input keyframes are provided, our system generates "video textures" with the content of the output chosen at random. Our system demonstrates promising results on large video collections and is a first step towards increased automation in non-linear video editing.

To my anna

Acknowledgements

I would like to thank Dr. Sudhir Mudur for his immense support throughout my association with Concordia University. His ideas formulated this research problem and his experience and expertise guided me to develop it into the present form. I express my deep gratitude to him for supervising and guiding me during the entire research work. Above all, I thank him for being so patient and accommodating.

I express my deep gratitude to Dr. Ananth Ranganathan for being a constant source of guidance and ideas. His support has been invaluable during this period.

A big thank you to all my colleagues, faculty members and staff of our department for being so wonderful and supportive all throughout!

I ascribe whatever I have achieved to Ramgopal Rajagopalan who has been my idol and mentor throughout my life. I dedicate this thesis to him and to my parents. ☺

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

As the adage goes "A picture is worth a thousand words". Man has evolved from a primitive form of Homo sapiens to create a highly intellectual and technological society today. Starting from the cave and rock art of Stone Age to the modern media of today, he has used visual representations as an important vehicle to convey his thoughts. In today's era, where there is an outburst of ever-evolving technologies, presenting visual data in more interesting ways has become a significant research challenge.

In present society, to the above adage we might also add "A video is worth a thousand pictures!" A static image merely captures an instant of time while a video can capture memories at length. Video also provides a powerful vehicle for informational, instructive, artistic, and many other forms of narrative. However, a major disadvantage of video is that it is a linear medium and the time required to view it is often large. While a photo collection containing hundreds of photos can be browsed in a few minutes using currently available software, a long video can be viewed only in a linear way such that we cannot randomly access any part of video without losing context. In today's world, this restriction on linear viewing is prohibitive in terms of time. People prefer shorter videos to longer ones, except when watching movies. Hence, popular video sharing sites such as YouTube usually restrict video lengths to ten minutes or so with no adverse effects on their usage

statistics. Consequently, systems for viewing and organizing video collections in a manner that can help mitigate shortcomings due to this limitation will have a wide impact.

While long videos are difficult to view due to time constraints, large collections of short video clips present a different problem. Such collections are commonplace today since some of the most ubiquitous devices are digital cameras and camcorders, which can capture short clips at will. Using these has become so intuitive that any enthusiast with minimal or no training can shoot videos of reasonable quality. Further, with the Internet being almost an integral part of our lives, the world is a small place today where sharing our pictures, videos or movies is just one-click away. At the same time, browsing video collections and editing video clips into a coherent movie remains far from easy. Hence, again, we encounter a growing need for applications that organize and present users' video collections in novel and efficient ways.

A number of applications and websites exist which cater to the specific needs of image organization, sharing, and editing. While digital albums and slideshows are convenient for organizing and viewing photos, photo editors also encompass a panoply of sophisticated features that have now become basic requirements, such as enhancement of color, sharpness, brightness, and contrast, highlight/shadow adjustment, image stitching and panorama creation, image layering etc. In addition, applications like Photoshop and Photofunia also allow creation of artificial images by splicing together elements of existing images. This form of image creation is being taken to the next level by research work such as Semantic Photo Synthesis [25], which can put together appropriate pieces from images in an image collection when given just a text based description of the final image desired (Figure 1.1). Phototourism [70] provides another very novel interaction method with photo collections by creating a 3D model of a monument or landmark for which photos are available from a large number of viewpoints.

While image applications have achieved the level of sophistication described above, the corresponding video editing software still has some ways to go. Firstly, there is no simple

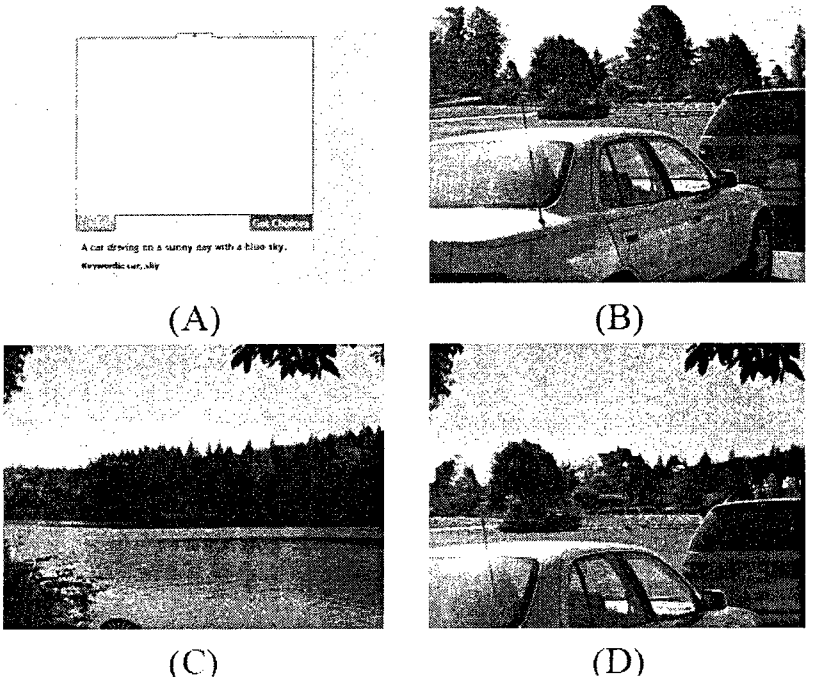


Figure 1.1: Semantic Photo Synthesis: In (A), the user specifies only a semantic caption. (B) and (C) show the automatically selected source images used to synthesize the designed photograph (D). Images taken from [25].

organizing system for video collections. The simplest concept is that of a playlist which is naturally suited only to linear access. Additionally, we can only know about the content of the video from the title or other textual description, which has to be supplied by the user. Providing such annotation is a tedious task and is also unsuitable for the short clips typically shot by amateurs using digital cameras. In the absence of a good organizational structure, applications have defaulted to visualizing the organization of videos in a manner similar to photos, by simply displaying a poster frame for each video. Currently available video editing tools are also largely focused on user-driven interaction with comparatively few generic editing tools available. Popular movie editing software such as Muvee and PowerDirector require that the user select and specify the order of the video clips to be edited into a movie. Removal of defects such as blur and unwanted camera motion have to be manually performed. Some of the top photo and video editing software available today are listed in Table 1.1.

Photo editing software	Video editing software
Adobe Photoshop elements	CyberLink PowerDirector
Corel Paint Shop Pro Photo	Corel VideoStudio
Serif PhotoPlus	Adobe Premiere Elements
Ulead PhotoImpact	Magix Movie Edit Pro
ACDSee Photo Editor	Muvee Reveal

Table 1.1: Some popular photo and video editing software.

1.1 Thesis Objectives and Summary

In this thesis, our goal is to create a system for automatically creating visually coherent videos from a given collection of video clips. We focus on using unedited video clips that would be typically obtained from a digital camera or a camcorder when used by an amateur. We abjure use of any textual annotations to the video clips since annotation is a tedious task and requiring them would be impractical. For the same reasons and because personal video clips usually do not have reliable audio, we also do not use any audio information in our system.

Our aim is to obtain visually coherent movies by concatenating appropriate segments selected from the video clips in the collection. By selecting segments that have similar image characteristics, we can obtain a video that does not contain too many abrupt transitions. At the same time, boundaries between video clips in the collection cannot be always removed and may appear in the output. The user can drive the selection and ordering process by optionally providing a set of input keyframe images that serve as "waypoints" for the output video, in the sense that the system has to select frames visually similar to the input keyframes in the sequence presented while also preserving visual coherence. The user can also determine, through the use of a parameter, how closely the output follows the video clips in the collection, i.e. how often it inserts transition frames in a video clip.

In summary, the objective of this thesis is -

> Develop algorithms and build a system for automatic creation of visually coherent videos. The two inputs to be initially provided to the system are (1)

a collection of video clips containing only image information, and (2) an optional set of keyframes that guide the content of the output video.

1.2 Applications

The primary application that we envision for our work is that of a presentation mechanism for video collections. Since our system can automatically put together coherent videos whose content can be driven by the user at a very high level, it is ideal for creating topical "highlights" of video collections. In addition, our system can also remove many video segments that have been rendered unusable due to improper lighting, motion blur, occlusions, and other technical shortcomings. Removing such unpleasant portions improves the quality of video clips significantly. Note that since our system does not explicitly detect motion blur or saturated lighting, it can only remove these where they are abrupt and severe. However, if required, special detectors for motion blur and saturated lighting [86] could be used in conjunction with our system. While we focus on personal video collections in this thesis, our system can be extended to automatically create sports highlights and movie trailers in a straight-forward manner. In these cases, audio input contains a large amount of information and cannot be ignored. The importance of these applications is attested to by the large amount of research in this area ([79] and references therein).

If no input keyframes are presented, our system creates a randomly generated, but still visually coherent, video which can be of infinite length and therefore, be useful as a screensaver video. This is similar in intent to VideoTextures [60], where video "sprites" were animated with random motions to create infinite videos. The sprites, in turn, were moving objects such as fish segmented out from other videos. In our case, the generated video has personal content from the video collection, instead of sprites learned from a few input videos. Personalized screensaver videos have the potential to become as popular as image slideshow screensavers that are currently widely used.

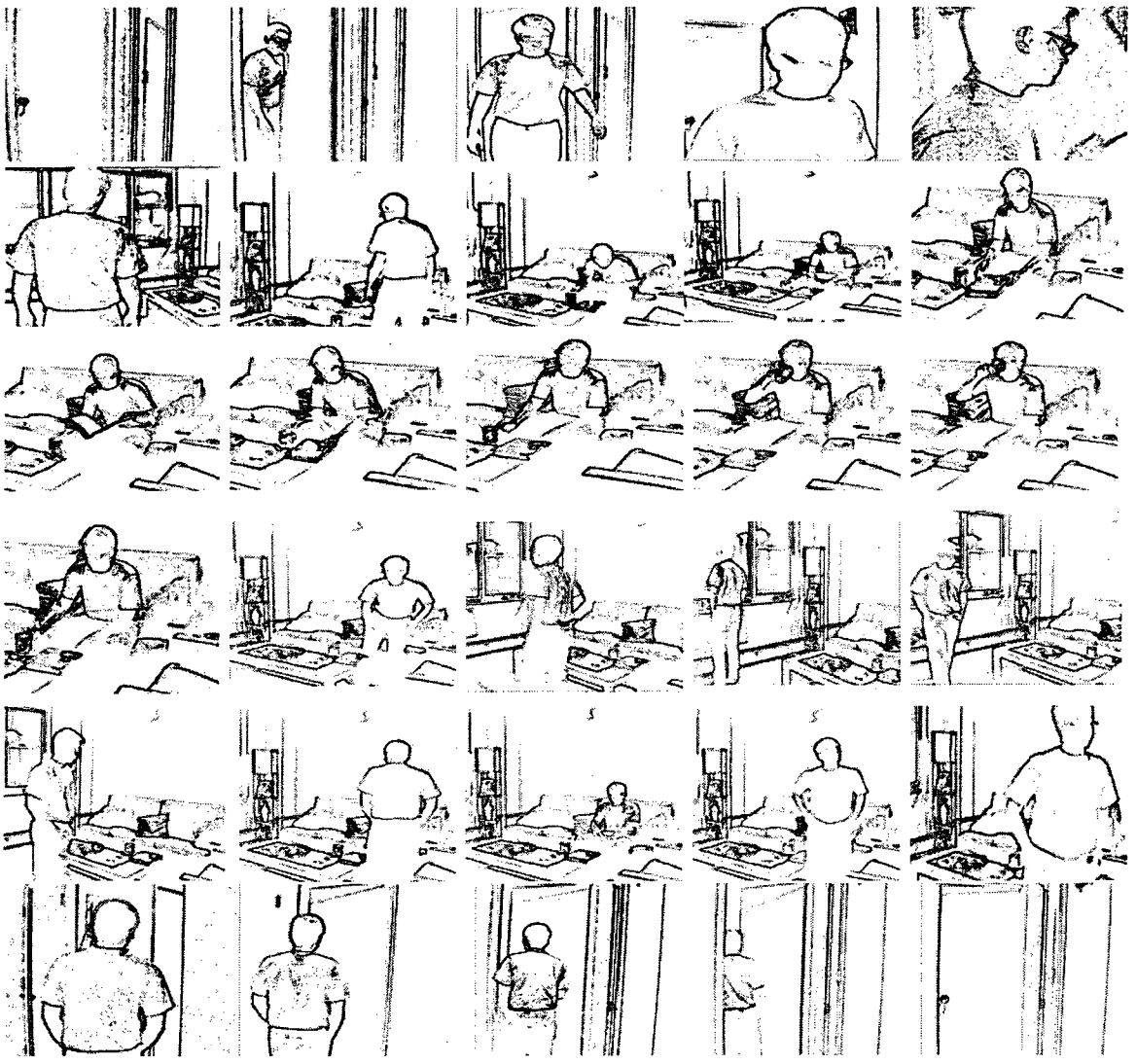


Figure 1.2: Image sequence showing a narrative where a man enters a room, sits on a sofa, picks up and reads a book from the table, keeps the book, picks up coffee mug, takes a sip of coffee, keeps the coffee mug on the table, walks to a window, and finally walks out of the room.

Another interesting application of our system is what is referred to as user-defined storytelling or automated storytelling. When a video collection contains many small clips that are visually similar, it may be possible to sequence them in more than one way while still maintaining a narrative. Of all the sequences possible, only one will represent an actual sequence in the video collection while the others will be sequences that were never shot in reality. These alternate sequences are called stories, and the creation of such videos is called automatic storytelling. Note that automated storytelling can use many modalities and sensors [56] even though the description just given only takes into account visual input.

An instance of automated storytelling is the following scenario where the user's video collection contains four short video clips. The content of the clips are as follows:

1. Clip1: A man walks into a room and sits on a sofa. He picks up a book from the table, reads it and later puts it back on the table.

2. Clip2: The man picks up the coffee mug from the table, sips the coffee and later keeps it down.

3. Clip3: The man gets up from the sofa and goes to the window.

4. Clip4: The man walks from the window and sits on the sofa. He then gets up and exits out of the door.

These four videos can be recombined in an interesting manner to make a single video that narrates a different sequence of events. We inter-mix the actions performed to create the following sequence:

- The man exits the room and after a while enters back into the room and walks to the window. He then sits on the sofa. He picks up the coffee mug from the table, sips the coffee and later keeps it down. He picks up a book from the table and starts reading it. Later, he keeps the book on the table and walks out of the room.

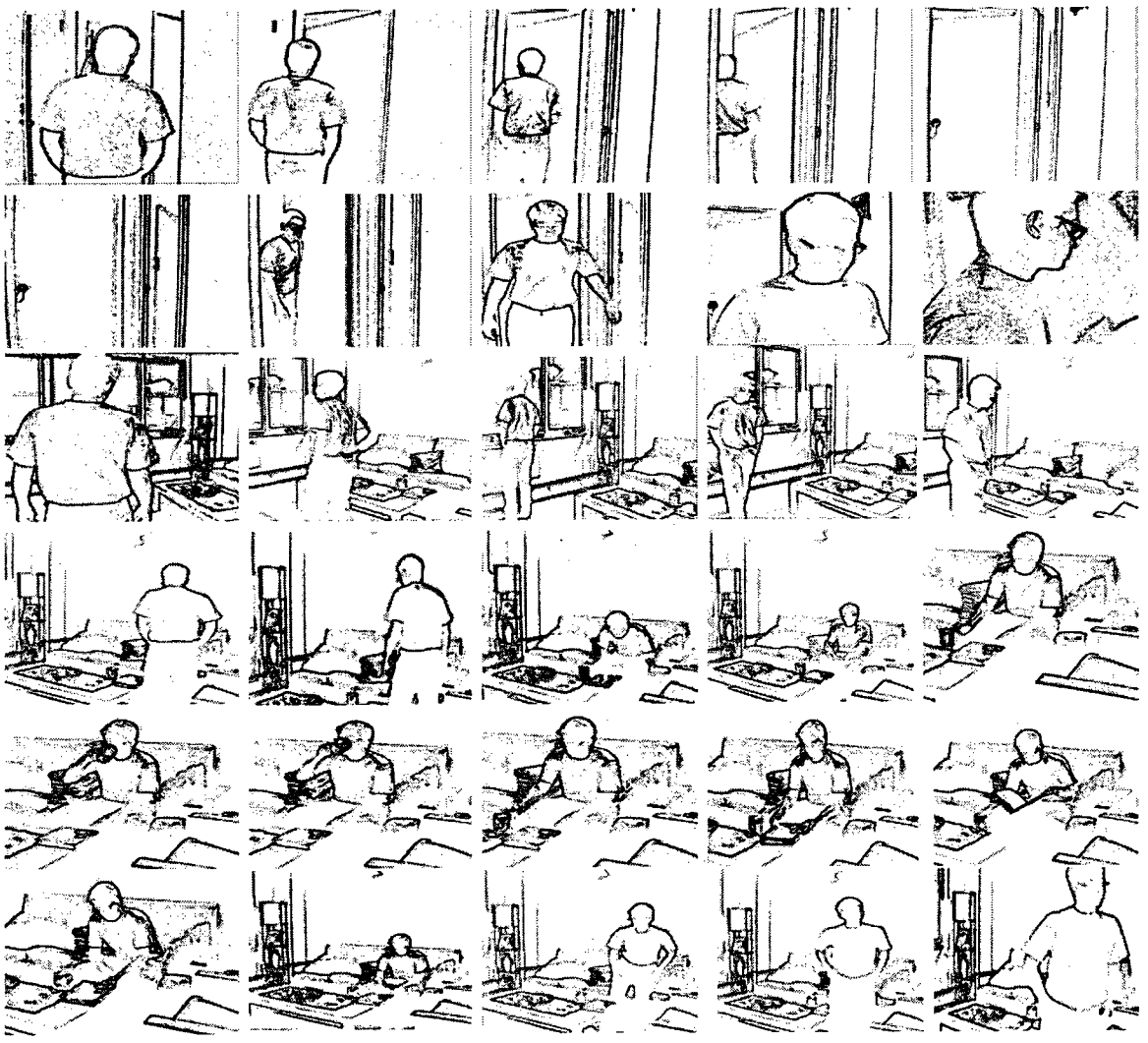


Figure 1.3: A simple instance of automated storytelling through re-arrangement of images in the sequence from Figure 1.2 . Note that the rearranged sequence maintains visual coherence, which is a necessary (but not sufficient) condition for it to carry a narrative. The narrative involves a man who goes out through a door, comes back again, walks to a window, sits on the sofa, sips coffee, reads the book, gets up and walks off.

This alternate sequence is a valid possibility that, however, never happened in reality. Our system can create such alternate sequences when appropriately guided by the user through input keyframes that drive the sequencing. The example above is illustrated through an image sketch sequence in Figures 1.2 and 1.3.

1.3 Technical Challenges

Automatic Video editing requires the solution of two problems -

1. **Data storage and retrieval, and algorithms for fast comparison of image frames**: The problem of storing and retrieving video frames from a large collection, which was a significant hurdle during early research, is no longer a serious concern with rapid increases in size of primary and secondary memory. However, comparison of image frames is a basic step in all automatic video editing methods since this determines visual similarity. If the features extracted from the video frames are in an amenable form (i.e. that of vectors), extremely fast nearest neighbor methods can be used for performing the comparison [62]. This is what is generally done in current work. Other specialized methods which are harder to scale to large video collections include image registration [53] and graph matching [51]. However, it is fair to say that the current state of the art can cope with image comparison up to the order of millions of images [74]. Note, however, that none of the popular image search tools on internet perform actual image comparison but instead rely on contextual text, captions, and metadata to return image search results.

2. **Video content understanding**:

   This relates to the ability to understand semantic content in the video collection which is a prerequisite to making coherent narratives. We discuss this problem in more detail below due to its inordinate importance for automatic editing.

Automatic video editing at the level of a human requires that the system understand the content of the videos, i.e the events occurring in the video, the objects in the frames, and the setting of all the scenes. Once all this is known, the system can put together this information along with the constraint of similarity to generate videos that will be indistinguishable from what a human can produce. Clearly though, this level of understanding is not possible given the current state of computer science research. We are currently at the stage where we can reliably recognize very few objects, faces being an exceptional case of recognition performance. Recognizing events and scene settings (i.e outdoors, office, etc) in a reliable manner is far beyond the abilities of current algorithms. Consequently, automatic editing also lags behind human level performance significantly.

The difficulty in detecting and recognizing events, objects, and scenes is due to the vast variation that is possible in imagery. Even when the setting is the same, changes in lighting and camera perspective can make a video look very different. If we add in possibilities of occlusion, where a part of the scene is blocked by some object, along with the changes in location of objects in the scene, the task becomes orders of magnitude harder. Due to these same reasons computing image similarity is also a hard task which has not been solved satisfactorily. The overwhelming majority of existing algorithms compare images on the basis of low-level features, assuming that the similarity at the pixel and patch level also holds at the semantic level. However, this assumption quite often fails in reality either when two semantically similar frames yield different low level features and vice versa due to lighting, perspective etc.

Given the current state of the art, three levels of implementing the task of comparing two images are possible (moving from higher to lower level comparison but from lower to higher accuracy):

1. Comparison based on the objects inside the images (Object recognition)

2. Comparison using annotations or labeling of the images under different scenarios, like nature or wildlife (Context-based)

3. Comparison based on color, texture, edges, and gradients of an image (Content-based)

In this thesis, we take the third approach and base our image similarity comparison on SIFT features computed densely on images. SIFT features have become the mainstay of feature-based computer vision work in the last decade due to their reproducibility, invariance to color and limited perspective change, and computational amenability. We compute an image representation using SIFT features that includes some location information from the image and provides a greater degree of robustness in image similarity comparison. However, as a consequence of the reasons stated in the above discussion, we note that SIFT-based image comparison is also not infallible.

1.4 System Overview

Our automatic video editing system consists of three stages -

1. Feature Detection
2. Computing Image Representation
3. Video Generation through image similarity computation

Feature Detection

We use SIFT features as the basis for detecting visual similarity in our system. SIFT features are detected on every video frame of the video collection, and are computed at a set of pixel location obtained by placing a grid over each frame. This is called dense SIFT computation and differs from initial work which only computed SIFT features around points of intensity maxima in scale-space [39]. SIFT features have been shown to be invariant to color and perspective and much more informative for the purposes of encoding underlying image characteristics than color, texture, and edge features. By covering the image with

densely detected SIFT features; we are assured of a reasonably unique feature set that will yield high similarity with other images only when they are truly similar.

Computing Image Representations

The image representation we use is a Spatial Pyramid histogram (SPH). In a straight forward scenario we would match the features from a frame with another frame to compute their similarity. Since each frame has thousands of features and the video collection itself may have thousands of frames, this can become intractable very quickly. Instead, the Spatial Pyramid (SP) approximates SIFT feature matching in a discretized space and is fast to compute.

The discretized feature space on which the Spatial Pyramid operates is obtained by clustering the feature space. All the features detected on all the frames are clustered to obtain a pre-specified number of clusters in SIFT feature space. Subsequently, for each frame, the SP computes histograms on the frequency of features occurring in each cluster. Histograms are computed at various spatial scales and on various parts of the image. All the histograms are then concatenated to yield the final image representation. Since histograms computed on various parts of the image have been included, the SP histogram contains rudimentary location information, making it even more robust for the task of computing image similarity. SP histograms are computed for every frame in the video collection.

Video Generation

Video generation is based on similarity between frames, which is computed using the Histogram Intersection metric on the SP histograms. As the video is generated, the next frame is selected as the one that is most similar to the current one. This requires comparing the current frame with all the frames in the video collection, which would be computationally expensive if done naively. We employ a two-level hierarchical method for the nearest neighbor search. The user may provide input keyframes that constrain the sequencing of

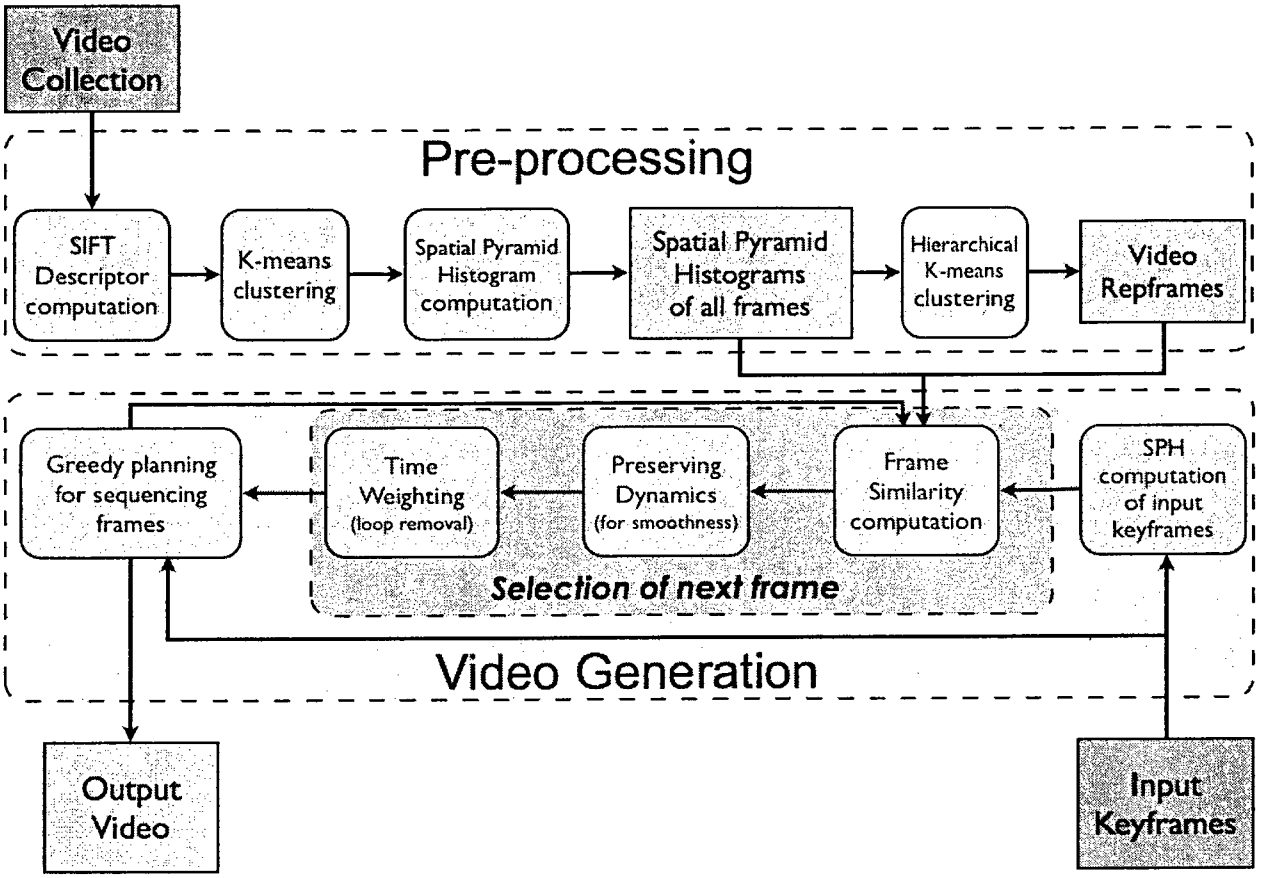


Figure 1.4: High-level schematic of our system: Pre-processing of the video collection is done offline while video generation can be performed online when the user presents input keyframes to the system that determine the content of the output. A detailed explanation occupies Chapter 4.

various videos. These are accommodated by computing a greedy shortest 'path' through the frames of the video collection where the path is constrained to pass through the keyframes.

While the histogram intersection metric is easy to compute, the visual similarity computation is made slightly more complicated by the need to avoid short loops and repetitiveness. At the same time, the transition from one video to another can be highly dissimilar but cannot be avoided. Finally, dynamics of the video, i.e. motions present in them, have to be preserved as far as possible to provide visual coherence. All these constraints are built into our visual similarity metric so that the correct frame is chosen for inclusion in the output video.

When computing similarity, the use of dense SIFT combined with an approximate matching scheme, implies that our system can perform rudimentary object detection when the object covers a large percentage of the video or is in a relatively uncluttered scene. For example, if we give an image depicting a Dalmatian as the input keyframe, the system will compare it to the frames in the videos and select those frames which are visually similar i.e. which contains Dalmatian. These frames are then combined with appropriate transition sequences, if available, to generate coherent videos. As mentioned before, our system does not require any annotation and only visual information derived from the frames is used. A high-level block diagram of our system is shown in Figure 1.4. Subsequent chapters provide required definitions and detailed descriptions of the various components of this system.

1.5 Contributions

This thesis makes four contributions. The primary contribution is:

> A novel system for automatically generating visually coherent videos from a video collection. The system needs minimal user input, scales to large video collections, and can also generate infinite videos similar to VideoTextures.

The secondary contributions are:

1. Use of dense SIFT-based spatial pyramid histograms for computation of frame similarity.

2. Introduction of spatial pyramid histograms using texture and census transforms, and their use as in (1.) above.

3. A greedy path computation method for generating video with visual similarity constraints.

1.6 Thesis Outline

In the next chapter, we provide background and related work in fields that overlap with our work. This is followed by a description of the Spatial Pyramid histogram in Chapter 3. Chapter 4 describes our overall system and contains the core of this thesis. Experiments and results are described in Chapter 5 followed by a conclusion and discussions in Chapter 6.

Chapter 2

Background and Related Work

Nonlinear video editing refers to editing methods that can perform random access on the source material. Before digital editing methods were invented, nonlinear editing involved cutting and pasting film rolls to perform the desired edits. It was thus a destructive process. Non-destructive methods became possible only with the advent of digital media. In this chapter, we provide background on non-linear editing methods and related research work, both at the system level and at the level of component methods used.

Human editors use their extensive semantic knowledge and their knowledge of the video narrative to produce edited videos. Automatic editing at such a level is currently not possible. Even basic semantic knowledge, such as events in a video and identity of objects that are being manipulated, is hard to obtain consistently. The problem of obtaining semantic information from low-level image features is called the semantic gap. Bridging the semantic gap in entirety would be equivalent to solving artificial intelligence. Due to the enormity of this problem, in this thesis we forego the use of semantic knowledge and focus on visual consistency. While many previous systems also claim to enforce visual consistency, we use features and image representations that are suitable for detecting visual similarity even in the presence of changes in color and perspective, thus obtaining greater robustness.

In this chapter, we discuss the work by other researchers on automatic non-linear editing, and also give some background on components we use in our system. We use image-level features in the form of histograms for similarity matching. We discuss our use of these features and also briefly give an overview of the methods used in other research. In addition, our system and the techniques presented in this thesis are closely related to a number of topics in computer graphics, video processing, and computer vision. Our system automatically generates video based on existing videos in a collection, and is based on selecting visually similar frames. Thus, our work is related to the areas of image indexing and content-based retrieval. Our use of image histograms is an instance of a "bag of words" model, which is a well-known method in computer vision. In computer graphics, our work can be considered to be an extension of video textures [60] to large, realistic video collections. The connection to methods such as Semantic Photo Synthesis [25], which creates novel images from pieces of annotated images available in a database, are also obvious.

More directly, at a system level our work relates to video segmentation and summarization, both of which have received significant attention in the literature. Automated storytelling is an application of non-linear video editing, which is also well-represented in the literature. We present relevant papers from these areas in the following sections.

2.1 Indexing and retrieval

Video data is generally stored sequentially in the form of frames which does not allow random access efficiently. Indexing and retrieval refers to transformations of the data or methods for extracting information from the video which lead to efficient random access and non-linear browsing. As discussed in Section 1.3, this is a hard and as yet unsolved task, though many systems that perform at a high standard exist. A comprehensive survey of the literature in image indexing is available in [8] and of video indexing in [2]. In this thesis, we index a video collection using the spatial pyramid histogram computed on each

frame. This transforms the indexing task to a nearest neighbor search.

We deal with an easier variant of the indexing and retrieval problem called "aimed search" [68], which involves finding a specific image similar to the input, the harder variant being category search. However, even this is made significantly hard by the wide variation in domain and visual characteristics.

Early methods for video indexing consisted of matching simple features such as color and texture histograms [73][69], scale space pyramids computed using Laplacian filters [55], and multiresolution wavelet coefficients [22]. More sophisticated indexing methods leverage the motion cues present in the video [54][1], while object-based indexing methods have also been popular [7]. The "Video Google" system by Sivic et. al [66]is a prime example of this approach. Video Google also uses view-invariant features which are matched between frames to recognize things of interest. Such features were introduced in [59] and a retrieval system based on these features was presented in [76]. Many variants of these features including Harris-Laplace [43] and maximally stable extremal regions [42] are now available, and an extensive comparison on various applications is presented in [44].

Transformations that convert image frames to other more amenable representations have also been used. Early work on iconic indexing is of this type, where the image is converted into a string representation [81]. More generally, a signature of the image frame or a region within the frame is computed. Image and region signatures have traditionally been modeled using probability distributions. The most common signature representation in the literature is using histograms modeled using discrete distributions. This is because, even though continuous distributions offer a more faithful representation [32, 9], they are less amenable for computational purposes.

We next discuss histogram representations and the associated bag of words models which are the most commonly used form of representation in image retrieval and computer vision.

2.2 Histogram Representation of Images

A histogram is simply the frequency counts of features that appear in each of the bins of the quantized feature space. Histograms have been the favorite image representation because they are easy to compute and offer multiple methods for feature matching during retrieval. Feature matching reduces to finding the distance between histograms for which various distance metrics with well-studied properties are available. The various components of obtaining a histogram representation are the underlying features used, the feature quantization method, and the histogram computation from the features. Additionally, we will also review various methods for histogram matching.

Widely disparate features have been used for computing image and region histograms. The common ones include color [73], texture derived from various type of filterbanks [41, 23], and image patch based features such as SIFT [66], census transforms [87], edgelets [82], and contour fragments [64]. Some of these are illustrated in Figure 2.1. That histograms can be computed from such a variety of features is itself an important reason for their popularity. Another advantage is that histograms computed using these different features can be combined to give a representation that captures multiple aspects of the image.

Feature quantization consists of discretizing the feature space so that each discrete component corresponds to a bin of the histogram. Quantization is very simple in the case of grayscale pixel values and consists of dividing the single dimensional value into segments. This can also be extended to RGB and other color spaces [48]. In some specific cases, such as with the Census transform, the feature space is itself discrete so that no additional quantization method is necessary [84]. More commonly, quantization is performed through clustering such as the widely used k-means algorithm. Sophisticated techniques that adaptively focus on regions in feature space with higher densities have also been tried and shown to yield better results over a number of tasks including image retrieval and categorization. Clustering using an ensemble of decision trees [46] and clustering through minimization of information loss [28] are two instances of work in this direction. All the quantization meth-

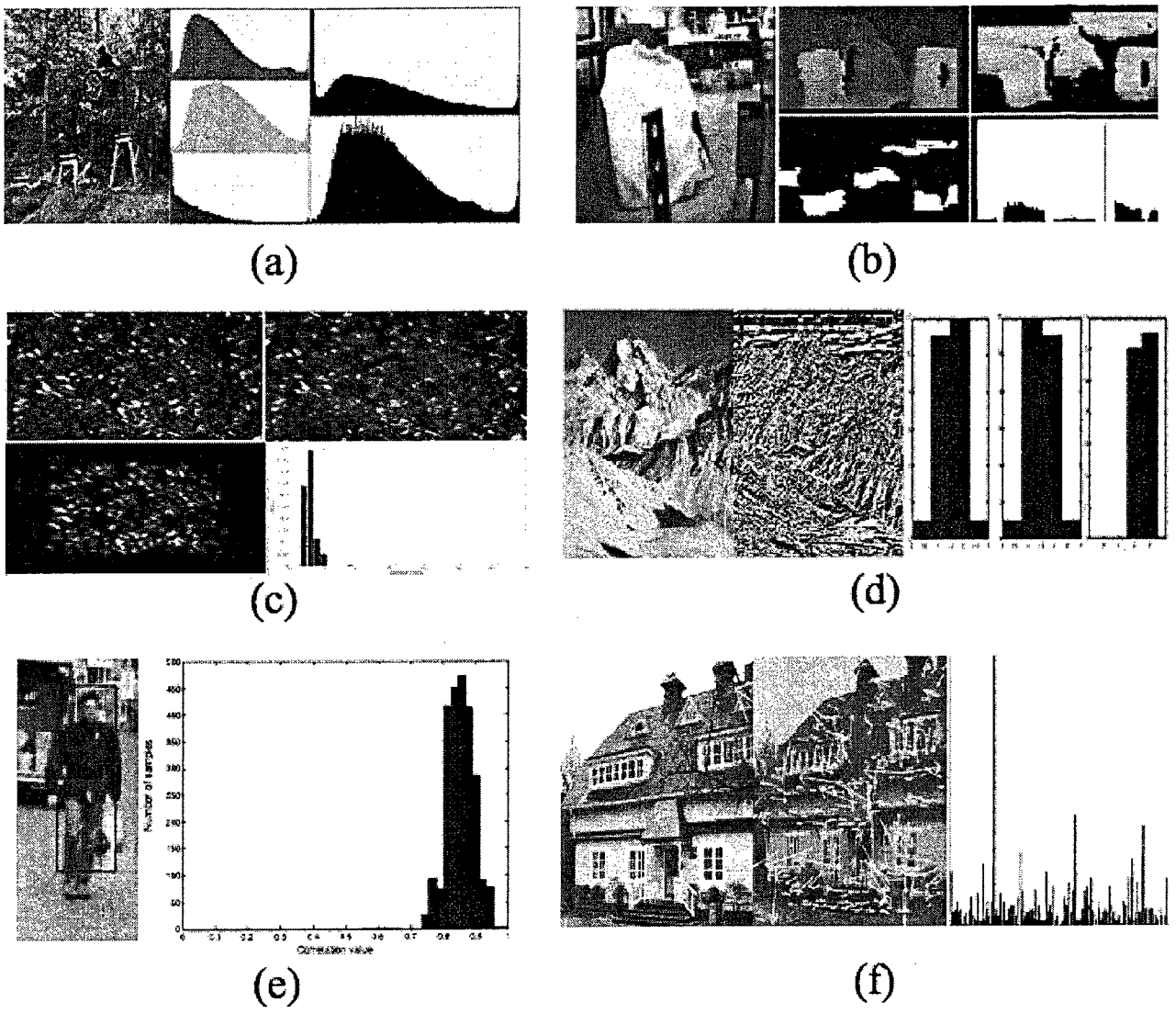


Figure 2.1: Histograms can be computed from a wide range of low-level image characteristics and features. Shown here are histograms computed from (a) Color (b) Contour (c) Texture (d) Census transform response (e) Edgelets (f) SIFT descriptors of Harris corner features.

ods discussed so far are unsupervised methods. For many domain-specific requirements, supervised learning using a training dataset is the best option [34].

Computing the histogram from the quantized feature space is straightforward though interesting variations exist. The most common method is to compute one histogram for the whole image by assigning all features from the image to the histogram bins. However, histograms can also be computed for regions of the image separately. These histograms are then concatenated to obtain the final image representation [84]. Alternatively, histograms can be computed at different spatial resolutions and then concatenated, and many variants of these multi-resolution histograms also exist [16]. In this thesis, we use a multi-resolution histogram computed on image regions obtained by placing successively finer grids on the image.

During image retrieval and for the purposes of image similarity computation, a number of histogram metrics are currently in use. The commonly used Euclidean distance is not theoretically appropriate but still often used due to force of habit, often with good results [26]. The L1-norm metric is another widely used distance function, especially useful for sparse histograms in which most entries are zero. The theoretically correct metric for comparing histograms is the Chi-squared metric, which is obtained by weighting each quantity in the Euclidean distance inversely by its frequency. The Chi-squared metric has a "variance stabilizing" property that compensates for greater spread at higher frequencies. The histogram intersection distance [85], which we use in this thesis, is useful when computing distance between unnormalized histograms, where the histograms entries are all integers. More complicated distance measures such as the Earth Movers' distance (EMD) [58] are also in use though limited by the effort required to compute them.

2.3 Bag of Words Models

In computer vision, methods that model images using collections of features are called "bag of words" models. The name comes from document modeling when documents are represented using just frequencies of word occurrences. The location and context of words is lost and for all practical purposes, the document may be viewed simply as a jumbled "bag" of words. Similarly, when the location information of features detected on images is not considered, we obtain bag of words models for images.

Bag of words models most often use histogram representations as their basis. The first impactful application of bag of words models to computer vision was Probabilistic Latent Semantic Analysis (PLSA) [67] which was used for object detection [12] and image classification [10]. PLSA has also been used for image annotation based on both image features and textual information [45]. Extensions to PLSA that allow location information have since become available [4] and more sophisticated models that can infer information about image segmentation and scene components have also appeared recently [72, 33]. Initially, learning of bag of words models was done in an unsupervised manner. However, this led to systems that could produce good results only on relatively simple datasets [52]. Supervised training methods have shown greater promise [5]. Bag of words models have also been used in video processing applications, such as for video segmentation [54] and clustering videos by the location they were shot in [61].

We use Spatial Pyramid Histograms [29] in this thesis, which is a bag of words model that uses multi-resolution histograms. Location information is maintained by computing histograms at various resolutions and spatial locations on the image and subsequently concatenating all the histograms. This provides more accuracy in the image matching and retrieval process than a model without any location information.

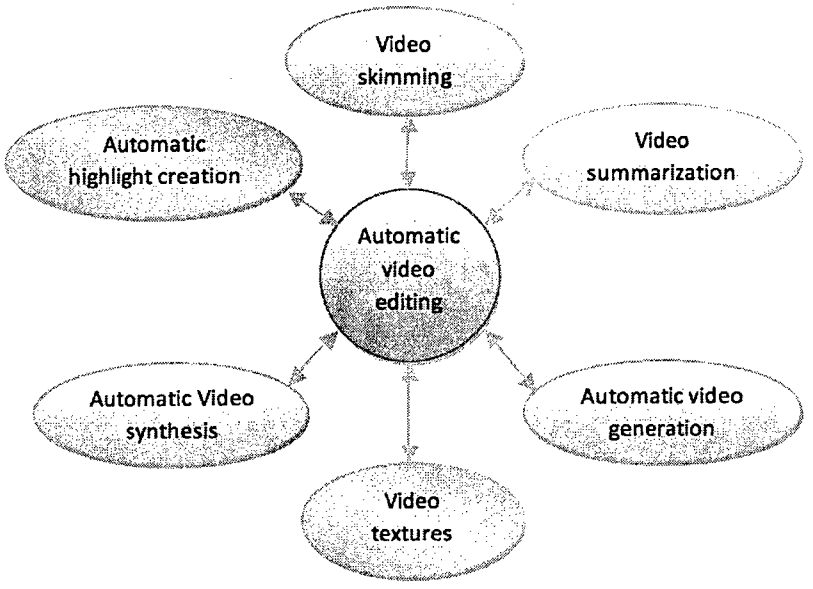


Figure 2.2: Automatic Video Editing is known by and closely relates to many video processing tasks in the research literature. Some of these tasks are illustrated here.

2.4 Automatic Video Editing, Summarization, and Storytelling

Automatic video editing is the focus of this thesis. However, the tasks that can be named automatic editing go under various names in the literature, including video summarization, automated storytelling, and automatic highlights creation as shown in Figure 2.2. We now review some of the literature in this area by providing a few instances to showcase the manner of problems categorized as automatic video editing while, at the same time, also describing the techniques used therein.

Peng et al. [50] introduce the problem under the name of "Automatic Video Skimming" and provide a method for accomplishing this. A major characteristic of their system is the attempt to satisfy aesthetic constraints by using movie editing theory [89] to compose scenes from existing video. This also includes removing unwanted portions containing blur, saturated lighting, and camera motion, which are done using simple heuristics. For instance, blurred images are defined to be those that have a lot fewer sharp edges than

normal images. They also focus on using an associated music track and matching the speed of the generated video to the rhythm of the music. Transition between shots is done through fades, dissolves, and short cuts, though visual similarity is not considered. Transitions are introduced by trimming shots, again based on music rhythm. This is similar to transitioning in commercial software such as Muvee, the disadvantage being that the user cannot control when the transitions happen and what content gets trimmed.

Automatic music video creation is another popular form of automatic video editing, and [50] above could also be considered to be this kind of work since they use music for tempo synchronization. Foote et al. [11] present methods for automatic and semi-automatic creation of music videos but do not explicitly remove errors due to motion blur as done above. Lee et al. [30] propose a video pre-processing and authoring technique facilitating music video creation but without using music rhythm and editing theory for clip selection.

Video summarization is a closely related topic on which extensive research has been conducted in the last decade and an exclusive TRECVID workshop on video summarization now exists [1]. The most basic summarization system compresses the original video by speeding up the playback [49] but this results in significant loss of detail. The goal is to create semantically meaningful summaries where parts of the videos that are "uninteresting" are deleted. For instance, in [24] an event-oriented abstraction scheme that detects interesting events and creates summary pieces around them is presented. Trajectories of moving objects are detected in [71] and video frames are chosen that would simplify these trajectories.

Automatic editing and summarization require semantic understanding of the video content. However, since this is hard, simpler approximations are often sought. Ma et al. [40] attempts to generate summaries by detecting "attractiveness" of each video frame or the degree that viewer may pay "attention" to the frames in the video, instead of understanding the semantic content. This method is also used by Hua et al. [19], who describe a complex

[1]Proceedings of the 2nd ACM TRECVid Video Summarization Workshop

system for editing highlights and music videos that takes into account an "attention" mechanism. The attention signals are derived from motion density and entropy of the image frames. An optimization algorithm is used to select high motion and high attention shots for portions with high tempo music and vice versa. In [80], the authors try to ascertain semantics by computing co-occurrence statistics between people appearing in a frame and the background. This requires robust face detection and identification. Summarization by clustering raw images based on color and computing a representative piece for each shot based on the clustering is presented in [18]. Generation of sports highlights is a related topic but one in which specialized techniques such as ball and player tracking, caption understanding, and player identification are used, an instance being [79]. Video-shop [78] is a system for implementing the low-level methods for editing analogous to photoshop for images. It provides methods for face replacement and painting, high dynamic range video compression and video compositing. While relevant to our work, these tools are orthogonal to the task of automatic video editing, which is more concerned with content selection and sequencing.

There are a number of automatic or semi-automatic summarization systems designed specially for home video collections, and hence, more relevant to the methods proposed in this thesis. Lienhart [36, 35] discusses the issues of low-quality home video and presents an automatic digest creation for home video. The method selects portions of video shots with good quality and inserts video effects for transitions. However, no sequencing is performed. Girgensohn et al. [13] propose a semiautomatic approach to home video editing, which also tries to find the best quality video clips based on automatic analysis. However, again sequencing is absent, and so, transitions for linking shots are not taken into account. Lindley and Nack [37] present a system similar to this thesis, where a planning mechanism is employed to select video clips from a video database and sequence them into a meaningful presentation for viewers. However, the features used by them are very low-level so that visual coherence is worse.

Another area closely related to automatic video editing is adaptive documentaries and cooperative storytelling. However, methods that are of importance in this area are quite far removed from those presented in this thesis. For instance, [88] presents a method to automatically create videos given annotated audio commentary. Here the main focus is to use a rhetorical structure commonly used in journalism to produce documentary videos automatically. This is extended to multiple users in [56], where it is termed as cooperative storytelling.

Some editing methods base their approach predicated on the availability of sophisticated computer vision components. Kosmopoulos et al. [27] describe a system for automatically producing personalized videos of museum visits. This is done by assuming availability of person tracking and identification, which allows extraction of frames in which a specific person occurs. In [14], the authors assume the availability of object detection and tracking for following objects of interest and modifying them in novel ways to produce new videos. However, object recognition and tracking are very complex tasks, as can be seen from the state of the art system proposed by Rothganger et al. [57] for this purpose, and are far from robust for general videos. However, they work better in controlled environments such as sports fields, and hence, are of more use for automatic sports highlights creation methods as mentioned above.

Chapter 3

Spatial Pyramid Histogram

We begin the technical description of our system by describing the Spatial Pyramid histogram (SPH) and its construction. Though the SPH is well-known in computer vision and has yielded the best results in image categorization so far [29], to our knowledge, this is the first instance of its application to video processing. Also, while the SPH has only been used with densely computed SIFT features previously; we extend its use to other types of low level features.

We use the SPH as our image representation, i.e. all images are converted to their corresponding SPHs before further processing is performed. The SPH is best-suited for our needs since it characterizes images excellently, while at the same time being amenable to similarity computation between frames. By saying that an SPH characterizes its image well, we mean that in the majority of cases, images that look alike have similar SPHs while images that do not, have disparate SPHs. This is so because it is a global representation but also contains location information about various regions of the image. In addition, these regions are distributed across various spatial scales. At the same time, there exists a well defined distance function over SPHs that makes it simple to compute a numerical estimate of the similarity or disparity of images.

The SPH was originally introduced for computing approximate correspondence be-

tween features in two images in an efficient manner. We explain this usage and the distance function that this naturally yields on SPHs. Following this, we explain the construction of SPHs on general feature spaces. The SPH is a histogram of lower level features, and while only SIFT-based SPHs have been used so far, any low-level feature that can be discretized can be used to build a SPH. Hence, we also explain the various instances of SPH used by us in detail at the end of this chapter.

3.1 Approximate Feature Correspondence using SPH

The SPH was originally introduced in [15] to solve the problem of feature correspondences between images. Correspondence refers to the problem of finding matching features between two images and is essential for a number of tasks in computer vision ranging from image retrieval and similarity computation to geometric reconstruction. When done naively, a feature in the first image has to be compared with every feature in the second, giving rise to the optimal solution with $O(n^2)$ complexity (where n is assumed to be the number of features in an image). However, the problem becomes even more complicated when we consider that the number of features in both images will invariably not be equal, and hence, many features in one of the images will have no corresponding match in the second. A well-known optimal solution for this partial matching problem exists, called the Hungarian algorithm [47], but is only guaranteed to be polynomial time. Considering that feature correspondence is a basic step in a variety of computer vision methods, this complexity is too high for practical use. Figure 3.1 illustrates an instance of the optimal partial feature matching problem.

The computational bottleneck in the above formulation is due to the insistence on optimal matching. By relaxing this, and searching for an approximate correspondence, we can do much better computationally. Grauman and Darrell [15] proposed spatial pyramid matching to find an approximate correspondence between two feature sets in n-dimensional

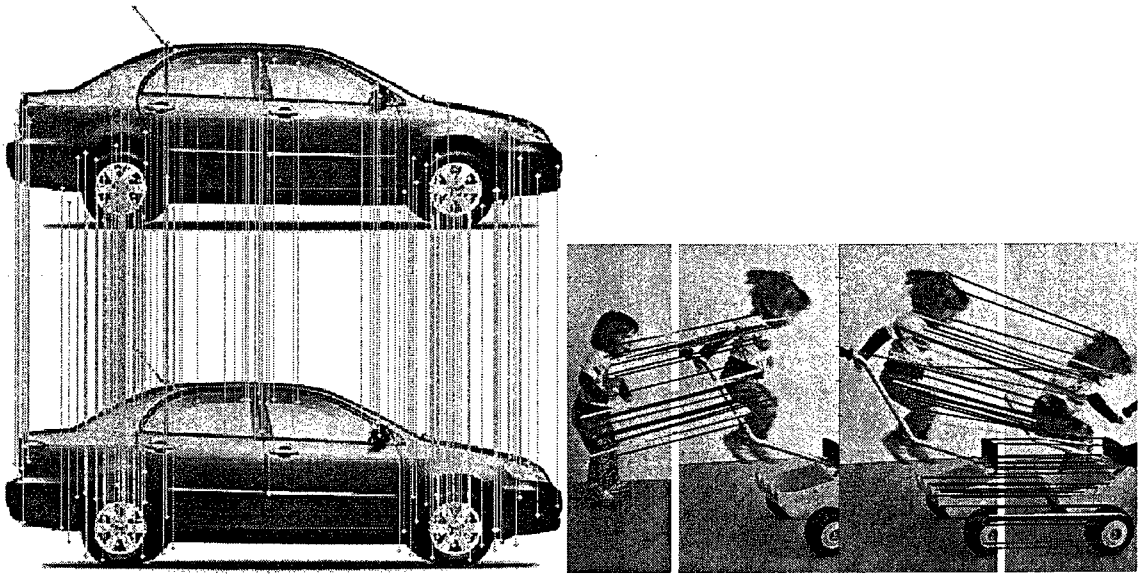


Figure 3.1: Feature correspondence between images (left) full correspondence where every feature is matched and feature cardinalities match between images (right) partial matching for a sequence of 4 images showing a girl jumping (from [75]). The number of features in each of the images is not the same in this case.

space. Informally, pyramid matching works by placing a sequence of increasingly coarser grids over the feature space and taking a weighted sum of the number of matches that occur at each level of resolution. At any fixed resolution, two points are said to match if they fall into the same cell of the grid; matches found at finer resolutions are weighted more highly than matches found at coarser resolutions.

The pyramid match approximation uses a multi-dimensional, multi-resolution histogram pyramid to partition the feature space into increasingly larger regions. At the finest resolution level in the pyramid, the partitions (bins) are very small; at successive levels they continue to grow in size until the point where a single partition encompasses the entire feature space. At some level along this gradation in bin sizes, any two points from any two feature sets will begin to share a bin, and when they do, they are considered matched. The pyramid allows us to extract a matching score without computing distances between any of the points in the input sets—when points sharing a bin are counted as matched, the size of that bin indicates the farthest distance any two points in it could be from one another. Each

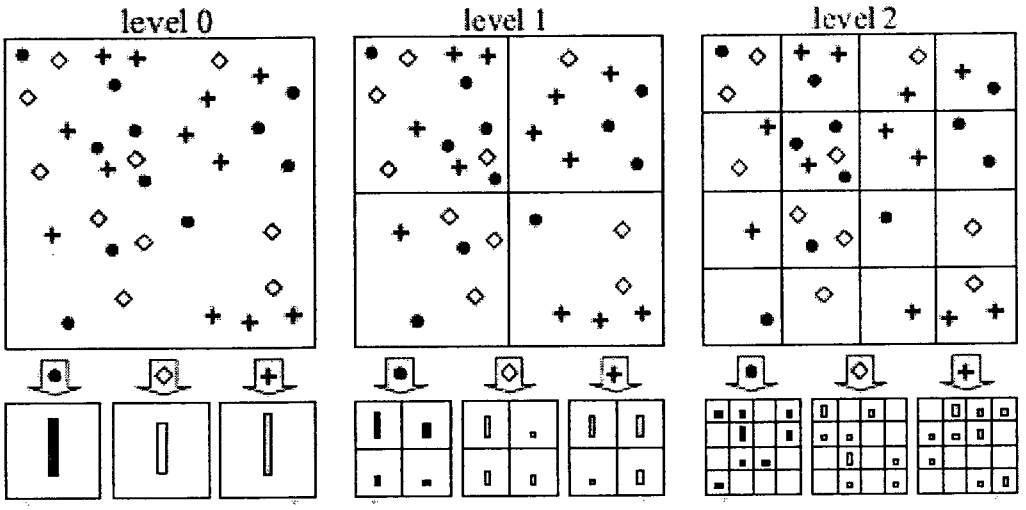


Figure 3.2: Toy example of constructing a three-level pyramid. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, the image is subdivided at three different levels of resolution. Next, for each level of resolution and each channel, the features that fall in each spatial bin are counted. Figure from [29]

feature set is mapped to a multi-resolution histogram that preserves the individual features' distinctness at the finest level.

Note that in the above algorithm, the bins at the finest level contain one feature each so that the algorithm can find a match to every feature if possible. However, since we are interested in similarity computation over a large collection of frames, we do not require such precision matching. Instead, we follow the approach of Lazebnik et al. [29], where instead of the features being an orderless set of points, they maintain some spatial information in 2D image space. Pyramid matching is then performed in the two-dimensional image space while traditional clustering techniques are used in feature space. Specifically, the feature vectors are quantized into M clusters and only features belonging to the same cluster are matched to one another. This is demonstrated in Figure 3.2.

3.1.1 The Histogram Intersection Metric

Once we have the histogram pyramids for the two images, they need to be compared to obtain the correspondence. As mentioned above, features in the same bin are said to have

matched, so that the number of correspondences for each bin of the histograms is simply the minimum of features in the corresponding bins. Performing this procedure for each bin of the histograms, we obtain the similarity between the histograms as

$$\mathscr{I}(x,y) = \sum_i \min\left(H_x(i), H_y(i)\right) \tag{3.1}$$

where x and y are images or image regions, H_x and H_y are their corresponding histograms, and the summation is over the bins of the histograms.

Correspondences at the different resolutions have to be weighted according to bin sizes to obtain valid implicit partial correspondence based on the finest resolution histogram cell where a matched pair first appears. We accomplish this by normalizing the histograms computed at various resolutions, shown in Figure 3.2, and concatenating them so that the final comparison can be done simply using a single application of (3.1). If there are L levels in the spatial pyramid, with level 0 corresponding to the whole image, and level L being the finest region, the histogram representation of the image, which we call the Spatial Pyramid Histogram, is given by

$$H = \frac{1}{2^L} h_0 + \sum_{l=1}^{L} \frac{1}{2^{L-l+1}} h_l \tag{3.2}$$

where h_l is the concatenated histogram from level l. Note the concatenation has to be done consistently, i.e. by taking the histograms from the various regions in the same order. The computation time of both the pyramids themselves as well as the weighted intersection is linear in the number of features. This may not seem true at first sight since the number of bins in the concatenated histogram for L levels and M clusters in feature space is $M\sum_{l=0}^{L} 4^l = \frac{M}{3}\left(4^{L+1}-1\right)$, where we have assumed a spatial subdivision into four parts at each level as in Figure 3.2. Hence for the typical values of $M = 200$ and $L = 3$, we obtain a histogram of length 17000 for each image. However, the important point is that the histograms, especially at the higher resolutions, are extremely sparse so that neither mem-

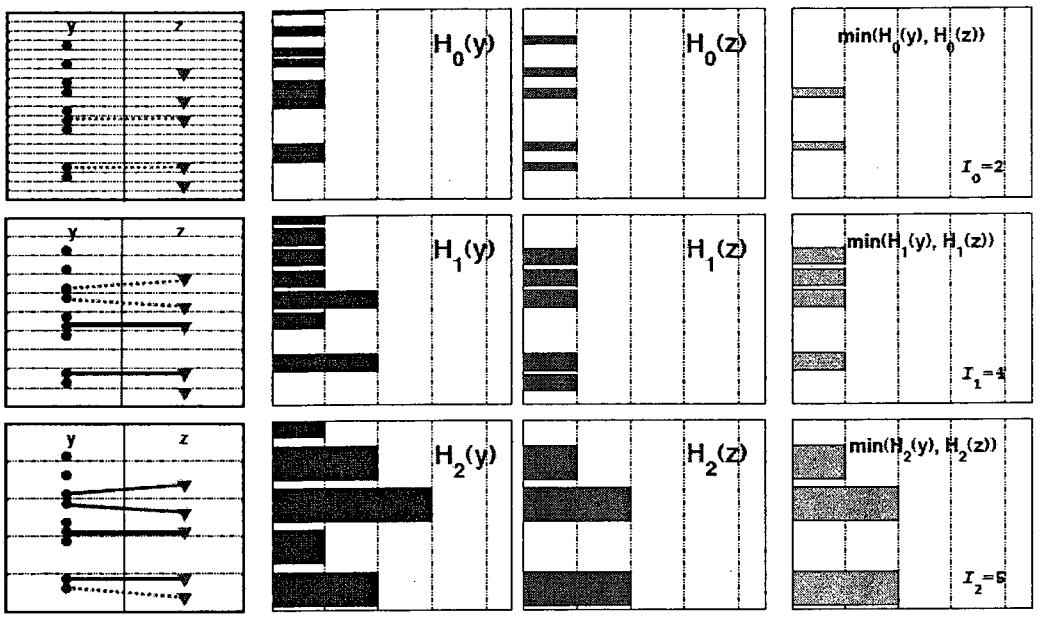


Figure 3.3: The spatial pyramid matching algorithm determines similarity between images based on partial correspondence by matching points once they fall into the same histogram bin. In this example, two 1-D feature sets are used to form two histogram pyramids. Each row corresponds to a pyramid level. In the first column, the set Y is on the left side, and the set Z is on the right. (Points are distributed along the vertical axis, and these same points are repeated at each level.) Light dotted lines are bin boundaries, bold dashed lines indicate a new pair matched at this level, and bold solid lines indicate a match already formed at a finer resolution level. In the second and third columns, multi-resolution histograms are shown for the two sets Y and Z, with bin counts along the horizontal axis. In the fourth column, the intersection counts between these histograms are shown. These intersection counts are used to measure how many new matches occurred at each level. A weighted sum of these gives the final pyramid match value. Figure taken from [15]

ory nor computational requirement increases with increasing L. The process of matching features using the histogram intersection function is illustrated in Figure 3.3.

Note that the histogram intersection value from (3.1) is a similarity value but not a metric since it gives a value of unity of exactly similar images, which reduces down to zero for totally disparate images. This is equivalent to an inner product in a vector space and we convert this into a metric as

$$D(x,y) = \mathscr{I}(x,x) + \mathscr{I}(y,y) - 2 * \mathscr{I}(x,y) \tag{3.3}$$

The histogram intersection metric of (3.3) is fast to compute, takes into account features at all spatial resolutions, and hence, is accurate in most of the cases. However, the effectiveness of (3.3) as a similarity metric is greatly dependent on the type of underlying feature used and we will discuss this further in the next section.

3.2 Image Features for SPH Computation

The only requirement for a feature to be used to compute a SPH is that it should be discretizable into clusters. Though this is the case, SPHs have been computed in the literature only using dense SIFT. We describe SPHs using three features, namely dense SIFT, texture, and census transforms, in the following discussion. We have found these features to yield the best results for our application with some trade-off possible among the three in terms of computational efficiency, as we will discuss in Chapter 5.

While it is possible to compute SPHs using even more basic features such as color, these do not yield good similarity values since even a slight change in lighting of the scene results in different features so that correspondence is lost. Hence, even though a feature space may be amenable to SPH computation, it may not be wise to use it in this manner since the feature itself may not maintain invariance required to provide good similarity values. We demonstrate this by implementing SPH with color as well and comparing it in Chapter 5 against the other features described below.

3.2.1 SPH using Dense SIFT

The use of dense SIFT for SPH was introduced by Lazebnik et al. [29]. Dense SIFT refers to the computation of SIFT descriptors on a set of patches in the image that cover most of the image. Most often, the set of patches is generated by placing a grid on the image. Dense SIFT provides good coverage of the image, and hence, is suitable for global image similarity computation.

Computation of SIFT descriptors is not be confused with SIFT *features* [39], which are obtained as maxima in scale-space of the image. SIFT features are computed using difference of Gaussian filters, and hence are also known as DoG features. In his introduction of SIFT features, Lowe [39] used DoG detectors as interest point operators on images, i.e. to detect point features of interest in the images. Image patches around these interest points were then converted into a standardized 128-dimensional vector called the SIFT *descriptor* of the image patch. In our work, we only utilize the SIFT descriptor, and not the DoG detector.

Moosmann et al. [46] demonstrated that the use of interest point detectors does not offer any advantage when attempting to capture the global characteristics of an image or image set. Instead, SIFT descriptors computed on randomly generated points on the image yielded better results. We follow a similar approach but instead of randomly generated points, which would make SPH computation cumbersome, we use points generated on a grid whose dimensions are a parameter of our system, though a typical value used is 8x8 pixels. Patches centered around pixels on this grid are used to compute SIFT descriptors.

Since we do not use the SIFT features themselves, we do not discuss them further. Instead, we focus on computation of the SIFT descriptor at a given image location. The descriptor is computed in a manner so that it is highly distinctive and partially invariant to variations such as illumination, 3D viewpoint, etc. While the original SIFT algorithm requires a scale at which to compute the descriptor, we simply use the natural image scale for this purpose in all cases.

We use a 16x16 pixel region, which we will call the image patch, around the desired location to compute the SIFT descriptor. This 16x16 patch is further divided into 16 regions of 4x4 each and histograms of 8 bins are computed from the magnitude and orientation values of samples for each of these 16 regions. This yields a 4x4 array of histograms with 8 bins each. During this computation, the magnitudes are further weighted by a Gaussian function with standard deviation equal to half the width of the descriptor window, i.e. 8

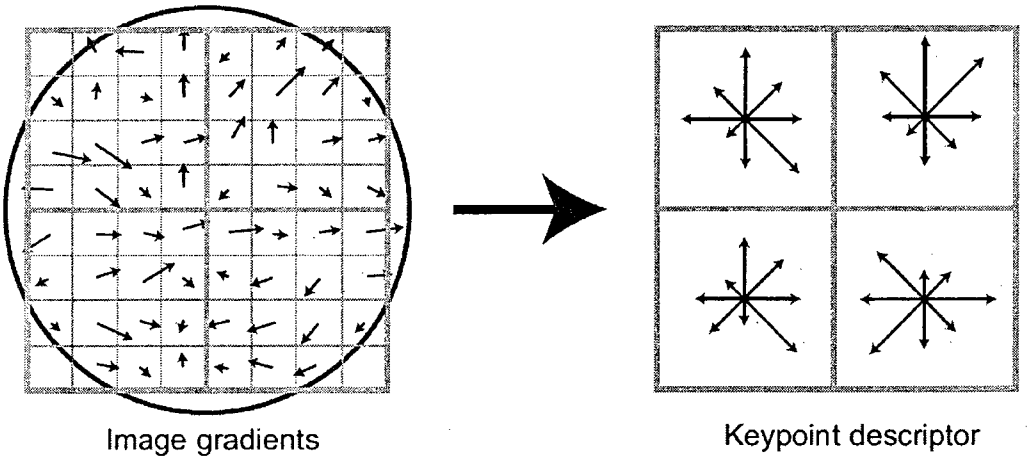


Figure 3.4: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples for illustration purposes, while in reality 4x4 descriptors computed from a 16x16 sample array are used. Figure taken from [39].

pixels. The descriptor then becomes a vector of all the values of these histograms. Since there are 4 x 4 = 16 histograms each with 8 bins the vector has 128 elements. This is shown in Figure 3.4.

Finally, the 128-dimensional vector is normalized to unit length in order to enhance invariance to affine changes in illumination. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in relative magnitudes for some gradients, but are less likely to affect the gradient orientations. Therefore, we reduce

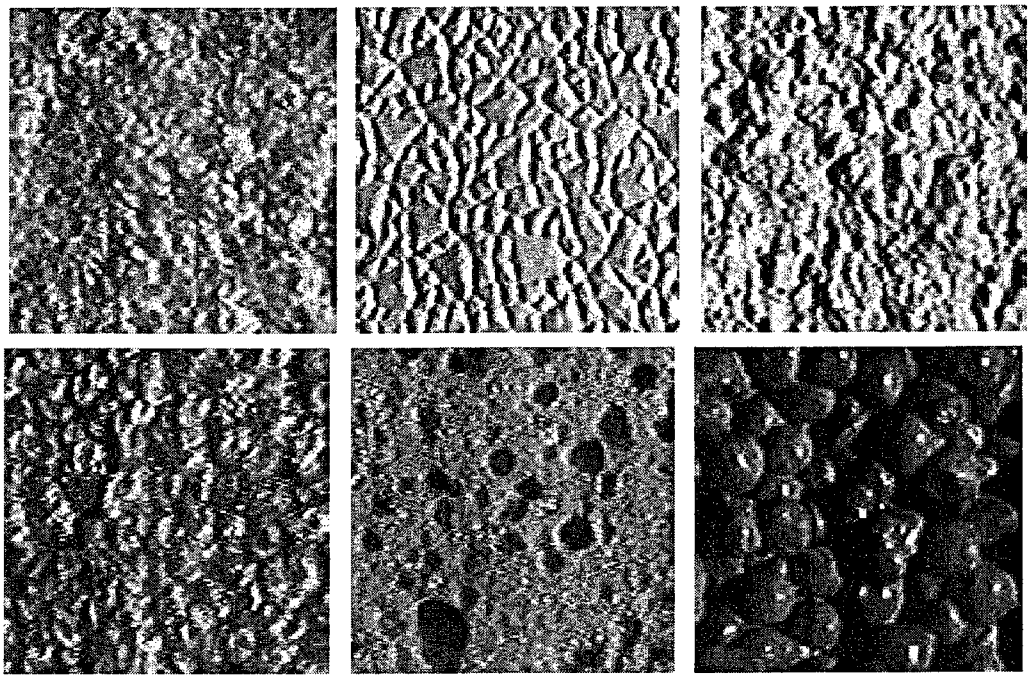


Figure 3.5: Examples of texture as repetitive patterns in real world objects. Note that texture can occur at different scales and can have different intrinsic orientations. Figure taken from [77]

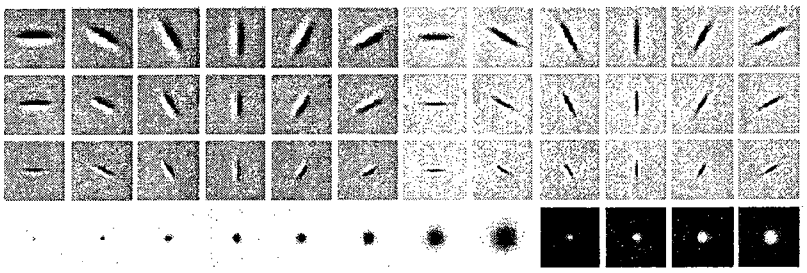
non-linear illumination effects by applying a threshold of 0.2, as in [39] and renormalizing the vector. The SIFT descriptors obtained are also partially invariant to perspective (3D viewpoint) and in-plane image rotation.

Once computed, the SIFT descriptors can be clustered and used to compute the SPH as described in Section 3.3.

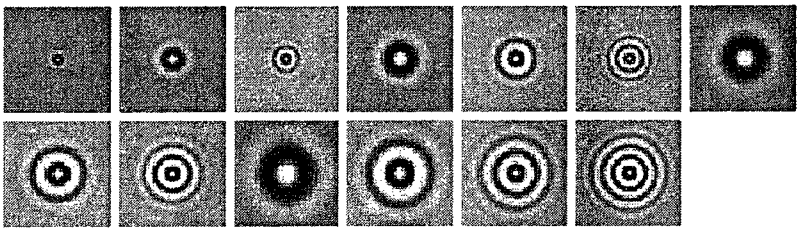
3.2.2 SPH using Texture

Texture refers to repetitive patterns in an image and the various types of textures can be used to characterize an image effectively for similarity computation. Since patterns are more common in images than regions of uniform intensity (objects such as wood, cloth, skin etc., as shown in Figure 3.5), texture is more useful than color in most cases.

Detection of texture requires care since the pattern defining it can occur at different scales and orientations. Hence, texture is usually detected not using a single detector but



(a) The Leung-Malik filter bank of Gaussian filters.



(b) The Schmid filter bank of rotationally-invariant Gabor filters.

Figure 3.6: Two commonly used filter banks for extracting texture from an image.

using a filter bank, consisting of a large number of filters, each having the capability to detect texture at different scales and orientations. The image is convolved with the filter bank to obtain responses at each pixel which capture the texture over the image. Commonly used filter banks include the Leung-Malik filter bank [31] which consists of 48 filters - 2 oriented derivatives of Gaussian filters at 3 scales and 6 orientations, 8 symmetric derivatives of Gaussian filters, and 4 are low-pass Gaussian filters at different scales. This is shown in Figure 3.6(a)[1]. Another commonly used filter bank consists of wavelets at various scales and is known as the Schmid filter bank (Figure 3.6(b)). 13 rotationally invariant Gabor wavelet functions comprise this filter bank. Though oriented textures cannot easily be captured using this filter bank, it is popular since the number of filters is much less than in the Leung-Malik case while texture recognition and segmentation performance is only slightly degraded [77].

Though the representation of textures using filter responses is extremely versatile, one might say that it is overly redundant. First, for an N-dimensional filter bank, each pixel

[1]Figures from `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`

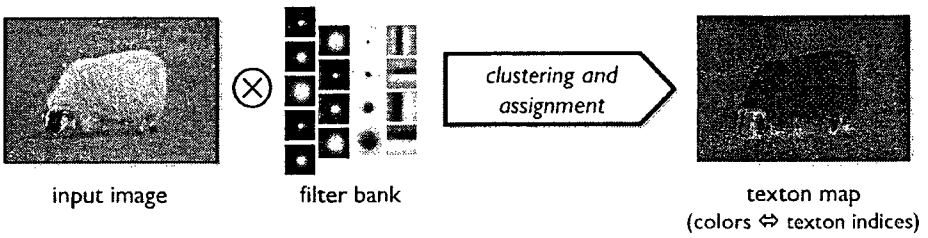


Figure 3.7: A texton map of an image is obtained by assigning the filter responses of the pixels to their corresponding clusters. Figure taken from [65].

of the image is converted to an N-dimensional vector after convolving with the filter bank. For large images, this implies a huge memory requirement. Second, it should be noted that since we are characterizing textures, which are entities with some spatially repeating properties by definition, we do not expect the filter responses to be totally different at each pixel over the texture. Thus, we expect that there should be several distinct filter response vectors and all others are noisy variations of them. This intuition has led to the use of clustered filter responses, called textons, rather than the original response vectors themselves [31, 41]. Textons can be considered as a set of prototype response vectors or as a dictionary of the various types of textures possible. Due to the latter analogy, textons are also known as texture words in the literature.

Textonization, the process of obtaining textons from filter responses, is performed by clustering the responses for the whole set of images of interest. Clustering is usually done using K-means with the number of desired textons being provided as an input parameter. The number of textons is conservatively chosen to be larger than the expected types of textures. This is because a larger number of textons only results in slightly less computational efficiency while too few textons can result in a massive reduction in retrieval performance.

Once textons are generated, the image pixels are mapped to their corresponding texton number (cluster in texture space). The resulting image is called a texton map (Figure 3.7). The texton map is much sparser texture representation than the filter responses since it consists of merely one number per pixel.

Computing the SPH using the texton map is straight-forward since the feature space has already been discretized. This is explained in Section 3.3.

3.2.3 SPH using Census Transform

We next describe the Census Transform [87] and SPH computation using it.

The Census transform is a non-parametric local transform originally designed for establishing correspondence between local patches. It compares the intensity value of a pixel, which is the center of a patch of interest, with the remaining pixels in the patch, as illustrated in (3.4) for the case of a 3x3 patch. If the center pixel is bigger than (or equal to) one of its neighbors, a bit 1 is set in the corresponding location. Otherwise a bit 0 is set.

$$\begin{array}{ccc} 32 & 64 & 96 \\ 32 & \mathbf{64} & 96 \\ 32 & 32 & 96 \end{array} \quad \Longrightarrow \quad \begin{array}{ccc} 1 & 1 & 0 \\ 1 & & 0 \\ 1 & 1 & 0 \end{array} \quad \Longrightarrow (11010110)_2 \Longrightarrow 214 \tag{3.4}$$

The eight bits generated from intensity comparisons in the above case, can be put together in any order (we collect bits from left to right, and from top to bottom), which is consequently converted to a base-10 number in [0 255]. This is the Census transform value for this center pixel. Similar to other non-parametric local transforms which are based on intensity comparisons, Census transform is robust to illumination changes and gamma variations. Analogous to a texton map, we can also replace the pixel values of an image with their census transform values. An example of this is shown in Figure 3.8.

Computation of an SPH using the Census transform is almost trivial since the feature space is intrinsically discretized as it can only take integer values. For instance, for the case of a 3x3 image patch illustrated above, the census transform can only take integer values between 0 and 255, which can be regarded as 256 clusters in feature space. For larger image patches, the number of clusters defined in this manner increases exponentially, and hence, conventional clustering methods may again be required. The computation of SPHs

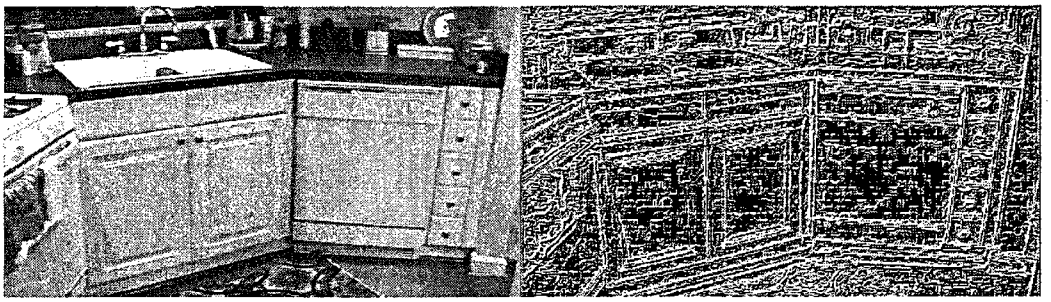


Figure 3.8: Example of a census transformed image. Figure from [83].

using the census transform is explained in the next section.

3.3 Computation of the Spatial Pyramid Histogram

The SPH is obtained by computing a multi-resolution histogram of the image using a discretized feature space. The discretized feature space is obtained by clustering the features over all the video frames in the video collection. The general algorithm for computing SPHs on any feature space is given in Algorithm 3.1.

In the case of dense SIFT features, we extract SIFT descriptors from all the video frames, as explained in Section 3.2.1, and cluster them using the K-means clustering algorithm. The number of clusters is typically chosen as between 200 to 400. After clustering has been performed, for every image, we assign the features in the image to their corresponding clusters. In the case of K-means, this is done by computing the cluster center nearest to the feature and assigning the feature to this cluster.

Subsequently, the image is divided into regions according to the spatial pyramid, and we compute the histogram for each region starting at the finest resolution. Each histogram consists of the number of features in each cluster in that region. Hence, the length of each histogram is equal to the number of feature clusters. The histograms of regions at a lower resolution are obtained simply by adding the higher resolution histograms of the regions that comprise it. More systematically, every region is divided into four regions at

Algorithm 3.1 Computation of Spatial Pyramid Histogram

1. Extract features from all the frames in the video collection
2. Cluster the features into a pre-specified number of clusters. (This step can be skipped if the feature values are naturally distributed in a small discrete set, eg. census transform)
3. For each frame f in the video collection
   (a) Assign the features from f to their corresponding clusters.
   (b) Divide f into subregions and keep dividing until the desired level in the spatial pyramid is reached.
   (c) Compute a histogram for each of the regions at the finest resolution level. Each bin of the histogram contains the number of features in the corresponding cluster.
   (d) Compute the histograms at the lower resolution levels l (bigger image regions) by adding the appropriate histograms at the higher resolution levels $l+1$.
   (e) Concatenate all the histograms after multiplying them with the appropriate normalizing factors given in (3.2). This is the SPH of frame f.

each level. Hence, the sum of these four histograms gives the histogram of the region at the lower resolution. Subsequently, all these histograms are concatenated with appropriate normalization as defined by (3.2). This is shown in Figure 3.9.

SPHs using texture are computed using textons, which are nothing but the discretized texture space. The texton-based SPH works similar to the manner described above except that the histogram in each region is computed as the number of pixels corresponding to a particular texton. This can be seen visually as the number of pixels corresponding to a particular color in the image to the right of Figure 3.7. The sub-division of the image into regions is done as above.

SPHs using Census transform can be computed even more simply since we need to only count the number of times a particular transform value appears in each region. However, this is true only if the patch size used to compute the transform is less than 3x3 pixels in size. For larger patches, we need to cluster the range of possible transform values into a

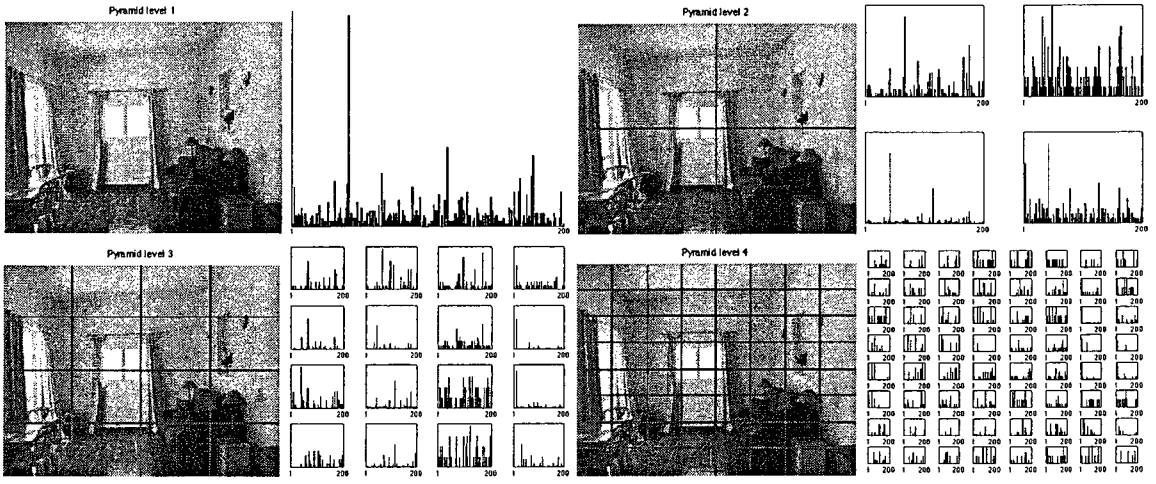


Figure 3.9: The Spatial Pyramid Histogram using Dense SIFT features. Histograms of clustered SIFT features computed on image regions at various spatial resolutions are concatenated after being multiplied with weighting factors to obtain the histogram representation of the image.

smaller and more manageable number. However, since the transform values are integers on the number line, this can be done very fast and without resorting to clustering algorithms such as K-means.

Chapter 4

Automatic Video Editing

We now describe the design and working of our system along with a detailed explanation on the associated algorithms. The purpose of the system is as an automatic non-linear video editor. To our knowledge, this is a first instance of a working system which, without being given any annotations, interacts with the user to automatically select and sequence the frames of a video collection into a visually coherent narrative. Currently, our system is a stand-alone system which is not web-based. The system can scale to large video collections.

The working of the system can be divided into two major parts based on its functionality - pre-processing of the video collection and runtime processing for video generation. Pre-processing involves two major tasks. First, to convert the video collection into a standard representation that can be used to compare the input images to the videos and find the similar images from the video. Second, to identify some of the frames from the video which can act as visual representatives or prototypes for the entire video. The first task involves offline computation of the spatial pyramid histogram (SPH) representation of all the frames of the video clips to be used for video generation. These histograms are used later at runtime for computing visual similarity between the frames. The second task of identifying representative frames, or *repframes,* of a video is necessary to make computation more

efficient when searching for similar frames during runtime.

The runtime processing of the system involves the task of video generation. This is when user interacts with the system to generate a video according to his specifications. The input is given in the form of keyframes that guide the video generation process in the sense that the output video contains frames similar to the keyframes, and in addition, the narrative is guided by the sequence in which these keyframes are presented. Video generation is performed using a greedy path finding algorithm that finds a sequence of visually coherent frames between the input keyframes. Video repframes computed during the pre-processing are used to speed up the selection of the frames during this process. We avoid repetitions and loops in the output video by incorporating the dynamics of the video in the similarity metric and also using time weighting to avoid repeatedly selecting a small set of frames.

4.1 Pre-processing

We compute the SPHs of all the frames in the video collection during pre-processing. Each frame is first converted to grayscale and then histogram equalized. Histogram equalization [20] is an image processing technique which is used for contrast adjustment in images. The number of different light intensities in an image often does not use the whole available spectrum and mostly accentuates a narrow spectrum. Images with such poor intensity distributions can be difficult to compare, and a more uniform intensity distribution is more amenable, which is the essence of what histogram equalization accomplishes (Figure 4.1[1]).

[1]Figure taken from `http://en.wikipedia.org/wiki/Histogram_equalization`

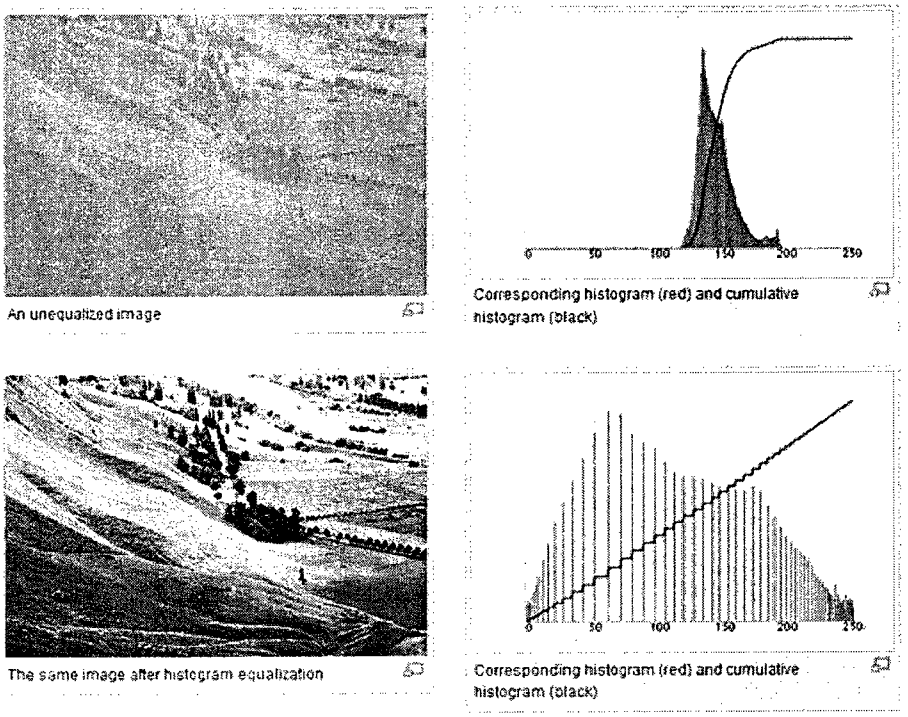


Figure 4.1: Histogram equalization adjusts contrast in images by making the intensity distribution more uniform.

SPH computation has previously been explained in depth in Chapter 3. The purpose of computing the SPHs is to enable fast similarity computation during runtime. However, the video collection may have thousands of frames, and it is not possible to compare all of these at each step of video generation process in an efficient manner. Hence, we resort to a two-level matching scheme where the most similar video shot is first selected based on representative frames, or repframes, of the video, and subsequently, the frame in the shot is selected.

4.1.1 Repframe Computation

Repframe computation is related to shot-boundary detection, which is a topic on which vast literature exists [17]. The purpose is to break down videos into smaller chunks that are visually similar and also more manageable than the much larger collection of individual frames. It is a basic component of many video processing algorithms, as it is in our case. Most shot-boundary detection work is aimed at well-produced film and television videos that have sharply defined sharp boundaries characterized by artistic transitions such as fade-outs, fade-ins, dissolves, and wipes. However, home videos do not have these pre-defined transitions and lower level cues have to be used to detect shots. A common method in use has been to cluster the video into visually similar and connected parts, each of which is declared a shot [61]. This is the approach we undertake. Following the clustering, we then identify one frame per shot as being most representative of the visual characteristics common in the shot. This is declared to be the repframe.

The primary difference between shot-boundary detection and our repframe computation here is that we do not enforce that the clusters form contiguous segments of video. Hence, the clusters do not form shots as they are defined in the literature. This is so because our approach is not dependent on using shots as the basic building block of the output video. We compute repframes solely for the purpose of making the similarity computation between frames more efficient.

Hierarchical K-means Clustering

We use hierarchical k-means clustering to automatically determine the number of clusters, and hence, the number of repframes in the video. In k-means based methods, a cluster is represented by its center (average of all features that belong to the cluster), which is simple and fast to compute. However, k-means requires that the number of clusters to be computed be given as input. Hence, a straight-forward application of k-means in this case would require us to know the number of clusters in the video, which is not the case.

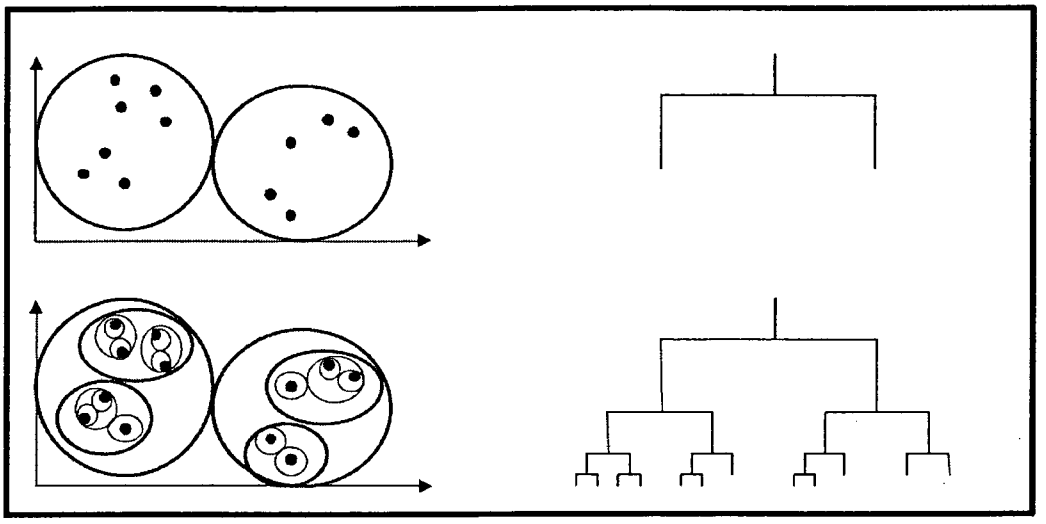


Figure 4.2: Hierarchical k-means works by recursively dividing the data into two clusters at each step until a stopping condition is met. The top row shows the state after a single clustering operation. The clustering tree for this scenario is shown on the top right. The bottom row illustrates the clusters and the clustering tree after a further four recursive clustering operations on both clusters of the top row. The division into two clusters at each step is accomplished using the standard k-means algorithm. Figure taken from [3].

Hierarchical k-means operates by dividing the data into two clusters at each step recursively. The clusters can be visualized as a tree with one cluster containing all the points at the root, as shown in Figure 4.2. The subdivision of data into two clusters at each step continues until some stop condition is met. Usually, this is either a minimum number of points in each cluster, or the maximum average distance between points provided as a threshold. Recursive clustering is continued until one or both of these conditions is met. In our case, all the clusters that form the leaves of the clustering tree are declared to be video clusters.

The division into two clusters at each step is performed using standard k-means. The above technique is called top-down hierarchical clustering. An analogous bottom-up method is also in use, where each cluster is initialized with a single data point, and clusters are merged pairwise until a stopping criterion is reached. We do not discuss this and other variations of hierarchical clustering since these are not related to our work here.

Hierarchical K-means for Repframe Computation

The video clustering could be done in image space, i.e. simply by converting the frames of the video into long vectors and clustering them. However, this is inefficient due to the size of the image vectors, and also has the same problems we encountered while trying to pick a robust visual similarity search, viz. it is not invariant to lighting, contrast, and a little perspective change.

We choose to perform the hierarchical k-means in the space of Spatial Pyramid Histograms (SPH) of the frames. However, we cannot use the usual k-means method which is based on Euclidean distance since this does not capture the visual similarity between two SPHs. Instead, this is done by the histogram intersection function, which we use to perform k-means. Just as in the Euclidian case, here too the center is typically not one of the points in the data set. Hence, the centers will not represent any image except in the most improbable circumstances. This poses a problem for us since in the usual scenario, the repframes would be the centers of the clusters computed for this purpose. We overcome this issue by declaring the repframe corresponding to the cluster to be the one having the minimum distance to the cluster center, i.e. the one most visually similar to the center.

We use the k-means variant for this purpose as explained in [85], though the clustering algorithm explained there is not hierarchical. The difference between the standard k-means algorithm and the hierarchical k-means algorithm used by us is shown in Algorithms 4.1 and 4.2.

In summary, we compute repframes for each video in the collection as follows. The SPHs of the video are clustered using hierarchical k-means as explained above. We use both the stopping criteria of specifying a minimum threshold size for the clusters and of specifying a maximum threshold average distance among the cluster members. These stopping criteria are chosen empirically so that, in the general case, 3-5 repframes are computed per video.

A workflow for the computation of SPHs is given in Figure 4.3 while the workflow for

Algorithm 4.1 Standard K-means

Input: Data points X and the number of clusters required k

1. Randomly select k data points from X as initial cluster centers.
2. While iter < MaxIter and cluster membership has changed, do
   (a) Form k clusters by assigning each data point to its closest cluster center where the distance is computed using the Euclidian metric.
   (b) For each cluster in 1 to k
   Calculate cluster centers using all data points contained in the cluster

Algorithm 4.2 Recursive algorithm for Hierarchical K-means using the histogram intersection function

Procedure Name: HKMeans(X, $N_{\min}$, $D_{\max}$)

Input: Data points X , minimum elements per cluster $N_{\min}$, maximum average distance among cluster members $D_{\max}$

1. Divide X into two clusters with elements X_L and X_R using k-means with the histogram intersection (HI) metric (3.3).
2. Compute average pairwise distance in X_L and X_R according to the HI metric. Denote these by D_L and D_R.
3. If no. of points in $X_L < N_{\min}$ or $D_L < D_{\max}$
   Find element closest to center of the cluster $\{X_L\}$. Add the corresponding frame to the set of repframes R.
   Else
   HKMeans(X_L, $N_{\min}$, $D_{\max}$)
4. If no. of points in $X_R < N_{\min}$ or $D_R < D_{\max}$
   Find element closest to center of the cluster $\{X_R\}$. Add the corresponding frame to the set of repframes R.
   Else
   HKMeans(X_R, $N_{\min}$, $D_{\max}$)

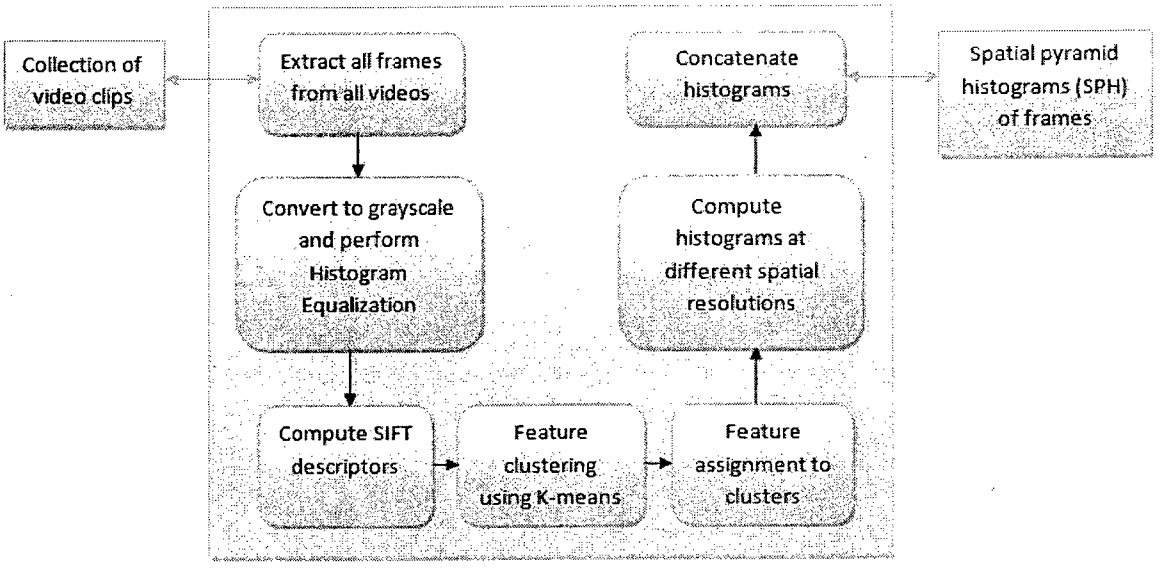


Figure 4.3: Workflow for SPH computation for the frames in the video collection.

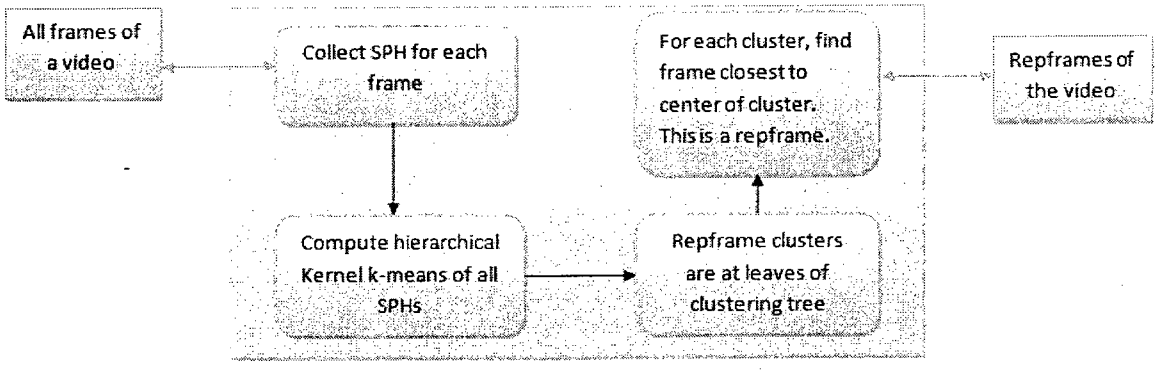


Figure 4.4: Workflow for Repframe computation for videos in the collection.

computing repframes is given in Figure 4.4. These two steps comprise pre-processing. We now move on to describing the video generation procedure at runtime.

4.2 Video Generation at Runtime

Runtime is when the user presents input to the system to generate videos. Input is in the form of frames that determine the content of the output video. We call these keyframes as the output video passes through these frames, and they act similar to waypoints in a route. This input is optional and the user may also simply ask the system to generate a video, in which case the content of the output video is randomly decided upon, as we describe in a subsequent section of this chapter.

Generating the output video requires finding visually similar frames at each step starting from the first input keyframe, if it is provided. These frames can be imagined to form a "path" through the video frames. Such paths are started from each input keyframe and these are merged using a greedy path finding algorithm. By this procedure, we ensure that the output video passes through all the keyframes and contains frames visually similar to them.

However, video produced this way is jerky and will usually contain small loops and repetitive frames. We avoid unnecessary repetition by introducing a time-weighting factor into the frame selection process. This ensures that frames are not repeated too often. Video smoothness is controlled by additionally incorporating the dynamics from the input video clips into the frame selection process. This is done by modifying the metric to compare the visual similarity of short sequences of frames rather than single frames. We now explain all these components below.

4.2.1 Finding visually similar frame(s)

Starting with the first input keyframe, if provided, or a randomly selected frame if not, video generation proceeds by selection of frames sequentially. First, the SPHs of the input keyframes are computed as described in Chapter 3 and Figure 1.4. The clusters computed during the pre-processing step are used for this purpose. Following this, a visual similarity search is conducted over all the frames of the video collections to obtain the next frame which is closest to the current frame according to the histogram intersection metric (3.1).

Since the number of frames in the video collection may be a huge number, it is not computationally tractable to perform a naive linear search to obtain the next frame. Though nearest neighbor methods such as KD-trees [21] can perform the search in $O(\log n)$ time where n is the total number of frames, these assume a Euclidian space and also require all the SPHs to be stored in main memory, which may not always be possible. Again, nearest neighbor methods that can deal with this by minimizing disk access or by ensuring disk accesses are linear rather than random [6]. However, these have the disadvantage of complexity.

We address this challenge by adopting a two-level search procedure. We first find the most similar frame clusters by comparing the current frame to all the video repframes, computed during pre-processing as described in Section 4.1.1. Subsequently, the most similar frame is obtained by comparing the frames in these clusters. This two-level search is illustrated in Figure 4.5. Only the repframes are kept in main memory at all times during the search process. The frame belonging to the clusters corresponding to the repframes need only be fetched when the repframe is found to be the most similar to the current frame. Hence, the average complexity of selecting the next frame of the video is $O(N_R N_c)$ where N_R is the number of repframes in the collection and N_C is the average number of frames per cluster.

The comparison between the current frame and repframes (in the first stage) and cluster frames (is the second stage) is done as follows. If the current frame is a keyframe, nothing

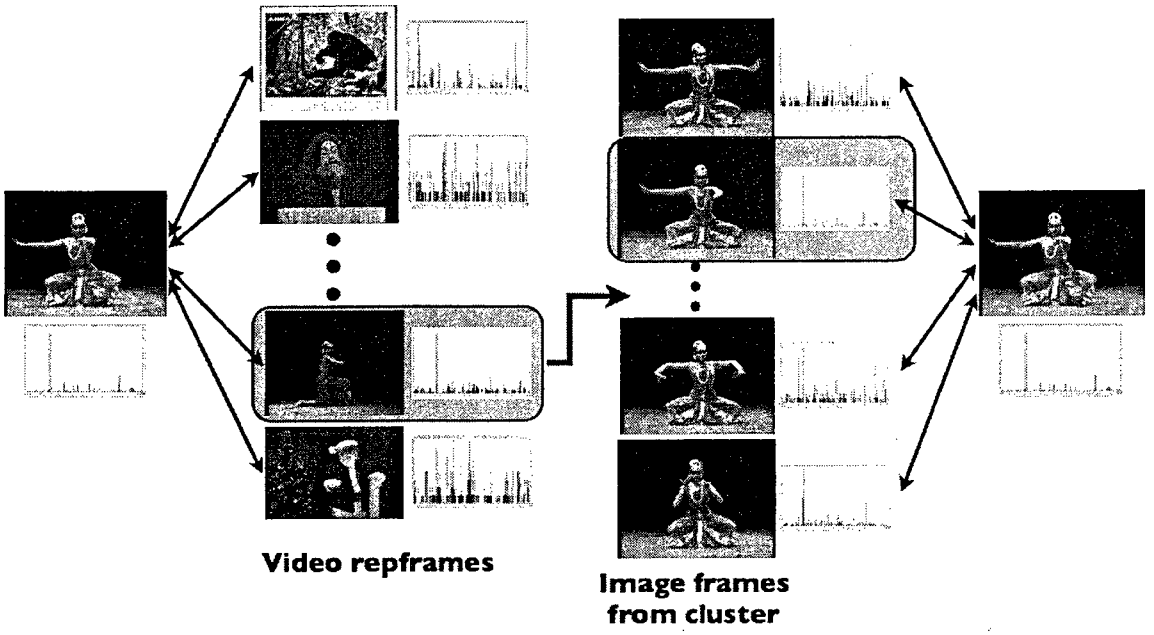


Figure 4.5: Selection of next frame of output video. (left) The current frame is matched with all repframes to select the closest cluster (shaded). (right) All the frames in the selected cluster are again matched with the current frame to obtain the best one (shaded). The current frame is shown at both the extreme left and right of the image.

more needs to be done. Otherwise, we obtain the frame succeeding the current frame from the video clip to which it belongs. This frame is used in the search process to compute the similarity using the histogram intersection distance metric. We use the succeeding frame so that the output video preserves some dynamics of the input video clips. However, even in this case, if the succeeding frame does not belong to the same repframe cluster as the current frame, or the current frame is the last one, we simply use the current frame in the search process. This is so because the succeeding frame may well be from a different shot or having unwanted effects such as blurs or light saturation. Hence, by not using these we automatically edit them out of the output video.

4.2.2 Greedy path finding algorithm

The above method only selects the next frame given the current frame. We now explain how these frames are sequenced to obtain the output video.

We consider the input keyframes to be arranged sequentially and the first and last keyframes are taken to be the start and end frames of the output video. Assuming N keyframes and denoting these by $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_N\}$, we compute the video segments between each pair of keyframes independently and concatenate them to obtain the output video. For each pair of keyframes $\mathscr{K}_i$ and $\mathscr{K}_{i+1}$, the video segment between these keyframes is generated using a greedy path finding algorithm.

Our greedy algorithm operates by simultaneously computing similar frames from $\mathscr{K}_i$ in the forward direction and from $\mathscr{K}_{i+1}$ in the backward direction. Forward computation is the same as described in Section 4.2.1 above. Backward computation of frames from $\mathscr{K}_{i+1}$ is similar except in the way that the frame used in the similarity search is selected. Instead of using the succeeding frame from the video clip to which the current frame belongs, we use the preceding frame, so that the video dynamics move in reverse. However, as before, if the preceding frame does not belong to the same repframe cluster or the current frame is the first frame of a video, we use the current frame itself in the similarity search.

The forward and backward frame "paths" are grown until they "merge", at which point the sequence is taken to be the video segment between keyframes $\mathscr{K}_i$ and $\mathscr{K}_{i+1}$ as shown in Figure 4.6. Repeating this for each pair of keyframes, we obtain the complete output video by concatenating all the video segments. In order to ensure that merging will indeed occur even in the cases when keyframes $\mathscr{K}_i$ and $\mathscr{K}_{i+1}$ are very dissimilar, we introduce a time weighting factor which is discussed in a following subsection.

It may appear that since we have a metric given by the histogram intersection function, we could use an optimal shortest path such as Dijkstra's algorithm to find the video segment between the keyframes. However, this does not work since the optimal shortest path between two keyframes is simply to transition from the first one to the second one with no frames in between. This is so because the histogram intersection function is a true metric,

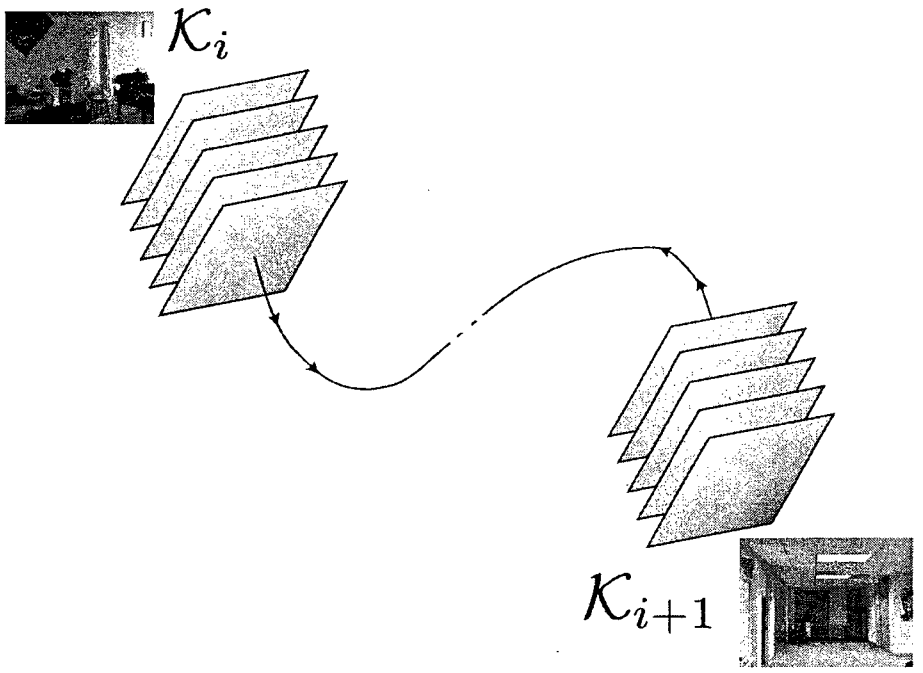


Figure 4.6: The greedy path finding algorithm works by searching for visual similar frames in the forward direction from keyframe $\mathscr{K}_i$ and in the backward direction from keyframe $\mathscr{K}_{i+1}$, and growing these paths until they merge.

i.e. it follows the triangle inequality, so that for any three frames A, B, C, we have

$$D(A,B) \leq D(A,C)+D(C,B) \tag{4.1}$$

where $D(.,.)$ is as in (3.3). Further, previous work on video textures [60] introduced the use of dynamic programming for computing a path between two frames given a similarity metric. However, both the time and space complexity of dynamic programming are polynomial in the number of data points. Hence, this method can only be adopted if the total number of frames under consideration is not more than a few hundreds, as was the case in [60]. In our case, where the total number of frames in the video collection can run into thousands, straight-forward use of this method is ruled out. Our greedy algorithm provides a simple but robust solution to this problem.

4.2.3 Incorporating dynamics into the metric for smooth output

The above algorithm while producing visually similar frames in sequence, does not maintain dynamics. By this we mean that if more than one frame exists, which is visually similar to the current frame, the frame which naturally follows the current frame based on actions occurring in the frame should be chosen. In other words, the frame order which appears in a video of the video collection should be preferentially chosen.

To maintain video dynamics, we modify the distance function to take into account the distance between succeeding frames. Remember that we use the succeeding (or preceding) frame in the similarity search above. We now extend this to take into account multiple succeeding (or preceding) frames. We take the distance between the ith frame of video v, and the jth frame of video w as

$$D_2(v_i, w_j) = \sum_{k=0}^{K} c_k D\left(v_{i+k}, w_{j+k}\right) \tag{4.2}$$

where K is the number of future frames to take into account, c_k are manually defined weights which determine the importance of these future frames in the metric, and $D(.,.)$ is the metric from (3.3).

A higher K leads to the output video following the videos in the collection more closely. The weighted distance (4.2) is the same as the one used in [60]. In our implementation of the weighted distance, we compute the two most similar repframes to the current frame. Successor frames to w_j are assumed to come from the second-most similar video if w_j is at the end of video w or if a successor frame is not in the same repframe cluster as w_j. If K successor frames meeting these conditions cannot be found, the largest number of successor frames that meet this conditions is used.

4.2.4 Incorporating time weighting to avoid repetition

The algorithm may get trapped into producing short, repetitive loops despite the dynamics incorporated in the distance function. This can happen if one of the keyframes provided is visually similar to a small set of frames but sufficiently dissimilar from all others in the video collection. This can result not only in poor output video but also in the greedy path finding algorithm never terminating since the paths from the two keyframes never meet up.

To avoid the above scenario, we introduce a time-weighting factor into the distance function. A counter t_{w_j} is maintained for each frame w_j, which is initialized to unity for all frames at the outset of video generation. Subsequently, the counter for a frame is incremented by one if the frame is selected. The distance function is modified to include the counter as

$$D_3(v_i, w_j) = t_{w_j} \times D_2(v_i, w_j) \tag{4.3}$$

so that frames which have already been selected into the output video have a correspondingly smaller chance of appearing again.

Since the distance to a frame that has been been selected before is sufficiently increased according to (4.3), the greedy algorithm is directed away from the frames that have already been selected. Hence, the problem of forever looping within a small set of visually similar frames is avoided. The greedy algorithm with time weighting is thus guaranteed to converge, though in the worst case all the frames may get selected a few times.

We initially implemented a variant of the above scenario where the time weight counters are initialized to unity in the beginning, as before, but set to a large value T_{max} when the frame is selected. At each step, all the counters are decremented by one unless they are equal to unity. However, this setup is inferior to the method described above since T_{max} has to be set by hand for each set of input keyframes. Further, there are very few instances when a frame can legitimately appear twice in a video.

4.2.5 Generation of Video Textures

We have stated that our system can operate even without user input. We now describe this mode of operation. If no input keyframes are provided, the system defaults to generating video textures [60]. Video textures are potentially infinite length videos obtained by permuting the order of input frames in a stochastic manner. Hence, the primary difference between this mode and the previously explained mode of our system is that the next frame is chosen from the current frame using a probability distribution rather than by deterministically choosing the most similar one.

We obtain the probability distribution on the next frame by converting the distance function (4.3) into a valid distribution using the following expression

$$p(f) \propto \exp\{-D_3(f_c, f)\} \tag{4.4}$$

where f_c is the current frame and f is the next frame. Since this distribution is conditioned on the current frame, it has to be calculated afresh at every step. Note that the (4.4) has to be normalized to be a valid probability distribution.

In practice, we select all the frames from the clusters corresponding to the three most similar repframes according to the weighted metric (4.3) and compute the probability distribution only on these frames. This is because it would be prohibitively expensive to compute the distribution for thousands of frames in the video collection at every step. In addition, most of the probability values will be negligible in any case since the frames are far removed from the current frame according to the distance metric.

The starting frame of the output video is picked uniformly at random. Subsequently, at each step, the next frame is obtained by randomly sampling from its probability distribution. Since the distance function already incorporates dynamics and time weighting, the generated video is relatively smooth. We use the decaying time-weighting factors for video textures since we do not mind repetitions after a certain period. The time factor initializa-

tion parameter T_{max} determines the interval between repetitions. Even though the video is generally smooth, occasional unrelated jumps can happen due to sampling effects. These sampling effects occur because even frames with relatively low probability can be selected occasionally.

The method above can be considered an extension of the VideoTextures paper [60] to more real world images. The original work used plain image difference as the metric for computing the probability distribution. This is suitable only for images that are extremely similar to each other as was the case with their examples. Further, due to the use of dynamic programming to find the optimal sequence of frames, their system could not scale beyond a few hundred frames. Through our use of SPHs and greedy path merging, we have overcome both these limitations so that our system is more applicable to practical use on personal home videos. This mode of our system can be used to produce visually coherent, but non-monotonous, video screensavers at random from a video collection.

An instance of the evolution of the video texture mode of our system is visualized and explained in Figure 4.7.

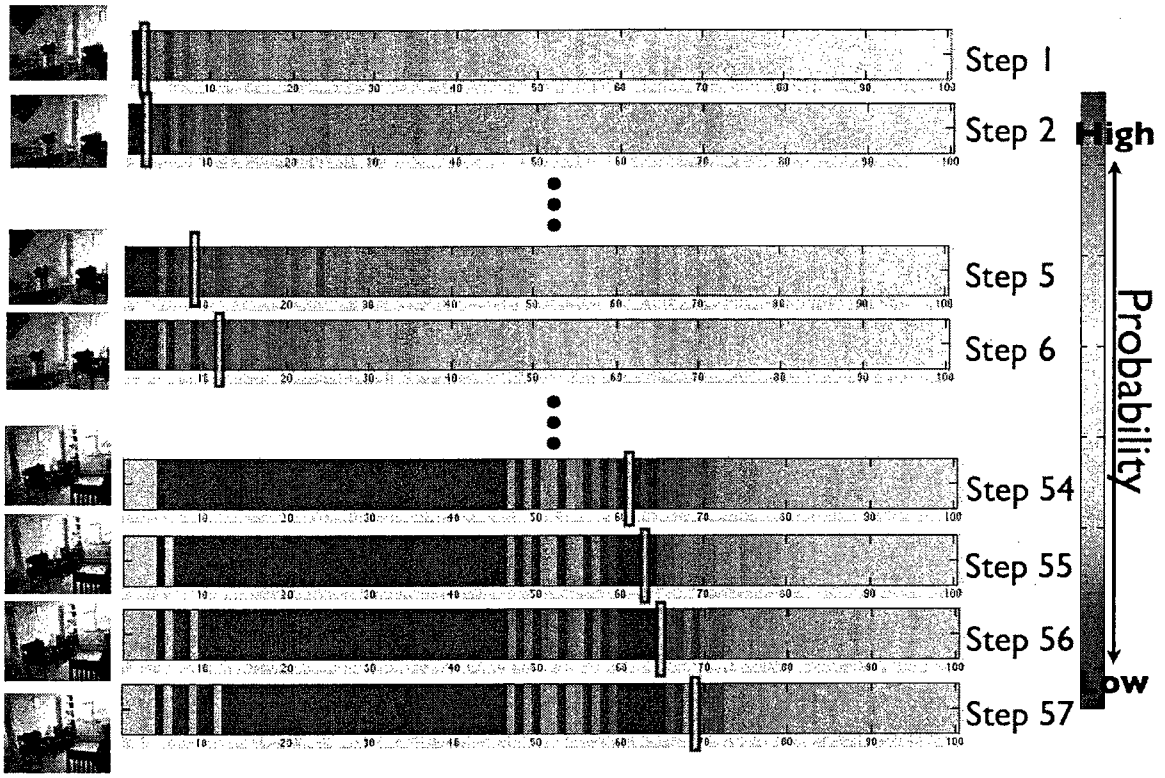


Figure 4.7: An instance of video texture output evolution for a small example with 100 video frames. The probability distribution on the next frame is shown at each step. The current frame location is marked by a rectangle on the distribution and the current image is shown alongside on the left. Red denotes high probability values and blue denotes low values as shown in the color bar on the right. Note the decaying time weighting whereby immediately selected frames get very low probabilities (become blue) but become higher over time. For instance, the frames selected at Steps 1 and 2 have larger probability values again by Step 57 at the bottom of the figure.

Chapter 5

Experiments and Results

In this chapter, we discuss the implementation of our system along with the experiments performed and the corresponding results. Our system is implemented in Matlab and we perform experiments using a collection of 45 videos ranging from 1 minute to 5 minutes in length. Each video contains approximately 1500 to 7000 frames. The videos contain diverse material such as lectures, dancing, skiing, wildlife, and parties. All videos are read as input and their frames are extracted. These frames are reduced to 320x240 pixels in size and stored as images. SIFT features are detected on 16x16 image patches over a grid with spacing of 8 pixels. We compute 200 SIFT clusters during the SPH computation with a pyramid level of 3. The number of frames K in (4.2) is also set to 3. Repframes are computed with the minimum number of frames per cluster set to 75 and the maximum average pairwise distance inside a cluster set to 0.25. This results in 5-7 repframes per video. Preprocessing the entire video collection takes about 70 minutes. Output video generation occurs at 2-3 frames/sec.

5.1 Quantitative Comparison of Features

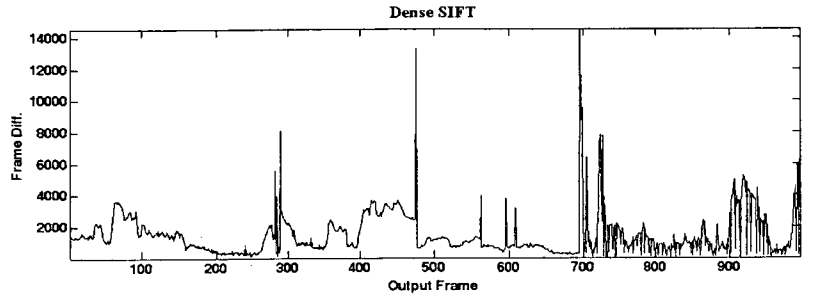
We start with presenting a comparison between the various features that can be used for image representation. In addition to dense SIFT features, we have implemented our system

with SPHs of the Census Transform, color, and texture respectively. We use the Census transform with window size 3 yielding a histogram with 256 bins. Texture histograms are computed by first applying the Leung-Malik filter bank described in Section 3.2.2 to the video frames and subsequently, clustering the output to obtain 200 textons. Color histograms are obtained by clustering the color space into 75 clusters. We observed that a higher number of clusters in the color space did not improve results but slowed down the algorithm considerably. Preprocessing the video collection took about 70 minutes again in the case of texture but was considerably faster for the census transform and color, being 53 and 45 minutes respectively.

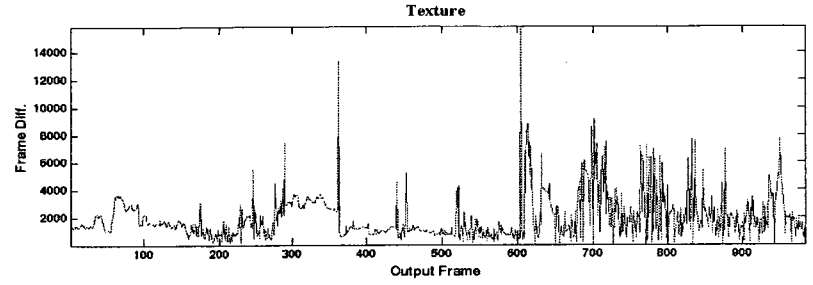
These features are quantitatively compared in the context of our system using the pixel-wise Euclidean distance between successive frames of the output video generated when using each of the features. This is given in Figure 5.1 for an output video of length 1000 frames. Five output videos were produced per feature type and the frame differences were averaged across the five runs to obtain the figure. A sudden jump in the graph indicates an abrupt transition in the output, and the ideal graph would be steady and have low values. As can be seen, dense SIFT performs the best while the Census transform and texture perform almost equally well and slightly worse than SIFT. Using color histograms produces output with many more abrupt transitions than with the other features, and hence, is not suited to our application as expected.

Note that frame difference is not a perfect measure of the image retrieval and sequencing capability since two frames having similar semantic content may have a large frame difference. Due to this reason, Content Based Image Retrieval (CBIR) systems have been tested on human labeled data to obtain a good idea of their performance [63]. However, in our case, since the number of videos is limited and most of the videos are completely different in appearance, this simple measure provides a reasonable estimate of system performance.

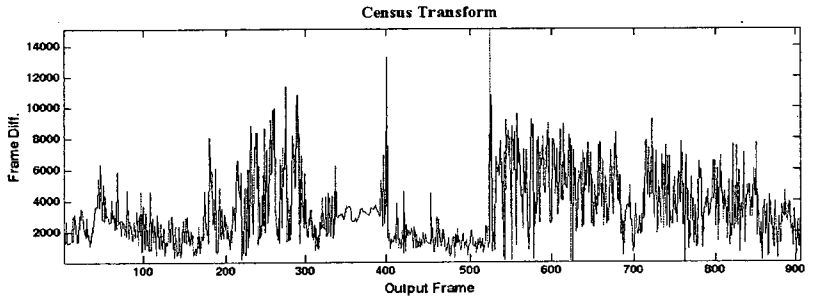
We repeated the same experiment as described above but this time to compare the dif-



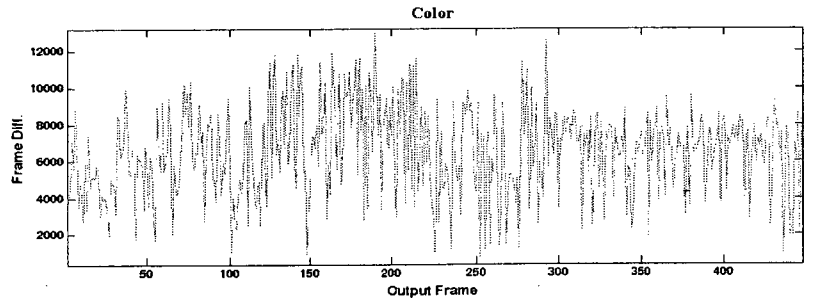
(a)



(b)

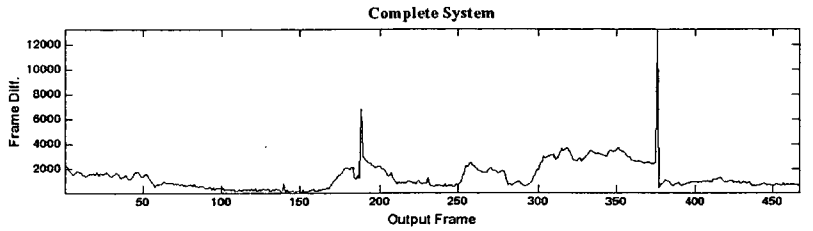


(c)

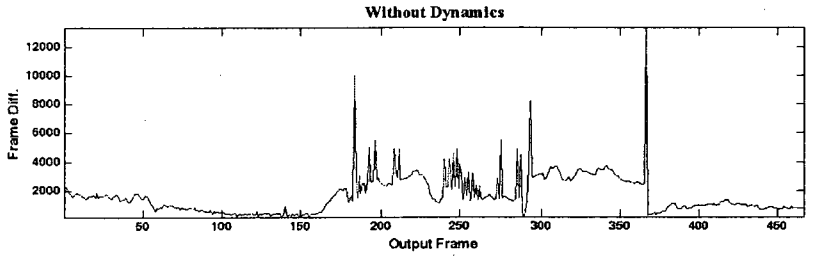


(d)

Figure 5.1: Comparison of the use of different features for automatic non-linear video editing: Norm of frame difference between successive frames of output video from our system using different types of features. The ideal graph has a steady, low value indicating that successively picked frames are visually similar to their predecessors. (a) Dense SIFT performs best, followed by (b) texture (c) census transform using 3x3 patches and (d) color, which performs worst.



(a)



(b)

Figure 5.2: Effect of use of dynamics on smoothness of output video: (a) Complete system with dynamics look-ahead $K = 3$ (b) system without dynamics. Graphs show the norm of frame difference between successive frames of output video from our system. The ideal graph has a steady, low value indicating that successively picked frames are visually similar to their predecessors.

ference between using video dynamics in the similarity metric and not using it. Dynamics were incorporated by comparing a short sequence of frames instead of single frames, as explained in Section 4.2.3. The frame differences for the complete system (using Dense SIFT) when using dynamics and when not using it, are shown in Figure 5.2. The use of dynamics produces videos that are much more visually coherent. Note that the effect of the time weighting (Section 4.2.4) cannot be compared in the same manner since the greedy path finding algorithm is not guaranteed to converge without its use.

5.2 Efficiency of Repframe Computation

We next compare the computational efficiency of the repframe-based system to one that does simple linear search over all the frames to find the next frame which is visually similar. Note that this next frame has to minimize the distance (4.3) that includes the dynamics-

	Time per output frame (secs)
Simple linear search	**1.13**
Repframe search	**0.59**

Table 5.1: Average per-frame generation time of output video using a simple visual similarity search against our repframe-based method. The repframe method is almost two times faster.

based smoothing factor as well as the time-weighting factor. Due to this reason, nearest neighbor methods that rely on a static metric, such as KD-trees, cannot be used in this case in a straight-forward manner. In practice, we implement the repframe method by fetching a few of the repframes most similar to the current frame rather than just a single repframe. This is because all the frames in the cluster belonging to the most similar repframe may have been picked recently, and hence, may not be good candidates due to the time-weighting factor adding a penalty factor. Table 5.1 shows the generation time per frame of output video for the two systems under consideration, obtained from the total time taken to generate a 1000 frame video using a video collection containing approximately 6000 frames. As can be seen, use of repframes results in a significant speed-up for the system without any decrease in quality of the output videos. The critical factor in the output quality is correct settings of the parameters used in the repframe computation, i.e. the minimum number of frames per cluster and maximum average intra-cluster distance. Unsuitable values for these parameters, i.e. those that lead to too few repframes per video, can result in dissimilar frames being grouped into the same repframe cluster. This in turn can lead to incorrect results in the visual similarity search process.

Note that the computation times in Table 5.1 do not only include the visual similarity search at every step but also time spent in the other parts of the system. However, since these other parts are the same for both the system variants under consideration, the comparison still holds.

5.3 Output Sequences

Figure 5.3 illustrates in detail the evolution of an output video sequence between two keyframes as it passes through the four videos of the collection. All the four videos pertain to the same topic as the two keyframes, i.e. dance, and furthermore, the progression of the output from the first keyframe to the second occurs in a visually pleasing manner. Figure 5.4 shows the distance matrix for the output sequence obtained by providing two more keyframes in addition to the ones shown in Figure 5.3. The output video is thus, an extension of the output sequence shown in Figure 5.3 and is about 1400 frames long.

It can be seen from the distance matrix, computed using the histogram intersection function, that the frames in the output are locally similar to the neighboring frames in the sequence. The major discontinuities in the matrix correspond to transitions between videos whose visual content differs significantly, for example between frames 320 and 340 of Figure 5.3. This provides further visual and quantitative confirmation of the smoothness of the output from our system. Figure 5.5 shows a sample SPH for a frame in one of the input videos computed at three levels of the pyramid.

The progression of another output video sequence is shown in Figure 5.6. This illustrates the behavior of our system when two vastly differing images are provided as keyframes. In this case, a drastic transition from one video to another is unavoidable but the output remains smooth except for this. Note that even though the first input keyframe does not appear in the video collection, video frames containing a Dalmatian are chosen correctly as the subject of the output. In this case, the output transitions smoothly between two Dalmatian videos in the collection. Our system produces coherent videos and is able to select frames resulting in smooth transitions except when such connective frames are not available. It is hoped that larger video collections will give rise to more coherent output videos.

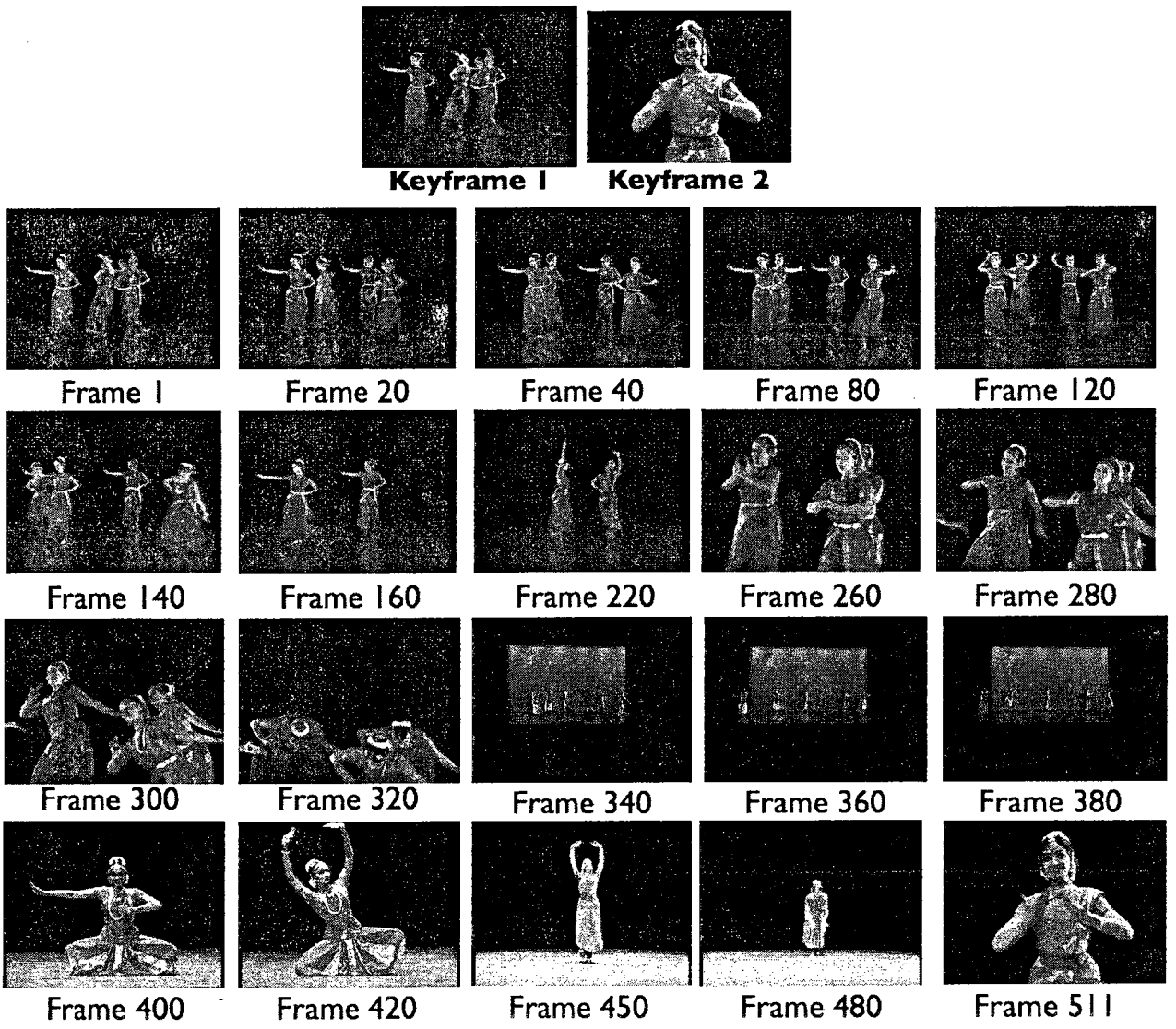


Figure 5.3: Evolution of an output sequence between a pair of input keyframes shown at the top. The output video passes through four videos of the video collection and is slightly longer than 500 frames in length. Note the visually pleasing gradual progression from frames similar to the first keyframe to frames similar to the second.

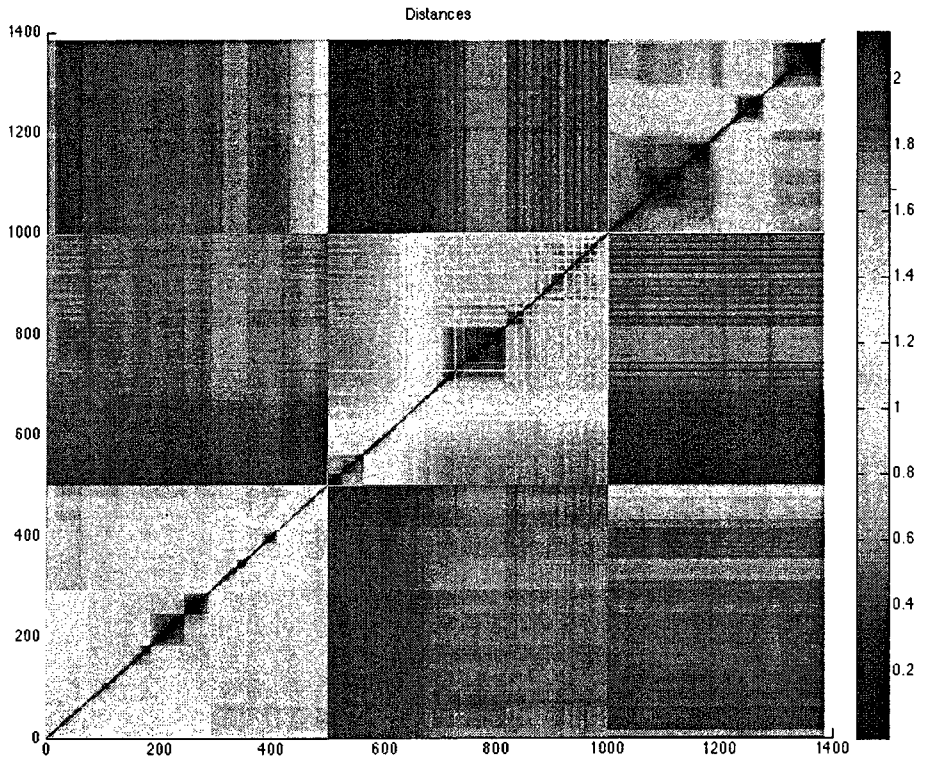


Figure 5.4: Distance matrix showing pairwise distances, computed using the histogram intersection metric, between frames of an output sequence obtained by providing four input keyframes (two more in addition to the ones shown in Figure 5.3. Major discontinuities (horizontal and vertical lines) occur only at a few instances and signal transitions between videos with differing visual content.

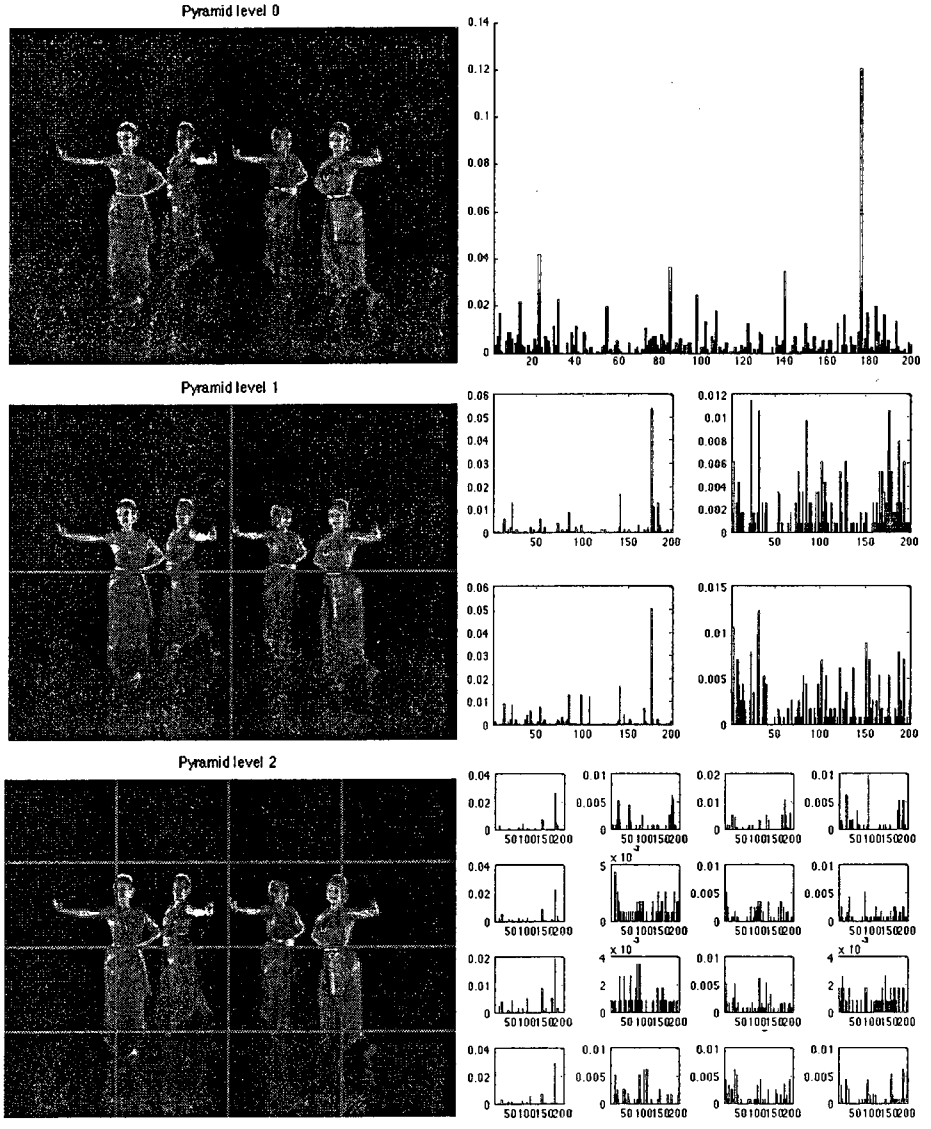


Figure 5.5: The three-level Spatial Pyramid Histogram (SPH) for an output frame

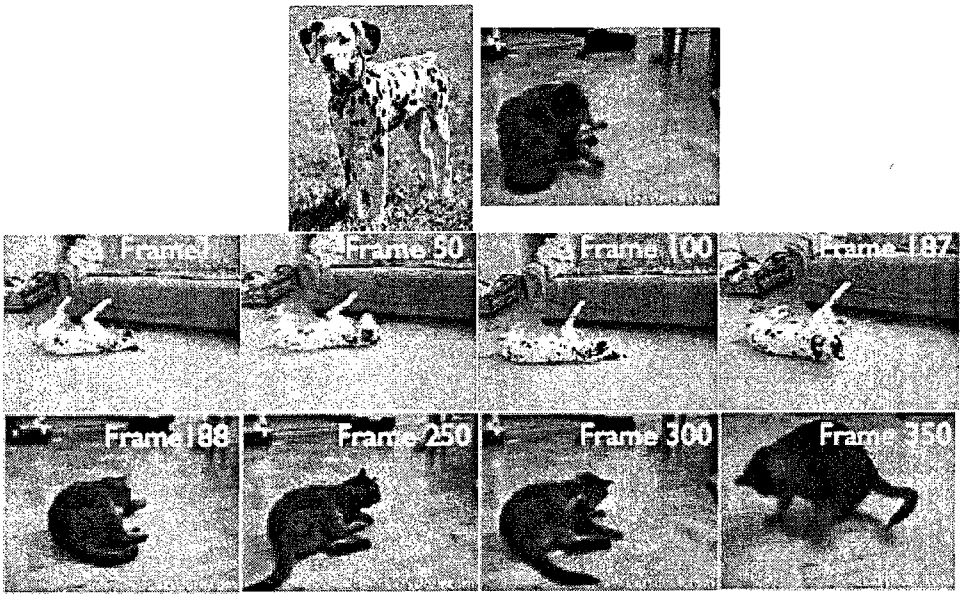


Figure 5.6: Evolution of output video when two vastly differing keyframes are provided as input. Output frames similar to the first keyframe (dalmatian) are selected until the time-weighting factor forces a drastic transition into a video containing frames similar to the second keyframe (cat).

Chapter 6

Discussion

In this work, we have developed an approach to automate the digital video editing process for personal video collections. Our method presents a solution to generate visually coherent videos out of unedited video collections. The content of the output videos is guided by input keyframes given by the user. The output video is generated in a manner that it is non-repetitive and follows the dynamics of the input videos. Thus, we can say that our system is a first step towards a fully automatic non-linear video editing system. When no input is provided, our system generates "video textures" with the content of the output chosen at random.

We use spatial pyramid histograms as our image representation to facilitate computation of visual similarity between images. Frame sequencing is done using a greedy algorithm that finds a "path" through images where successive frames are selected based on the histogram intersection similarity metric. Selection of frames in the sequencing is performed in an efficient manner through the use of repframes, which represent similar clusters of frames in each video. These repframes are computed using hierarchical clustering of the spatial pyramid histograms, and frame selection is speeded up by incorporating them in a two-level search procedure.

We demonstrated in our experiments that the use of dense SIFT features results in su-

perior performance in comparison to other features such as texture and census transform. The use of repframes significantly speeds up video generation at runtime. We also provided instances of output video that provide evidence for the visual coherence of the system output. In light of the above, we conclude that our system satisfies the objective of this thesis, restated below from Section 1.1

> Develop algorithms and build a system for automatic creation of visually coherent videos. The two inputs to the system are (1) a collection of video clips containing only image information, and (2) an optional set of keyframes that determine the content of the output video.

6.1 Applications

We now discuss some possible applications of the current work, and extensions which can be implemented as future work. The system currently can be considered to have basic functionality and can be enhanced in many ways to yield various advanced applications. Our system allows a user to select frames containing specific visual content. For example, if the user requires a short video summary to be generated from a collection of video clips taken during a visit to a zoological garden, this is possible with our system. This can be thought of as a content-specific view of the video collection, where the need to wade through the complete video collection manually, is avoided. The advantage lies in intuitive organization and ease of access. An additional advantage is that the privacy of the other parts of the user's video collection is maintained. A web-based video editing system that uses a common collection of videos from an online website is also a possibility.

Our system can also be used for making a video collage [38], simply by showing multiple output videos simultaneously. Video collage research also includes methods for content selection so that the various videos shown at the same time have similar themes. This could be an avenue for future work based on our system. Automated storytelling as an extension

of our system has been introduced in Chapter 1. This type of work can include on-demand documentaries and adaptive movies, were the user can adapt the narrative and sequence according to his/her requirements. This also enables artistic expression by the user and makes these genres interactive rather than passive forms of entertainment. The use of huge video collections such as that of online video sharing websites will drive these types of applications.

Another application is that of similarity-based video search on the internet. All video search engines, such as Youtube, currently only use text tags for this purpose. However, this can make searching for video very difficult if the essential tags of the video, such as name of characters, location etc are not known but only some content of the video, such as a car chase, is remembered. Additionally, text tags may overlap in meaning, for instance searching for videos using the keyword 'Jaguar' yields both videos of felines and of cars. In such cases, it is more intuitive to upload one or more images similar in appearance to the video being searched for and conduct the video search based on this information. This is exactly what our system does.

6.2 Future Work

At the algorithmic level, we plan to make our system more efficient through the use of nearest neighbor methods for the similarity search. This is not straight-forward since the metric changes at every query due to the use of dynamics-based smoothing and time-weighting. Hence, the most similar image obtained by SPH comparison alone, or even the top 50 or 100 images, may not be the ones we require because their distances become too high when these factors are considered. Currently, the system also does not provide a means to control the length of the output video except through a possible post-processing step. Modifying the greedy path finding algorithm to take into account such constraints would lead to a more usable system. It is also future work to post-process our output videos to improve

their quality. This can include implementing shot transitions such as fade-out fade-ins and frame blending in the few cases where successive frames in the output are not visually similar.

We have not used annotations in this work. However, text is still the interface of choice, both on the internet and for people to request videos by content. It is easier to type in a few keywords to drive the video generation rather than provide input keyframes as we have done here. However, since automatic object detection and image categorization are still in their nascent stage, annotation has to be provided manually. This could, at its simplest, take the form of providing a context-based search wherein a few topical keywords are assigned to each video in the video collection rather than frame-level annotation. When output video is requested, only the video clips matching the required annotation need to be used for the video generation process. This will also make the visual similarity search process faster.

Object-level annotation can also be used to improve output video quality. If the types of some of the objects occurring in a video are known, this can be used to create videos with or without those objects and by combining multiple objects in object-level logical operations (table AND chair, dog OR cat, etc.). Parts of frames can also be put together to generate completely synthetic videos, even at the frame level. This is similar to Semantic Photo Synthesis [25] for videos. Geotagging is another popular form of annotation available nowadays for a huge number of images from websites such as Panoramio. Geo-tagged image and video databases could also be used to produce videos of selective locations.

Bibliography

[1] E. Ardizzone, M. La Cascia, A. Avanzato, and A. Bruna. Video indexing using mpeg motion compensation vectors. In *IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 725–732, 1999.

[2] F. Bashir, S. Khanvilkar, D. Schonfeld, and A. Khokhar. Chapter 6:multimedia systems: Content-based indexing and retrieval. In W.K. Chen, editor, *The Electrical Engineering Handbook*. Academic Press, 2004.

[3] A. Bocker, S. Derksen and E. Schmidt and A. Teckentrup, and G. Schneider. A hierarchical clustering approach for large compound libraries. *Journal of Chemical Information and Modeling*, 45(4):807–815, 2005.

[4] A. Bosch, A. Zisserman, and X. Munoz. Scene classification via plsa. In *Eur. Conf. on Computer Vision (ECCV)*, pages 517–530, 2006.

[5] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1–8, 2007.

[6] G.-H. Cha, X. Zhu, D. Petkovic, and C.-W. Chung. An efficient indexing method for nearest neighbor searches in high-dimensional image databases. *IEEE Transactions On Multimedia*, 4(1):76–87, 2002.

[7] S.-F. C. Chen, W. Meng, H. J. Sundaram, and H. Di Zhong. A fully automated content-based video search engine supporting spatiotemporal queries. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):602–615, 1998.

[8] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, 2008.

[9] M. N. Do and M. Vetterli. Wavelet-based texture retrieval using generalized gaussian density and kullback-leibler distance. *IEEE Trans. Image Processing*, 11(2):146–158, 2002.

[10] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning object categories from google's image search. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1816–1823, 2005.

[11] J. Foote, M. Cooper, and A. Girgensohn. Creating music videos using automatic media analysis. In *ACM international conference on Multimedia*, pages 553–560, 2002.

[12] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1022–1029, 2009.

[13] A. Girgensohn, J. Boreczky, P. Chiu, J. Doherty, J. Foote, G. Golovchinsky, S. Uchihashi, and L. Wilcox. A semi-automatic approach to home video editing. In *ACM Symposium on User Interface Software and Technology*, pages 81–89, 2000.

[14] D. B. Goldman, C. Gonterman, B. Curless, D. Salesin, and S. M. Seitz. Video annotation, navigation, and composition. In *ACM symposium on User Interface Software and Technology*, pages 3–12, 2008.

[15] K. Grauman and T. Darrell. The pyramid match kernel: Efficient learning with sets of features. *Journal of Machine Learning Research*, 8:725–760, April 2007.

[16] E. Hadjidemetriou, M. D. Grossberg, and S. K. Nayar. Multiresolution histograms and their use for recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 26(7):831–847, July 2004.

[17] A. Hanjalic. Shot-boundary detection: Unraveled and resolved? *IEEE Transactions on Circuits And Systems For Video Technology*, 12(2):90–105, 2002.

[18] A. G. Hauptmann, M. G. Christel, W.-H. Lin, B. Maher, J. Yang, R. V. Baron, and G. Xiang. Summarizing bbc rushes the informedia way. In *TRECVID BBC Summarization Workshop*, 2007.

[19] X.-S. Hua, L. Lu, and H.-J. Zhang. Optimization-based automated home video editing system. *IEEE Transactions On Circuits And Systems For Video Technology*, 14(5):572–583, 2004.

[20] R. A. Hummel. Image enhancement by histogram transformation. *Computer Graphics and Image Processing*, 6:184–195, 1977.

[21] P. Indyk. Chapter 39 : Nearest neighbors in high-dimensional spaces. In J. E. Goodman, J. O'Rourke, and P. Indyk, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, Boca Raton, FL, 2^{nd} edition, 2004.

[22] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *SIGGRAPH*, pages 277–286, 1995.

[23] A. Jain and F. Farrokhnia. Unsupervised texture segmentation using gabor filters. *International Conference on Systems, Man and Cybernetics*, 24(12):1167–1186, 1990.

[24] N. Jeho and H. T. Ahmed. Dynamic video summarization and visualization. In *Proc. ACM Intl. Conference on Multimedia*, pages 53–56, 1999.

[25] M. Johnson, G. Brostow, J. Shotton, O. Arandjelovic, V. Kwatra, and R. Cipolla. Semantic photo synthesis. *Computer Graphics Forum*, 25(3):407–413, 2006.

[26] K. Konstantinidis and I. Andreadis. Performance and computational burden of histogram based color image retrieval techniques. *Journal of Computational Methods in Science and Engineering*, 5(2):141–147, 2005.

[27] D. I. Kosmopoulos, A. Doulamis, A. Makris, N. Doulamis, S. Chatzis, and S. E. Middleton. Vision-based production of personalized video. 24(3):158–176, 2009.

[28] S. Lazebnik and M. Raginsky. Supervised learning of quantizer codebooks by information loss minimization. *IEEE Trans. Pattern Anal. Machine Intell.*, 31(7):1294–1309, 2009.

[29] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2169–2178, 2006.

[30] S. H. Lee, C. H. Yeh, and C. C. Kuo. Home video content analysis for mv-style video generation. In *International Symposium on Electronic Imaging*, 2005.

[31] T. Leung and J. Malik. Recognizing surfaces using three-dimensional textons. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1010–1017, 1999.

[32] J. Li and J. Z. Wang. Studying digital imagery of ancient paintings by mixtures of stochastic models. *IEEE Trans. Image Processing*, 13(3):340–353, 2004.

[33] L.-J. Li, R. Socher, and L. Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2036–2043, 2009.

[34] T. Li and T. Mei. Learning optimal compact codebook for efficient object categorization. In *IEEE Workshop on Applications of Computer Vision*, pages 1–6, 2008.

[35] R. Lienhart. Abstracting home video automatically. In *Proc. ACM International Conference on Multimedia*, pages 37–40, 1999.

[36] R. Lienhart. Dynamic video summarization of home video. In *Proc. SPIE Storage and Retrieval for Media Databases*, pages 378–386, 2000.

[37] C. Lindley and F. Nack. Hybrid narrative and associative/categorical strategies for interactive and dynamic video presentation generation. *New Review of Hypermedia and Multimedia*, 6:111–145, 2000.

[38] X. Liu, T. Mei, X.-S. Hua, B. Yang, and H.-Q. Zhou. Video collage. In *International Conference on Multimedia*, pages 461–462, 2007.

[39] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. J. of Computer Vision*, 60(2):91–110, 2004.

[40] Y. F. Ma, L. Lu, H. J. Zhang, and M. J. Li. A user attention model for video summarization. In *Proc. ACM International Conference on Multimedia*, pages 533–542, 2002.

[41] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *Intl. J. of Computer Vision*, 43:7–27, June 2001.

[42] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conf. (BMVC)*, pages 414–431, 2002.

[43] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *Eur. Conf. on Computer Vision (ECCV)*, volume 1, pages 128–142, 2002.

[44] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *Intl. J. of Computer Vision*, 65(1/2):43–72, 2005.

[45] F. Monay and D. Gatica-Perez. Plsa-based image auto-annotation: constraining the latent space. In *ACM International Conference on Multimedia*, pages 348–351, 2004.

[46] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Neural Information Processing Systems (NIPS)*, 2006.

[47] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[48] C. L. Novak and S. A. Shafer. Anatomy of a color histogram. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 599–605, 1992.

[49] N. Omoigui, L. He, A. Gupta, J. Grudin, and E. Sanoki. Time-compression: System concerns, usage, and benefits. In *In Proc. Conf. Human Factors in Computing Systems*, pages 136–143, 1999.

[50] W.-T. Peng, Y.-H. Chiang, W.-T. Chu, W.-J. Huang, W.-L. Chang, P.-C. Huang, and Y.-P. Hung. Aesthetics-based automatic home video skimming system. In *Proceedings of International Multimedia Modeling Conference (MMM'08)*, pages 186–197, 2008.

[51] E. G. M. Petrakis. Fast retrieval by spatial structure in image databases. *Journal of Visual Language and Computing*, 13(5):545–569, 2002.

[52] N. Pinto, D. D. Cox, and J. J. DiCarlo. Why is real-world visual object recognition hard? *PLoS Computational Biology*, 4(1):e27, 2008.

[53] A. Quddus and O. Basir. Wavelet-based medical image registration for retrieval applications. In *International Conference on BioMedical Engineering and Informatics*, volume 2, pages 301–305, 2008.

[54] Z. Rasheed and M. Shah. A graph theoretic approach for scene detection in produced videos. In *Multimedia Information Retrieval Workshop*, 2003.

[55] S. Ravela, R. Manmatha, and E. M. Riseman. Image retrieval using scale-space matching. In *Eur. Conf. on Computer Vision (ECCV)*, pages 273–282, 1996.

[56] C. Rocchi and M. Zancanaro. Rhetorical patterns for adaptive video documentaries. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 324–327, 2004.

[57] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. Segmenting, modeling, and matching video clips containing multiple moving objects. *IEEE Trans. Pattern Anal. Machine Intell.*, 29(3):477–491, 2007.

[58] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Intl. J. of Computer Vision*, 40:99–121, 2000.

[59] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Trans. Pattern Anal. Machine Intell.*, 19(5):530–535, 1997.

[60] A. Schödl, R. Szeliski, D.H. Salesin, and I. Essa. Video textures. In *SIGGRAPH*, pages 489–498, 2000.

[61] F. Schroff, C. L. Zitnick, and S. Baker. Clustering videos by location. In *British Machine Vision Conf. (BMVC)*, 2009.

[62] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, Cambridge, 2006.

[63] N. V. Shirahatti and K. Barnard. Evaluating image retrieval. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 955–961, 2005.

[64] J. Shotton, A. Blake, and R. Cipolla. Contour-based learning for object detection. In *Intl. Conf. on Computer Vision (ICCV)*, pages 503–510, 2005.

[65] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling appearance, shape and context. *Intl. J. of Computer Vision*, 81(1):2–23, 2009.

[66] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1470–1477, 2003.

[67] Josef Sivic, Bryan Russell, Alexei A. Efros, Andrew Zisserman, and Bill Freeman. Discovering objects and their location in images. In *Intl. Conf. on Computer Vision (ICCV)*, pages 370–377, 2005.

[68] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Machine Intell.*, 22(12):1349–1380, 2000.

[69] J. R. Smith and S.-F. Chang. Local color and texture extraction and spatial query. In *IEEE International Conference on Image Processing*, pages 1011–1014, 1996.

[70] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3D. In *SIGGRAPH*, pages 835–846, 2006.

[71] A. Stefanidis, P. Partsinevelos, P. Agouris, and P. Doucette. Summarizing video datasets in the spatiotemporal domain. In *Proc. 11th Int. Workshop on Database and Expert Systems Applications*, pages 906–912, 2000.

[72] E. Sudderth, A. Torralba, W. Freeman, and A. S. Willsky. Describing visual scenes using transformed objects and parts. *Intl. J. of Computer Vision*, 77:291–330, March 2008.

[73] M. Swain and B. Ballard. Color indexing. *Intl. J. of Computer Vision*, 7(1):11–32, 1991.

[74] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 30(11):1958–1970, 2008.

[75] L. Torresani, V. Kolmogorov, and C. Rother. Feature correspondence via graph matching: Models and global. In *Eur. Conf. on Computer Vision (ECCV)*, pages 596–609, 2008.

[76] T. Tuytelaars and L. Van Gool. Content-based image retrieval based on local affinely invariant regions. In *International Conference on Visual Information Systems (VISUAL)*, pages 493–500, 1999.

[77] M. Varma and A. Zisserman. A statistical approach to texture classification from single images. *Intl. J. of Computer Vision*, 62:61–81, April 2005.

[78] H. Wang, N. Xu, R. Raskar, and N. Ahuja. Videoshop: A new framework for spatio-temporal video editing in gradient domain. *Graphical Models*, 69(1):57–70, January 2007.

[79] J. Wang, C. Xu, E. Chng, H. Lu, and Qi Tian. Automatic composition of broadcast sports video. *Multimedia Systems*, 14(4):179–193, September 2008.

[80] P. P. Wang, T. Wang, J. Li, and Y. Zhang. Information-theoretic content selection for automated home video editing. In *IEEE International Conference on Image Processing*, pages IV:537–540, 2007.

[81] Y. H. Wang. Image indexing and similarity retrieval based on spatial relationship model. *Inf. Sci. Inf. Comput. Sci.*, 154(1):39–58, 2003.

[82] B. Wu and R. Nevatia. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Intl. Conf. on Computer Vision (ICCV)*, pages I:90–97, 2005.

[83] J. Wu. *Visual Place Categorization.* PhD thesis, College of Computing, Georgia Institute of Technology, 2009.

[84] J. Wu and J. M. Rehg. Where am i: Place instance and category recognition using spatial pact. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[85] J. Wu and J. M. Rehg. Beyond the euclidean distance: Creating effective visual codebooks using the histogram intersection kernel. In *Intl. Conf. on Computer Vision (ICCV)*, 2009.

[86] Y. Yitzhaky and N. S. Kopeika. Identification of blur parameters from motion blurred images. *Graphical Models and Image Processing*, 59(5):310–320, 1997.

[87] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Eur. Conf. on Computer Vision (ECCV)*, volume 2, pages 151–158, 1994.

[88] M. Zancanaro, C. Rocchi, and O. Stock. Automatic video composition. In *Smart Graphics*, pages 217–222. Springer Berlin, 2003.

[89] H. Zettl. *Sight, sound, motion: applied media aesthetics*. Wadsworth, 3rd edition, 2004.