SPATIO-TEMPORAL BROWSING OF MULTIMEDIA PRESENTATIONS

By

Ramazan Savaş Aygün

May 2003

A DISSERTATION SUBMITTED TO THE

FACULTY OF THE GRADUATE SCHOOL OF STATE

UNIVERSITY OF NEW YORK AT BUFFALO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Acknowledgments

I would like to express my deep appreciation to my major advisor, Professor Aidong Zhang, not only for the academic guidance but also for the encouragement she has given to me in all these years. It is her continuous help, motivation, and insightful advice that made this dissertation possible. I have enjoyed the active atmosphere in the Multimedia and Database Lab led by her and the freedom she has guaranteed to the students. I also want to show my respects to her for her incredible enthusiasm for research.

I am deeply grateful to Professor Jan Chomicki, one of the committee members. He has been very helpful in the development of the synchronization model and motivated me for model checking of the synchronization model. I have obtained invaluable advice and helpful comments from him.

Many thanks to Professor Bharat Jayaraman, our department Chair and the other committee member, for his insightful advice and comments on my research work.

I would also like to express my deep appreciation to Prof. Gültekin Özsoyğlu for serving as the outside reader of my dissertation and for his invaluable comments. Many thanks to him for the great support.

During my research, I had the pleasure to cooperate with many good friends including Dr. Gholamhosein Sheikholeslami, Dr. Dantong Yu, Dr. Markus Mielke, Dr. Yuqing Song, Dr. Lei Zhu, Mr. Wei Wang, Mr. Yimin Wu, Mr. Pengjun Pei, Mr. Li Zhang, Ms. Chun Tang, Mr. Daxin Jiang, Mr. Yong Shi, Mr. Dokyun Kim, and Mr. Adrian Rusu.

I would like to thank the Department of Computer Science and Engineering in the State University of New York at Buffalo for the financial assistantship during my study. I also thank all the faculty and staff members of our department.

Special thanks to my dear wife, Emel, for her great support from every aspect of everyday life and research! She had been at the right time at the right place where I needed her very much. I have no doubt that the completion of my dissertation would take longer without her help. She made me realize that *love* is the root of all good deeds and the path to the truth.

I am indebted to my family members because I seldom have time to take care of them due to the heavy load of research in these years. I hope I will be a person that my family will be proud of.

Abstract

Emerging applications like asynchronous distant learning and collaborative engineering require organization of media streams as multimedia presentations. The browsing of presentations enables interactive surfing of the multimedia documents. We propose spatio-temporal browsing of multimedia presentations in the sense that browsing can be performed both in the spatial and temporal domain.

The spatial browsing is provided by incorporation of camera controls like panning, tilting, and zooming. Panoramic images enable a kind of browsing by storing the image at high resolutions from various angles. However, the generation of high resolution sprite (mosaic) from digital video is not an easy task. Since the video data may also exist in a compressed format, new features like boundaries have to be extracted from the compressed video. We consider compressed data that is generated by Discrete Cosine Transform (DCT), which has been used in MPEG-1, MPEG-2, MPEG-4, and H263.1. Global Motion Estimation (GME) has been improved for videos where motion does not occur frequently. Motion sensors, which are sensitive pixels to motion, are proposed to indicate the existence of motion and yield quick approximation to the motion. Motion sensors reduce the amount of computations of the hierarchical evaluation of low-pass filtered images in iterative descent methods. The generated sprites are usually more blurred than original frames due to image warping stage and errors in motion estimation. The temporal integration of images is performed using the *histemporal* filter based on the histogram of values within an interval. The initial frame in the video sequence is registered at a higher resolution to generate high resolution sprite. Instead of warping of each frame, the frames are warped into the sprite at intervals to reduce the blurring in the sprite. We also introduce a new sprite called *conservative sprite* where new pixels are exclusively mapped on the sprite during temporal integration phase. The *sprite pyramid* is introduced to handle sprite at different resolutions. To measure the quality of the sprite, a new measure called sharpness is used to estimate the blurring in the sprite. The generated sprite is used for spatial browsing.

On the other hand, temporal browsing is closely related with the synchronization of different streams. The power of synchronization models is limited to the synchronization specifications and user interactions. The proposed synchronization model is an event-based model that can handle time-based actions while enabling user interactions like backward and skip. The synchronization model processes the synchronization rules based on Event-Condition-Action (ECA) rules. Since the structure of a synchronization rule is simple, the manipulation of the rules can be performed easily in existence of user interactions.

The synchronization model uses Receiver-Controller-Actor (RCA) scheme to execute the rules. In RCA scheme, receivers, controllers, and actors are objects to receive events, to check conditions, and to execute actions, respectively. The synchronization rules can easily be regenerated from SMIL expressions. The deduction of synchronization rules is based on author's specification. A middle layer between the specification and the synchronization model assists the synchronization model to provide user interactions while keeping the synchronization specification minimal. We call this middle layer as *middle-tier*. The middle-tier for multimedia synchronization handles synchronization rules that can be extracted explicitly from the user specification and synchronization rules that can be deduced implicitly from explicit synchronization rules. The synchronization model also generates a virtual timeline to manage the user interactions that change the course of the presentation. The verification and correctness of schedules are also important. The general methods to check the correctness of a specification are theoretical verification, simulation, and testing. Model checking is a technique that automatically detects all the states that a model can enter and checks the truthness of well-formed formulas. Moreover model checking can present contradictory examples if the formulas are not satisfied. PROMELA/SPIN [45, 44] tool has been used for model checking to check LTL (Linear Temporal Logic) formulas. These formulas can automatically be generated and verified.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The management of multimedia presentations have gained great significance as applications such as video teleconferencing, video-on-demand, educational learning and tutoring, asynchronous distant learning, and collaborative engineering emerged. The first presentations were similar to TV broadcast where users have to be ready at the beginning of the presentation and could not interact with the presentation. As the technology for the multimedia data improved, the multimedia presentations are able to be stored and accessed at different times by various users. There have been challenging problems confronted when multimedia presentations enable user interactions and the multimedia data are transmitted over networks shared by many users. The loss of the data over the networks requires a comprehensive specification of the synchronization requirements. The user interactions that change the course of a presentation either increased the complexity of the specification or are not allowed. Spatial browsing of video documents enables the viewers to visualize interesting objects from their perspective. Spatial browsing requires accurate sprite generation from video. Sprite generation has to be performed in compressed domain if it is possible. Low resolution sprites and blurred sprites are results of traditional sprite generation techniques that cannot be applied for spatial browsing.

The browsing of multimedia presentations in distributed environments is significant to present correct presentations effectively. In this dissertation, browsing is considered as spatio-temporal browsing since it can be performed in the spatial and temporal domain. The spatial browsing provides browsing of videos and images in the presentations. Spatial browsing supplies browsing of video without actually stopping and rewinding the video. On the other hand, temporal browsing enables investigation of specific sections within a presentation. Spatial browsing depends on accurate sprite generation whereas temporal browsing requires robust and flexible multimedia synchronization model.

1.1 Sprite Generation

The size of multimedia presentations is large due to the size of video streams that are presented. Video standards like H263.1, MPEG-1, and MPEG-2 have been proposed to reduce the redundancy in video streams that concentrate on spatial redundancy in a single frame and temporal redundancy in consecutive frames. The introduction of MPEG-4 [87] has revealed that there is still redundancy in the object domain. MPEG-4 has increased the motivation for the sprite generation and video object segmentation. A camera shot can only capture a window from a large scene. The process of generation of the big picture from shots taken consecutively or at intervals from calibrated or uncalibrated points is called *sprite generation*. A sprite is an image of aligned pixels belonging to a video object that is visible throughout a video scene [74]. Once the sprite is generated and objects are segmented, the video objects can be layered on the regions of a sprite. The sprite generation and segmentation of video objects reduce the size of video data significantly.

MPEG-4 uses a general term *sprite* instead of mosaic for the big picture of video objects. The term background is also used instead of the term mosaic. The representations of mosaics are investigated in [48]. A mosaic image is constructed from image sequences giving a panoramic view of the scene. A *static mosaic* is obtained by applying some temporal filters (mean, median). Static mosaic represents the whole picture without the moving objects. *Dynamic mosaic* is the current frame with the most recent updated scene. This can be considered as an extension of the most recent frame with its surrounding. If the mosaic also includes the objects of all the frames, it is called *synopsis mosaic*. Synopsis mosaic is used to display how the moving objects displace their locations in the whole picture. Mosaic generation is composed of image alignment, image integration, and residual estimation.

Mosaic can be constructed using average temporal intensity, temporal median, weighted temporal median or average, or combination of these. Image alignment may be performed using the 2D motion model or 3D motion model. The frames may be aligned from frame to frame, frame to mosaic, or mosaic to frame.

The video data may exist as raw data where frames consist of pixel values or in compressed form as in MPEG standards. Most of the previous compression methods utilize Discrete Cosine Transform (DCT) [37]. Decompressing data and processing in the pixel domain require huge amount of processing due to the size of videos. Compressed data provide additional features in the frequency domain that decreases the amount of processing. Since most of the compression is performed using DCT, extraction of features for compressed data using DCT may decrease the amount of processing and provide more information for the sprite generation and object segmentation. The sprite generation and object segmentation require camera motion detection. Camera motion detection, sprite generation, and object segmentation have been closely studied.

1.2 Multimedia Synchronization

Multimedia synchronization deals with the synchronization of media streams in a presentation. Synchronization is classified as *intra-stream* synchronization and *inter-stream* synchronization. The intra-stream synchronization manages the presentations of streams at a required rate (e.g. playing video 30frames/second). The inter-stream synchronization manages the relationships among the streams. There are two types of inter-stream synchronization: *fine-grained* synchronization and *coarse-grained* synchronization. Fine-grained synchronization requires a tight synchronization between each segment of two streams like a lip-synchronization between audio and video. Most of the research in fine-grained synchronization aims at lip-synchronization between audio and video [92]. Coarse-grained synchronization handles the relationships among streams and determines when streams start and end. Synchronization specification languages like SMIL [88] focuses on the synchronization requirements for coarse-grained synchronization.

There have been various approaches to deal with modeling of multimedia presentations and management of browsing capabilities. The models in the previous work can be classified as either time-based or event-based. The advantage of time-based models is the easy specification of the temporal layout of the media objects in a presentation. The disadvantage is the limitations with the specification and application of the synchronization requirements. Even if the complex synchronization requirements are ignored in time-based models, the user interactions such as pause, play, resume, and (fast-slow) forward need to update the start time and durations of streams. The management of the backward presentations or skip operations is not very complex since the start and ending times of each media object are known. The time of media objects for a backward presentation, and the media objects that are active at a specific point can be figured out through the time relations. If time-based models enforce constraints to satisfy synchronization requirements, the skip and backward functions also become difficult to handle.

1.3 Spatial Browsing

Spatial browsing for a recorded video enables the users to view interesting scenes and objects in the environment from the perspective of users. Although spatial browsing of recorded video was not considered, there has been research on building blocks of spatial browsing. The fundamental issue is accurate sprite or mosaic generation from the video. Sprites help reduce the size of original video streams that are presented.

In recent years, camera control has been studied by using different strategies at hardware and software level. Although temporal browsing of video is supported by many applications, incorporation of spatial browsing has been delayed due to the late improvements in global motion estimation, sprite generation, and manufacturing of special cameras. It is hard to determine the most interesting objects and scenes that are favored by all viewers in the environment. In traditional applications, cameramen are responsible for capturing the most interesting events, activities, and scenes. Therefore, the stored video contains the scene recorded from the perspective of the cameramen. It has been realized that what has been important to the cameramen is not necessarily the most important object or scene for all viewers. There are a couple of ways to handle the camera control: transforming to a virtual environment, automating the camera to follow the pre-defined objects, and using advanced camera like panoramic camera and then extracting important regions.

Video object segmentation, which emphasizes on partitioning the video frames into semantically video objects and the background, has become a significant issue for the effective manipulation of MPEG-4 and MPEG-7. However, there has been significant amount of video that has been compressed using MPEG-1 and MPEG-2, which use Discrete Cosine Transform (DCT) to compress data. Decompressing data and then processing decompressed data is computational intensive. It is important to extract some coarse features to reduce decompression and to use these features in processing.

The sprite generation methods benefit from recent global motion estimation (GME) methods, which yield almost accurate estimation of motion parameters. However, the generated sprites are usually more blurred than original frames due to image warping stage and errors in motion estimation. The transformed coordinates resulting from GME are generally real numbers whereas images are sampled into integer values. Although GME methods generate proper motion parameters, a slight error in motion estimation may propagate to subsequent sprite generation steps.

The ordinary sprite generation techniques focus on camera movement, accurate motion estimation, alignment, and integration. These techniques ignore the resolution of original images and the regenerated images from the sprite are likely to have lower resolutions than the original ones. Especially, if the scenes have finite depth and zoom-in and zoom-out operations occur, the segments of the scene are captured at different resolutions. The traditional sprite generation methods either blur the sprite by integrating lower resolution segments or use unnecessary large storage for the sprite. The previous approaches have not considered the accurate generation and efficient storage of the sprite. Since the sprite generation methods are based on mosaic generation techniques, the sprite generation is a lossy process.

1.3.1 Features for Compressed Domain and Accurate Sprite Generation

The features extracted from the compressed data can be used to identify significant regions before the segmentation. Since the compression technique in MPEG-1, MPEG-2, and MPEG-4 is DCT, we propose a reliable method to extract significant blocks by extracting features about the smoothness and boundaries from DCT compressed blocks. Features about smoothness and structure of boundaries are evaluated to determine the significant blocks from compressed video for object segmentation.

The temporal integration is one of the core parts that introduce extra blurring in sprite generation. The temporal integration of images is performed using the *histemporal* filter based on the histogram of values within an interval. The *histemporal* filter is a generalized filter and keeps the histogram of values that map to a specific interval. The initial frame in the video sequence is registered at a higher resolution to generate high resolution sprite. Instead of warping each frame, the frames are warped into the sprite at intervals. This reduces the blurring in the sprite.

The *sprite pyramid (or layered sprite)* allows efficient storage of images or video clips of overlapping scenes at different resolutions. Moreover, the images or video frames can be reconstructed from the sprite pyramid at the necessary resolutions. A *sprite pyramid* allows the regeneration of the video at the proper resolutions. Each layer of the sprite pyramid corresponds to a different resolution. The sprite pyramid allows the regeneration of different segments at different resolutions as they were captured. The sprite pyramid is created if the scene has finite depth and there are zoom-in and zoom-out operations.

1.3.2 *VideoCruise:* Spatial Browser for Recorded Digital Video

Spatial browsing of video documents enables the viewers to visualize interesting objects from their perspective. We have developed a system, termed as *VideoCruise*, to spatially browse the video documents. VideoCruise requires accurate global motion estimation and accurate sprite generation. Although there have been methods developed to perform these operations, the output of these techniques can only be used with motion compensation. VideoCruise manipulates the sprite and the original frames to allow interactive spatial browsing that enables panning, tilting, and zooming. We do not assume that the scene is predefined. We consider scenes that are captured using a single camera. The user can browse the scene by using camera operations like pan left and right, tilt up and down, and zoom in and out.

1.4 Temporal Browsing

Multimedia presentation management research started with organization of streams that participate in the presentation, and VCR-based user interactions are incorporated at different levels at the later research. Initial models only consider simple interactions like play, pause, and resume. Flexible models do not enforce timing constraints and temporal organization is rather performed by relating events in the presentation. For example, *stream A* *starts after (meets) stream B.* There is no enforcement on media clock time like *stream B* has to end at an instant and at that instant *stream A* has to start. Since there may be delay in the play of *stream B*, to start *stream A* after *stream B* brings flexibility by not enforcing timing constraints. Speed-up and slow-down operations are included at a later stage in initial models. Skip and backward interactions can be incorporated in time-based models. But flexible models could not incorporate these functionalities adequately. There are only few flexible models considering skip operation. These models have some restrictions on the application of skip functionality.

The user interactions that change the course of the presentation are necessary in applications where a detailed investigation of a presentation is required. Distance education and sports presentations are examples where this type of user interactions is needed. Backward and skip are two examples of user interactions of this kind at the lowest level. Skip directs to a new position where some constraints may be violated and the presentation may yield a false presentation. The backward presentation on the other hand requires more information that is usually not specified. We focus on VCR-based user interactions in this dissertation.

1.4.1 Tackling Limitations in Synchronization Models

Defining reverse temporal relationships depending on forward temporal relationships does not solve the problem. For example, the reverse temporal relationship for “A meets B” is

"B meets A". There are a couple of scenarios that "A meets B" holds. Ending of A may start B, beginning of B may end A, or an independent event may end A and start B. This kind of specification is possible in SMIL. In all cases, *meets* relationship holds. Should backwarding A terminate B in backward direction or should ending of B in backward direction backward A or should an independent event end B and backward A? Even for a single relationship, there are many options. "Which option is better than the others?" is one of the questions that we try to answer for event-based models.

There are tools that enable the querying of temporal relationships in multimedia presentations. These tools compare the time intervals of streams and may check the correctness of the specification at a level. Flexible models usually do not consider time instants and may yield different presentations from expected. These models do not aim to achieve deadlines for the playout of streams. In most cases in a distributed environment, these expected deadlines are violated. The author or the user may receive an unsynchronized presentation. Some of the questions that the author should consider are "is there a better specification for the presentation?" and "does the synchronization tool really satisfy the temporal relationships?". The querying tools usually assume perfect on-time presentations of streams. The author cannot verify whether the requirements are really satisfied by the model.

The previous work has limited user interactions, not supported necessary synchronization requirements, or has assumptions on the structure of the multimedia presentations. FLIPS

[83] has powerful synchronization requirements but cannot support backward and it has limited skip functionality. Hurst et. al. [46] require at least one master stream throughout the presentation. NSync [12] does not allow backward and it can only allow skip after user also specifies the operations for each interval of skip. Either robust synchronization models with limited user interactions and/or complex specification or less robust synchronization models while enabling comprehensive user interactions were proposed. The backwarding or skipping is considered at the modeling level using Petri-Nets [73]. This requires the author to model different Petri-Net for each different skip and backwarding. The authors usually do not have enough information about Petri-Nets or they do not want to spend so much time on detailed modeling. In the previous (time-based) models, most of the information given by the user was satisfactory for the model and user interactions. As the synchronization requirements are specified by events, constraints or synchronization expressions, the information to the system became implicit. To overcome this, either the user has to specify more information as in NSync or the functionalities are limited as in FLIPS and PREMO [41]. It is still a question what to specify and how much to specify. We consider SMIL expressions as the level of specification and show how our model is powerful. The reason using SMIL expressions is to show the compatibility of model with SMIL.

1.4.2 RuleSync: A Flexible Rule-Based Synchronization Model with Model Checking

It has been shown that event-based models have been more robust and flexible for multimedia presentations. A disadvantage of the event-based models is the inapplicability of the model in case there is a change in the course of the presentation (like backwarding and skipping). Most of the previous models are based on event-action relationships. The condition of the presentation and participating streams also influence the actions to be executed. Thus event-condition-action (ECA) rules [61], which have been successfully employed in active database systems, are applied to multimedia presentations. Since these rules are used for synchronization, they are termed as *synchronization rules*. Since the structure of a synchronization rule is simple, the manipulation of the rules can be performed easily in existence of user interactions. The synchronization model uses Receiver-Controller-Actor (RCA) scheme to execute the rules. In RCA scheme, receivers, controllers and actors are objects to receive events, to check conditions, and to execute actions, respectively. The synchronization rules can easily be regenerated from SMIL expressions. The authors usually detain from providing extra information for backward and skip operations. This kind of information should be deduced by the model and corrected by the author if it is necessary. This deduction is based on author's specification. We assume that there are reasons behind the way the author makes the specification and deduction of the rules are bound by these reasons. A middle layer between the specification and the synchronization model assists the synchronization model to provide user interactions while keeping the synchronization specification minimal. We call this middle layer as *middle-tier*. The middle tier is previously defined as the logical layer in a distributed system between a user interface or Web client and the database. It is a collection of business rules and functions that generate and operate upon receiving information. The middle-tier for multimedia synchronization handles synchronization rules that can be extracted explicitly from the user specification and synchronization rules that can be deduced implicitly from explicit synchronization rules. The middle-tier reduces the amount of user specification while increasing the power of the synchronization model. The synchronization model also generates a virtual timeline to manage the user interactions that change the course of the presentation.

The verification and correctness of schedules are also important. Mostly, it is the responsibility of the author to verify the requirements in the specification. Today any user without sophisticated information about verification can specify a multimedia presentation. The general methods to check these are theoretical verification, simulation, and testing. These are usually hard for the user (author) to handle. Model checking is a technique that automatically detects all the states that a model can enter and checks the truthness of well-formed formulas. Moreover model checking can present contradictory examples if the formulas are not satisfied. PROMELA/SPIN [45, 44] tool has been used for model checking which checks LTL (Linear Temporal Logic) formulas. These formulas can automatically be generated and verified. We give examples of how our model is built using PROMELA and truthness of formulas is checked by SPIN [44]. The author does not have to know this tool but needs to know what to verify. We have developed a program that the author can verify the truthness of the properties.

1.5 Spatio-Temporal Browsing

In this dissertation, we present solutions for flexible and robust spatio-temporal browsing of multimedia presentations. The dissertation is divided into spatial browsing and temporal browsing. A system called *VideoCruise* is developed to provide spatial browsing. A system called *RuleSync* is developed to temporally browse the multimedia presentations when VCR-type user interactions are enabled in distributed environments. These two systems are integrated with NetMedia [108] and allows flexible and robust spatio-temporal browsing.

We have achieved the following goals as the result of my work:

- a spatial browser to view objects or scenes by generation of accurate sprites from the video
- a synchronization model that flexibly supports interactive multimedia presentations

  in distributed multimedia systems with model checking support,

- to integrate these functionalities in a framework.

The Chapters 2 to 6 are related with spatial browsing and Chapters 7 to 10 are related with temporal browsing. The rest of the dissertation is organized as follows.

- Chapter 2 gives a comprehensive introduction of the latest approaches proposed in the sprite generation.
- Chapter 3 explains the stationary background generation and the extraction of effective features for video object segmentation from MPEG compressed video.
- Chapter 4 explains the improvements in global motion estimation using motion sensors.
- Chapter 5 presents how to generate more accurate sprites by reduction of blurring and discusses the issues for the generation of sprite in video where zoom-in and zoom-out camera operations occur and emphasizes the necessity of a sprite pyramid.
- Chapter 6 explains the spatial browsing tool, *VideoCruise*, and reports the results of our experiments.
- Chapter 7 gives a comprehensive introduction of the latest approaches proposed in multimedia synchronization.

- Chapter 8 presents our flexible and robust synchronization model, *RuleSync*.
- Chapter 9 explains how *RuleSync* handles VCR-type user interactions.
- Chapter 10 describes how model checking techniques can be applied to multimedia synchronization where user interactions are allowed. The model checking is performed using PROMELA/SPIN and also the analysis of the results is reported.
- Chapter 11 provides the conclusion and discusses the future work.

Chapter 2

An Introduction to Sprite Generation

The new video coding standard MPEG-4 [87] enables the content-based functionalities by introducing the concept of video object planes (VOP). The coding of video sequences that are segmented based on video contents; flexible reconstruction, and manipulation of video contents at the decoder have been the primary objective. Thus, video object segmentation, which emphasizes on partitioning the video frames into semantically video objects and the background, becomes a significant issue for the effective manipulation of MPEG-4 and MPEG-7.

Most of the algorithms for sprite generation and object segmentation have to estimate the motion. So, we will start with the motion estimation methods.

2.1 Motion Estimation

There have been 2D and 3D models have been proposed to describe the motion in a scene. The problem is the estimation of the parameters in these models. Motion estimation is usually classified as global motion estimation and local motion estimation. Global Motion Estimation (GME) detects the motion of a camera or huge objects in the video. On the other hand, local motion estimation is related with the motion of objects. The local movements of objects in the video affect GME.

There are different types of motion models that are used in GME depending on the camera operations and the structure of the scene. The perspective camera motion model can be parameterized as:

$$x'_i = \frac{a_0 + a_2 x_i + a_3 y_i}{a_6 x_i + a_7 y_i + 1} \qquad (2.1.1)$$
$$y'_i = \frac{a_1 + a_4 x_i + a_5 y_i}{a_6 x_i + a_7 y_i + 1}$$

where a_0, a_1, a_2, a_3, a_4, a_5, a_6, and a_7 are motion parameters and (x'_i, y'_i) is the transformed coordinate for (x_i, y_i). This model turns into affine motion when $(a_6 = 0,\ a_7 = 0)$, translation-zoom-rotation motion when $(a_4 = -a_3,\ a_5 = a_2,\ a_6 = 0,\ a_7 = 0)$, and translational motion when $(a_2 = 1,\ a_5 = 1,\ a_4 = 0,\ a_5 = 0,\ a_6 = 0,\ a_7 = 0)$. The elementary camera movements are panning, tilting, rotation, and zooming of a camera. If the motion model reflects these parameters, it is easier to extract how the camera moves. To obtain a reliable model, it is assumed that the captured scene is flat concerning the depth along the optical axis and the rotation is small.

In matrix form, affine motion model matrix is denoted with

(2.1.2)
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_2 & a_3 & a_0 \\ a_4 & a_5 & a_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

To perform perspective motion transformation, the Cartesian coordinates need to be converted to homogeneous coordinates.

The error between two frames can be declared as

(2.1.3)
$$\varepsilon = \sum^{N} e_i^2$$

where $e_i = I'(x_i', y_i') - I(x_i, y_i)$, $I(x_i, y_i)$ is the intensity at (x_i, y_i) in the previous frame, and $I'(x_i', y_i')$ is the intensity at the transformed coordinate in the current frame. Error ε is computed for overlapping pixels in two frames.

In matrix form, affine motion estimation can also be written as

(2.1.4)
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_2 & a_3 \\ a_4 & a_5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}.$$

More generally, this can be written as

(2.1.5)
$$v' = Mv + t$$

where M contains the motion parameters for the first matrix and t contains the translational parameters. The relative motion is computed as

(2.1.6) $$v'' = M'(Mv+t)+t' = M'Mv+(M't+t')$$

where v'' is the vector for the new transformed coordinates; M' and t' hold the current motion parameters; and M and t hold the motion parameters up to the current frame.

To increase the robustness of motion estimation, M-estimators [28, 90] are used and the error is expressed as:

(2.1.7) $$\sum^{N} \rho(e_i)$$

where $\rho(e_i) = e_i^2$ in the original formulation. Since this function gives more weight to large errors, it is biased by local motion (which are outliers for global motion). To decrease the effect of outliers, the truncated quadratic motion is used:

(2.1.8) $$\rho(e_i) = \{ \begin{array}{ll} e_i^2, & \text{if } |e_i| \leq \tau \\ 0, & \text{if } |e_i| > \tau \end{array}$$

where τ is a threshold selected according to the histogram of the errors.

Laplacian pyramid [19] is frequently used in hierarchical models. Laplacian pyramid is a hierarchical way of representing an image usually at low resolutions at the high levels and low resolutions at the low levels. A coarse-to fine iterative estimation of motion is proposed in [13]. The motion model is first determined for low resolution images. The motion model is refined after increasing the resolution for the image. Most of the GME techniques concentrate on the accuracy of motion parameters of the chosen motion models [28, 90, 89]. These methods usually include an initial estimation of the subset of the motion parameters and then adjusting of the motion parameters using a hierarchical pyramid of low-pass filtered images.

Instead of using all the pixels in the image, features have also been considered for motion estimation. The feature models for edges, corners, and vertices are presented in [15, 27]. A corner is represented by the amplitude of the wedge, its aperture angle, and a parameter to measure the smoothness of the wedge. Vertices are defined as a superposition of corner models. Triple junctions are defined by 3 adjacent regions.

MPEG uses 2D translational model for matching 16x16 blocks. A block is assumed to move horizontally, vertically, or both. The distance between macroblocks is usually measured by Minimum Square Method (MSE) or Minimum Absolute Distance (MAD).

Optical flow algorithms do not handle image sequences having large motion. The median radial basis function network is used for optical flow estimation and moving object segmentation [17]. The network is trained using the information obtained from the user in the first frames of the video. This requires the training of the network for each video or even for each shot.

2.2 Sprite Generation

The introduction of the MPEG-4 [87] video standard has motivated many research fields like object segmentation and sprite generation. MPEG-4 [87] enables decoding and encoding of layered sprites for the objects and the background. The sprite generation has initially been studied as *mosaic generation* [47, 110, 23, 97]. Mosaic presents a wide picture of the environment that cannot be captured in a single frame. Mosaics are mostly used in content-based retrieval and video compression. The mosaic should include every section that is visible throughout the video sequence. If there is no a priori motion information for a video sequence, the motion has to be estimated between each sequential frame. The previous approaches have not considered the accurate generation and efficient storage of the mosaic. Since the sprite generation methods are based on mosaic generation techniques, the sprite generation is a lossy process.

Sprite generation is closely related with extraction of good features. These features may also need to be extracted in the compressed domain. Feature extraction and edge detection in compressed DCT domain have been studied in [70, 3, 85]. Patterns of the first AC coefficients by zigzag ordering have been used for coarse edge detection [85]. The number of non-zero coefficients has been studied in [3]. First two AC coefficients are used for edge and shape detection in [70]. The general problem with the coarse edge detection problems is that most of the weight is given to coefficients that appear first according to the zigzag ordering.

In traditional mosaicing methods, mosaics are generated by mapping onto a predetermined single space. The order of images is important in mosaic generation. In most cases, the images are mapped according to the first image in the sequence. If the image has the lowest resolution, then a low resolution mosaic is generated and if the first image has the highest resolution, a high resolution mosaic is generated. In the first case, if the images are generated from the mosaic, they will have lower resolutions than those of the originals. If the first image has high resolution, the final mosaic will be huge to reserve the resolution of the first image after the images are aligned.

There have been various methods proposed on mosaic, sprite, or background generation. These terms are usually used in the same sense in the literature. Sprite (mosaic) generation methods mostly differ in the ways motion estimation and features used to create the mosaic. Different representations of mosaics like static, dynamic, and synopsis mosaic have been investigated in [47]. A direct method is used to align images and to generate the mosaic. When the planar scenes are captured from different angles or from the same point with rotation or zooming, the translation is called homography. 2D mosaic generation from a set of images using homographies is proposed in [110]. The extracted corners from images are used for mosaic generation. A sprite creation method based on connected operators is presented in [81]. The image is represented as flat zones and pruned using a max-tree representation. The outlier detection is obtained by comparing the original frame and current on the pixel level and this can eliminate some regions erroneously.

There are different types of mosaic depending on the camera motion. If the camera is translating sideways, a planar mosaic is generated [47, 97]. The cylindrical camera is used for the panning camera [110]. The spherical mosaic is generated when camera is both panning and tilting [23]. These methods do not consider forward motion or zooming of the camera. In [72], the forward motion of the camera is modeled and the mosaic is generated using the *pipe* projection. For example, the mosaics are effectively generated from shots taken from a plane or a car moving in the forward direction. In this kind of videos, the depth of the scene can be considered as infinite and projecting thin strips from images onto manifolds is effective in the mosaic generation. If the scene depth is finite, another method has to be applied since an image is already included in another image. Projecting thin strips is not effective in this case. If there is a zoom in a closed environment as in distance learning applications, the pipe projection cannot be applied properly. A detailed work on estimation of motion parameters and generation of sprites has been presented in [90]. A high resolution mosaic is generated by sliding the mosaic and warping the next frame into the mosaic [89]. Since warping occurs for every frame, the generated mosaic can still be blurred. Temporal integration methods are used according to the type of the mosaic that will be generated. The temporal integration methods also cause blurring in the mosaic.

Some methods use the depth information obtained using a couple of cameras [38, 30]. The background is estimated using clustering method to fit the data with an approximation of a mixture of Gaussians [38]. When the background is estimated, they use the color information along with the range (depth) information obtained from a pair of cameras. A disparity background model is created from video-rate stereo sequences in [30]. The problem is that there are many cases where only one camera is used to capture the scene. The depth information is obtained by the location of the projections on the epipolar line [94]. Assuming a rigid body motion, the area laying behind the objects that are close to the camera can be generated using the depth information.

Methods based on existence of error in capturing images or obtaining motion parameters have also been presented [82, 23]. A method for multi-image alignment and mosaicing having lens distortion is proposed in [82]. They also use a hierarchical coarse to fine iteration using Gaussian-Laplacian pyramid. Spherical mosaics generation is obtained by using quaternions and dense correlation [23]. They attempt to reduce the mismatches between pixels that cause blurring and ghosting due to inaccurate estimates of camera pose.

Background generation based on Kalman filtering and adaptive AR filter using pixels values has also been proposed to extract the background [77, 14]. Kalman filtering method assumes that the best information of a system state is obtained by an estimation that considers noise on the measurement of the system values [77]. The pixels are not updated if they belong to the background without intervening of foreground objects [14]. The example for image mosaicing does not include moving objects. An adaptive color background model for real-time segmentation of video streams is generated in Hue-Saturation-Value domain instead of RGB space. If the background cannot be detected because of the occluding objects, the undetected part is generated by morphing of the background [98]. Images are compared with a stored background model to decrease the bandwidth [67]. The images are divided into macroblocks. Each macroblock is compared with the macroblocks of the background to detect whether it belongs to a background or foreground object by thresholding.

The background mosaic is generated by segmenting the foreground objects from the background object and then finding the global motion [31]. The background is segmented by clustering of the dominant motion. The alignment of images is performed by using 3 unknowns in rotation matrix instead of 8-parameter matrix [97]. This requires the estimation of the focal length of the camera. Since the camera focal length camera cannot be always estimated properly, there may be some gaps in the panoramic images generated. To handle this, a close gap algorithm is proposed by recalculating the focal length. The 8-parameter alignment has also been used in [49].

2.3 Object Segmentation

Object segmentation is usually performed using regions, contours, and special features like corners, edges, vertices, etc. There are approaches that try to distinguish shadows from the objects. There are also methods that are proposed for the compressed domain rather than pixel domain.

One method for extraction of objects extracts region information [80, 33, 34, 26, 100, 106, 32, 60, 101, 35]. A generic partition tree for the regions that appear in the image is generated in [80]. This generic partition tree can be used for unsupervised and supervised spatial segmentation, region-based coding, semantic region segmentation, and motion spatial segmentation. The binary partitioning tree based on regions are also used in [33] and they use it for retrieval from databases. A merging algorithm is proposed containing merging order, merging criterion, and region model [34]. Regions in an image are represented with a region adjacency graph (RAG). The regions in I-frames are tracked comparing the size and texture, motion information and the distance of regions [26]. A video segmentation algorithm based on watersheds and temporal tracking is proposed in [100]. The regions are projected using motion parameters. The projected regions and regions in the current frame are compared. If the project region is overlapped by the current region a marker is created for that region to be used by watershed information. The watershed algorithm assigns labels for each region in the image. If the projected region does not coincide in another region, the algorithm for watersheds cannot be used properly. Regions are created using color boundaries and Bayesian clustering [106]. The probability density function of assigning pixels to regions has to be submitted to cluster using Bayes. A Markovian framework associated with a Gaussian modeling of the color distribution for each region is used for color spatial segmentation [32]. An image is divided into regions of pixels that correspond to those pixels that are displaced from the previous frame [60]. The performance of the algorithm depends on the apriori probabilities and the cost information. Kernel is defined as a group of neighboring macroblocks conforming to a similar motion [101]. The algorithm relies on MPEG motion vectors and good kernel extraction.

The algorithms that follow contours of objects assume that the objects do not move fast in consecutive frames. The objects may be tracked by checking the neighbors of borders. One method for object segmentation is the boundary extraction and/or tracking [16, 54, 21, 58, 39]. The boundaries may not fulfill the constraints of a region. Gap filling approaches are utilized to connect the boundary segments. Initial boundaries are obtained via pixel-based segmentation [16]. The tracking of the boundaries are performed at the boundary level. The nodes at the intersection of more than two regions; the boundaries, and regions are kept in a list to perform the segmentation. If the nodes are not linked, they are connected to the closest link with a segment. There are approaches that require initial user input before processing of video data [54, 21, 58]. A semi-automatic object tracking method for spatio-temporal segmentation is proposed based on the concept of snakes in [54]. After the image is segmented is regions, the user interacts to split erroneously merged regions or to merge regions to create the object [21]. The rough contours of the objects are acquired from the user in [58]. The contours are used to track the objects. Once the boundaries are obtained from the user, a boundary tracking algorithm is proposed to follow the boundaries in case of motion [39].

Feature-based methods first extract features from pixels or select pixels having distinct features from others [25, 64, 52, 69, 62, 15, 27, 102]. The feature models for edges, corners, and vertices are presented in [15, 27]. A corner is represented by the amplitude of the wedge, its aperture angle and a parameter to measure the smoothness of the wedge. Vertices are defined as a superposition of corner models. The extracted information is used in object segmentation. An image tracking algorithm using Jacobian based on selective pixels is proposed in [25]. If there is a planar surface, the pixels in the planar surface are not included as selected pixels. An object tracker matches the binary images of edges against subsequent frames [64]. The model is updated every frame to accommodate for rotation and changes in shape of the object. A model based on detection and tracking of edges discovers edges by using the edges in consecutive frames and in the background [52]. The method proposed in [69] consists of a motion detection phase employing higher order statistics and a regularization phase to achieve spatial continuity. VOPs are generated from an estimated change detection mask (CDM) in [62]. A buffer is used to increase temporal stability by labeling each pixel as changed if it belongs to an object at least once in the last L change detection masks. Algorithm proposed in [63] has been involved in *Core Experiments* of the MPEG-4 standardization process. The global motion is estimated and a scene cut detector is used to determine whether the initial frame of the shot is considered. Motion information is used to create a change detection mask indicating the existence of moving objects. The shape information is refined after obtaining regions belonging to the uncovered background. These types of algorithms may detect false objects in scenarios containing shadows or reflections. The model in [65] uses the edge pixels in an edge image detection. The model is updated every frame to accommodate for rotation and changes in the shape of an object. A segmentation algorithm based on Hausdorff distance by extracting edges is proposed in [65]. A model based on detection and tracking of edges [52] discovers edges by using the edges in consecutive frames and in the background. Video object segmentation methods [65, 52] can detect the edges relying on the moving edges of the objects. These methods either require the raw data or need to decompress the original data. The video objects can be segmented using the background model. The blocks from the frames and the background model are compared to extract objects.

There are also approaches that try to separate shadows from the objects [104, 78, 93]. Pfinder [104] separates shadows from human body in the scene using normalization of color components by the luminance. They use a person model to extract the person from the image. Once some pixels are associated with an object, a morphological growing operation is applied to create the person body. In [78], they assume that the variance among pixels within shadow area will be less than the variance among pixels in the original background region. A method based on detection of the penumbra and umbra of shadows is proposed in [93]. This algorithm assumes a single light source, a stationary camera, and a plane background.

There are methods that perform their operations on the block level rather than pixel level thus decreasing sensitivity to noise in the environment [109, 84]. There are also techniques that exploit the data structure in the compressed domain and simplify the image contents working in DC images obtained from Discrete Cosine Transform [105]. DC image and AC energy of DCT coefficients are used for initial segmentation [95]. The entropy of AC energy is used to distinguish the regions belonging to objects and the background. The entropy for object is assumed to be high. The regions are merged spatially using Gaussian random variable.

Chapter 3

Compressed Domain Processing

The video data may also exist in compressed form. Video sequences usually contain enormous data for video processing. Decompressing data and then processing uncompressed data is computational intensive. Moreover, this may yield erroneous results due to processing unnecessary data. In this chapter, methods for stationary background generation and video object segmentation in compressed domain are expressed briefly. The stationary background generation is performed by using basic features from Discrete Cosine Transform (DCT) blocks. The background is generated using clustering based on temporal information [6]. Advanced coarse boundary features are extracted to eliminate insignificant blocks for video object segmentation. The features are extracted about the smoothness, the boundary visibility, and the boundary structure of a block [7]. Since MPEG-1 is a standardized video compression method and it uses DCT as other standard methods, we performed some of our operations on MPEG-1.

The new video coding standard MPEG-4 [87] enables the content-based functionalities by introducing the concept of video object planes (VOP). The coding of video sequences that are segmented based on video contents, flexible reconstruction, and manipulation of video contents at the decoder have been the primary objective. Thus, video object segmentation, which emphasizes on partitioning the video frames into semantically video objects and the background, becomes a significant issue for the effective manipulation of MPEG-4 and MPEG-7. However, there has been significant amount of video that has been compressed using MPEG-1 and MPEG-2, which use DCT to compress data. Decompressing data and then processing decompressed data is computational intensive. It is important to extract some coarse features to reduce decompression and to use these features in processing.

This chapter is organized as follows. The following section gives basic information about MPEG video streams. Section 3.2 describes the extraction of features from DCT compressed blocks. Stationary background generation using clustering based on DC coefficients of blocks are explained in Section 3.3. The extraction of object boundaries and object segmentation are explained in Section 3.4. The last section summarizes the chapter.

3.1 MPEG Video Stream

An MPEG-1 video stream is composed of I, P, and B frames, where P and B frames exploit the similarity between the frames. P and B frames assume very little change with respect to their dependent frames and their macroblocks are decoded using macroblocks in I and P frames.

In MPEG-1, each macroblock of I-Frame is composed of 6 blocks: 4 luminance blocks and 2 color components (Figure 3.1.1). Blocks are compressed using DCT and each block is represented with DC and AC coefficients. Our algorithm works at the level of macroblocks since the compression is performed at the level of macroblocks. Let $MB^{pq}(\alpha)$ be the macroblock at the p^{th} row and q^{th} column of frame α (Figure 3.1.1. Each DC coefficient of $MB^{pq}(\alpha)$ can be denoted as $DC^{pq}(\alpha)$. $MB^{pq}(\alpha)$ may be represented with its DC coefficients as follows:
$\{Y_0^{pq}(\alpha), Y_1^{pq}(\alpha), Y_2^{pq}(\alpha), Y_3^{pq}(\alpha), Cb^{pq}(\alpha), Cr^{pq}(\alpha)\}$ where Y_0, Y_1, Y_2, Y_3, Cb, and Cr are DC coefficients of the blocks that are shown in Figure 3.1.1.

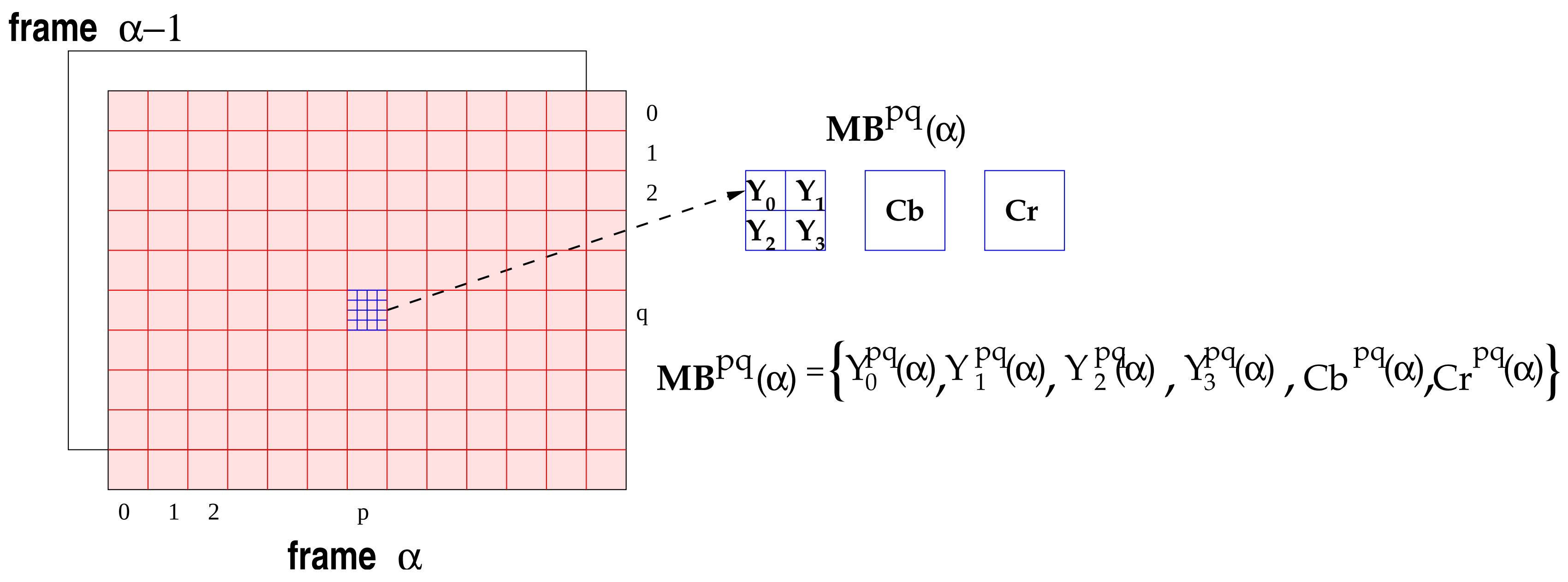

frame α−1
frame α
0
1
2
p
q
MB^{pq}(α)
Y_0 Y_1 Y_2 Y_3
Cb
Cr
MB^{pq}(α) = {Y_0^{pq}(α), Y_1^{pq}(α), Y_2^{pq}(α), Y_3^{pq}(α), Cb^{pq}(α), Cr^{pq}(α)}

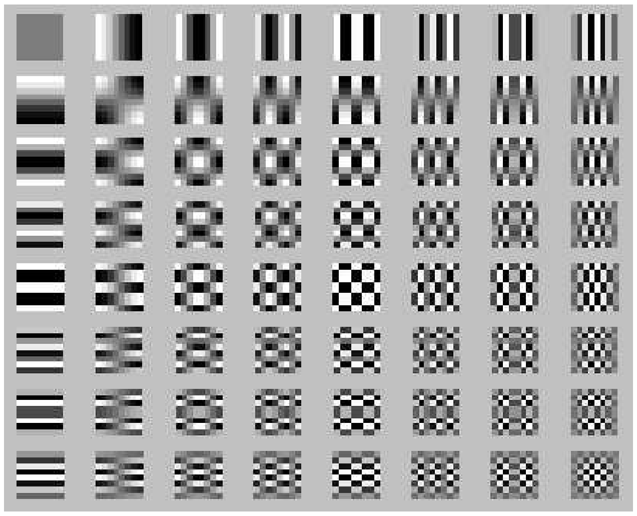
Figure 3.1.1: Macroblocks of MPEG frames.

3.2 Extraction of Coarse Boundary Features from a DCT Block

There has been significant video data that are compressed using DCT. The traditional methods decompress all the blocks and then apply video processing techniques. This increases computational complexity due to decompressing unnecessary blocks and processing these insignificant blocks. The significant blocks can be determined by analyzing DCT blocks. In this section, we propose several features that can be extracted to compare two blocks. A DCT compressed block is obtained from 8x8 rectangular region of pixels. The major features that are extracted are smoothness, the complexity of patterns, and the boundaries (or edges) inside a block.

3.2.1 Smoothness and Patterns

Each block is composed of DC and AC coefficients. DC coefficient of a block carries the most information about the block that is the 8 times of the average of the pixel values inside the block. However, DC coefficient provides no information on the structure of a block or how pixels are spread. If only DC coefficients are used for comparing blocks, different blocks may be assumed to be similar because of the equality in the average of the values in a block.



0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure 3.2.1: 2D DCT basis images and their zigzag numbering.

AC coefficients must be used effectively to determine the contents of a block. Each AC coefficient is the coefficient of a basis image shown in Figure 3.2.1. The number of non-zero AC coefficients (NZ_{AC}) shows how complicated the block pattern is [3]. If NZ_{AC} is 0, the block pattern is smooth. This also indicates the absence of an edge in the block. As NZ_{AC} increases, smoothness in the block decreases and some patterns become visible.

The magnitude of AC coefficients is an indicator of edges or boundaries that may exist in the block. AC coefficients are sorted in descending order according to the absolute value of the AC coefficients. We sum up the absolute value of M highest AC coefficients, and the sum yields the block boundary visibility (BV). High values of BV indicate clear boundaries whereas low values indicate weak boundaries. We do not use all the coefficients since low coefficients determine the shape of the boundary rather than the visibility. This method does not rely on first coefficients with respect to zigzag ordering as in [85, 70].

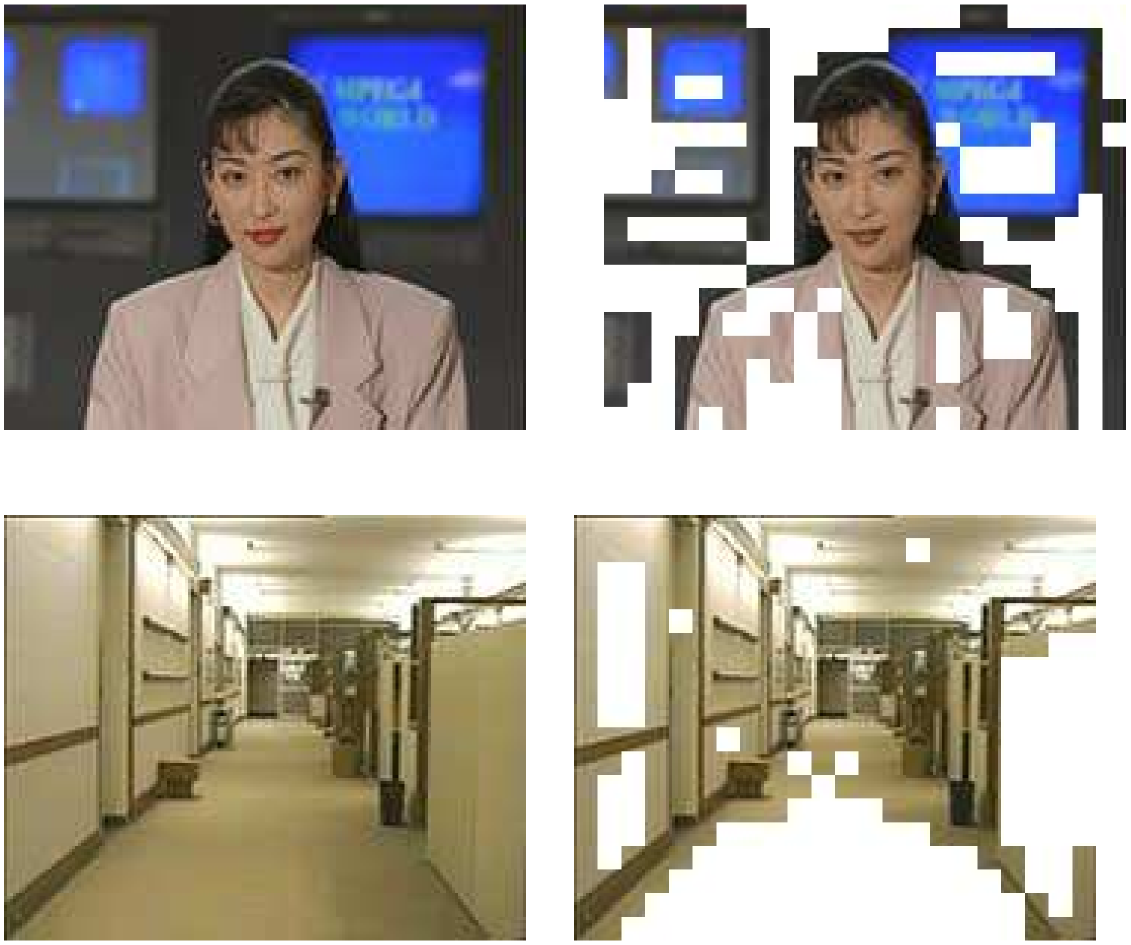


Figure 3.2.2: Blocks that contain edges.

Figure 3.2.2 shows the blocks that contain edges from the first frames of ’Akiyo’ and ’Hall Monitor’ test sequences.

3.2.2 Boundaries

The zigzag index (Figure 3.2.1) of the highest absolute magnitude of AC coefficient indicates the boundary type in the block. $Ordered_{AC}$ maintains the indices of AC coefficients in descending order according to their absolute values and $Ordered_{AC}(i)$ points the i^{th} $(0 \leq i < NZ_{AC})$ highest index. There is a *vertical* boundary if $Ordered_{AC}(0) \in \{1, 5, 6, 14, 15, 17, 28\}$ where $Ordered_{AC}(0)$ is the index of the highest AC coefficient. Basis images of these AC coefficients only have vertical borders. There is a *horizontal* boundary if $Ordered_{AC}(0) \in \{2, 3, 9, 10, 20, 21, 35\}$. If $Ordered_{AC}(0) \in \{4\}$, there is a *diagonal* boundary in the block.

The order of indices with respect to the absolute magnitude of AC coefficients indicates where the boundary is in the block. The sign of the AC coefficients indicates the dark and light regions in the block. Thus, we can extract coarse information by ordering of indices and using the signs of AC coefficients.

We extracted blocks that contain boundaries from the first image of 'Akiyo' test sequence. For example, $Ordered_{AC} =< -1, +6 >$ denotes that the index of the highest AC coefficient is 1 and it is negative, and the second highest index is 6 and it is positive. We deduce that the left part of the block is darker than its right side because the highest index 1 has a negative coefficient. The second index helps us locate the boundary. The second AC index is still a vertical AC index; therefore the block has a vertical boundary. Since the second AC index is $+6$, if basis images of -1 and $+6$ are overlapped, the boundary appears to be $2^{nd}-3^{rd}$ column from the left. Some examples from the blocks of 'Akiyo' test sequence are shown in Figure 3.2.3. First two coefficients usually determine the position of the boundary and the third one indicates the slope in the boundary. Since AC coefficients having larger magnitudes are effective in determining the boundaries, this method is more reliable than the other methods that rely on the zigzag ordering of coefficients.

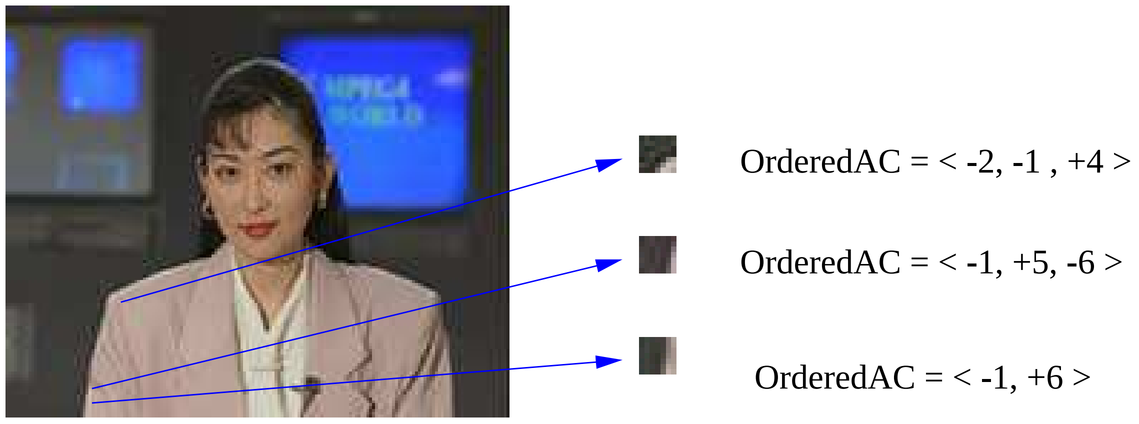


Figure 3.2.3: Boundaries in a block.

The feature vector of a block is 5-tuple $B=\{DC,NZ,BV,BoundaryType,Darkness\}$. The comparison of two blocks is performed by comparing their feature vectors. The type of the boundary is important to compare the similarity of blocks. Sometimes, a block is compared with neighboring blocks to check the continuity of the boundary.

3.3 Stationary Background Generation

The detection of the background can be accomplished if the moving object displaces its location. In case of displacement, the hidden background behind the object can be visible. In consecutive frames, there is usually very slight change. Therefore, the process of the background generation at intervals of frames provides faster speed. P and B frames assume very little change with respect to their dependent frames and their macroblocks are decoded using macroblocks in I frames. So, I frames are better candidates for the background construction since there is enough object displacement in video sequence and they do not depend on any other frame. But, if the mobile objects are small (i.e. that can fit in a macroblock) and their displacement is also small, then further processing is needed. In that case, P and B frames need to be processed. In this work, we address large mobile objects.

The background generation has three steps:

- The generation of the background from a video shot
- Merging of the same backgrounds that are generated from different video shots that belong to the same video scene
- Merging of the same backgrounds that appear in different video scenes

Our algorithm has three phases: the clustering of the macroblocks, the selection of the cluster that may contain the background macroblock and the selection of the background macroblock from the cluster. The algorithm is thus as follows:

Algorithm 3.3.1 Background generateStationaryBackground(Video *vid*)

```
// $ClusterList(p,q)$ keeps all clusters of macroblocks that appeared at $p^{th}$ row and $q^{th}$
// column,
// $cluster(p,q)$ is the selected cluster (that has probably have the background
// macroblock) from the $ClusterList(p,q)$,
// $vid$ is the video sequence of stationary frames
for each I-frame $\alpha$ of the video sequence $vid$ do
    for each macroblock $MB^{pq}(\alpha)$ do
        Cluster $MB^{pq}(\alpha)$ into $ClusterList(p,q)$
    end for
end for
for each macroblock $MB^{pq}$ do
    Select the $cluster(p,q)$ from $ClusterList(p,q)$
    Select the background macroblock from $cluster(p,q)$
end for
Combine all macroblocks selected
```

3.3.1 Clustering

Two methodologies may be used for clustering: non-incremental clustering and incremental clustering. In a non-incremental clustering method, all the macroblocks must be stored until a shot change occurs. The macroblocks can be clustered using a non-incremental clustering method for a video shot. This method is good if the video shot length is short (less than 5 seconds). But if the shot length is long and if it is possible to generate the background macroblock earlier, there is no need to first process and then cluster all the macroblocks in a video shot. In this case, it is better to use incremental clustering.

Let the background macroblock at location (p,q) need to be generated. All macroblocks that showed up at this location are clustered. The feature vector for a macroblock is the DC coefficients of the blocks. In our case, we map macroblocks to a one-dimensional space and order them according to the distance from a specific point and then cluster them incrementally as macroblocks arrive. The macroblocks are clustered using Nearest Neighbor Rule (1-NNR). If the distance from the existing clusters is more than a specific threshold (τ), a new cluster is created for the macroblock. Most clustering algorithms are satisfactory to cluster the macroblocks that have 6 elements in their feature vector. Since the DC coefficients are already approximation to the macroblock, the distance function and how features are evaluated gain significance in clustering.

Distance Function. In our work, two types of distance measures are considered: additive and selective. Additive distance measures accumulate the difference at each feature (e.g. Manhattan, Euclidean). Selective distance measures depend on the selection of one of the difference of features (e.g. maximum, minimum). As an example for additive distance measure, Euclidean distance measure is used. Both *maximum* and *minimum* distance measures are considered for the selective distance measure.

The features may be evaluated in several ways. Four methods are stated here. First method assigns equal weights to each DC coefficient. Let $MB^{pq}(\alpha)$ and $MB^{pq}(\beta)$ be the macroblocks that are compared. The absolute difference between two DC coefficients of

$MB^{pq}(\alpha)$ and $MB^{pq}(\beta)$ can be represented as $\triangle DC^{pq}(\alpha,\beta)$. Then the Euclidean distance between $MB^{pq}(\alpha)$ and $MB^{pq}(\beta)$ is computed as

$$(3.3.1) \qquad \sqrt{\sum_{i=0}^{3} \triangle Y_i^{pq}(\alpha,\beta)^2 + \triangle Cb^{pq}(\alpha,\beta)^2 + \triangle Cr^{pq}(\alpha,\beta)^2}$$

The second method assigns the same weight to the chrominance and the luminance and takes the average of the luminance coefficients. The average of the luminance DC coefficients of $MB^{pq}(\alpha)$ is denoted with $Y_{avg}^{pq}(\alpha)$ which is $\frac{1}{4}\sum_{i=0}^{3} Y_i^{pq}(\alpha)$. This distance is computed as

$$(3.3.2) \qquad \sqrt{(\triangle Y_{avg}^{pq}(\alpha,\beta)^2 + Cb^{pq}(\alpha,\beta)^2 + \triangle Cr^{pq}(\alpha,\beta)^2}$$

The DC coefficient is 8 times the average of the values in the block. So, a DC coefficient is the smoothing of the values in the block. It may be better to keep the differences as much as possible. Instead of averaging luminance values, the maximum luminance difference, $\triangle Y_{max}^{pq}(\alpha,\beta)$, which is $max_{i=0,3} Y_i^{pq}(\alpha,\beta)$, may be evaluated. Then the distance is

$$(3.3.3) \qquad \sqrt{\triangle Y_{max}^{pq}(\alpha,\beta)^2 + \triangle Cb^{pq}(\alpha,\beta)^2 + \triangle Cr^{pq}(\alpha,\beta)^2}$$

Sometimes, it may only be necessary to consider sharp changes in the sequence. Instead of computing the maximum of luminance difference, the minimum luminance difference can be taken, $\triangle Y_{min}^{pq}(\alpha,\beta)$, which is $min_{i=0,3} \triangle Y_i^{pq}(\alpha,\beta)$. Then the distance is

$$(3.3.4) \qquad \sqrt{\triangle Y_{min}^{pq}(\alpha,\beta)^2 + \triangle Cb^{pq}(\alpha,\beta)^2 + \triangle Cr^{pq}(\alpha,\beta)^2}.$$

Maximum selective measure is used if the difference between two macroblocks needs to be emphasized as much as possible. The function for the *maximum* distance is:

$$max(\triangle Y_{max}^{pq}(\alpha,\beta),\triangle Cb^{pq}(\alpha,\beta),\triangle Cr^{pq}(\alpha,\beta)). \tag{3.3.5}$$

The sharp changes at a macroblock can be detected using the *minimum* distance measure as follows:

$$min(\triangle Y_{min}^{pq}(\alpha,\beta),\triangle Cb^{pq}(\alpha,\beta),\triangle Cr^{pq}(\alpha,\beta)). \tag{3.3.6}$$

3.3.2 The Cluster Selection

If the number of clusters is more than one for a macroblock location, there is a moving object at that macroblock. If the number of clusters is one, then there is no movement at that macroblock and the cluster is the only candidate that contains the background macroblock.

There are two basic factors that are used for the selection of the cluster: *frequency* and *continuity*. The frequency of a cluster is the number of elements in that cluster. The continuity of a cluster denotes the maximum length of the sequence of macroblocks of the cluster that appeared sequentially. The clusters are first chosen according to their frequency. If there is a tie, the cluster that has a higher continuity is selected. Sometimes, there may be no or very little motion in succeeding I-Frames. All these frames have the effect of a single frame on the frequency and the continuity of the clusters.

3.3.3 The Background Macroblock Selection

The background is chosen from the cluster in four ways. Three of them are about luminance and the other one considers both luminance and chrominance. In the first case, if the moving object absorbs light (non-shining), it causes darkness on the background. It is better to choose the candidate that has a higher luminance. Low luminance is due to the object in the environment. In the second case, if the moving object is a light source or reflects light, it causes the background to shine. So, it is better to choose the candidate that has a lower luminance. High luminance is caused by the object in the environment. In the third case, if the type of the object is unknown or it may have the properties of both shining and non-shining object, it may be better to be neutral. So, the background block that is closer to the mean is chosen. Finally, if the color is considered, the background macroblock that is closer to the mean of all the block coefficients is the candidate.

3.3.4 Enhancement with Motion Vectors

The motion vector indicates whether the macroblock moves to other parts of the frame. Although the term ”motion vector” is used in MPEG streams, motion vectors do not exactly express the displacement of a macroblock. They rather give the location of the closest

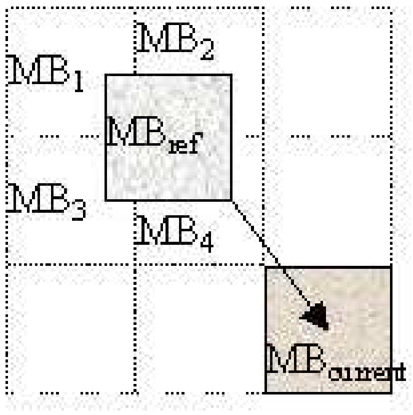


Figure 3.3.1: Estimation of DC coefficient of the reference macroblock.

macroblock in the previous or next frame. This is crucial if there is a pattern in the background. If the macroblock is previously located in another location, this usually shows the parts where moving objects exist. If there is a pattern in the frame, it is also possible that this macroblock points to another location sharing the same pattern. We apply two simple methods to detect this situation.

Since we do our operations on the macroblock level, we do not have the exact DCT coefficients of this block. One way to deal with this is to check all macroblocks that intersect with this macroblock. If all of them have the same characteristics as in the predicted frame, the macroblock has not moved. Another way is to estimate the DC coefficients of the referenced block. The estimation of the DC coefficients [105] is done by giving weights according to the macroblock coefficients by the area that share with the reference macroblock MB_{ref} (Figure 3.3.1):

$$DC_{ref} = \sum_{i=1}^{4} (w_i \times DC_i) \tag{3.3.7}$$

where DC_i gives the DC value of a block in Figure 3.3.1 and w_i is the ratio of region covered by MB_i to the region of the whole macroblock (8x8=64 pixels).

Let $MB^{pq}(\alpha)$ and $MB^{pq}(\beta)$ represent the referenced macroblock in the referenced frame and in the current frame, respectively. If

$Distance(MB^{pq}(\alpha), MB^{pq}(\beta)) < \tau$,

where $Distance$ is a distance function and τ is a threshold, this implies that the reference block has not changed in the current frame. Although the motion vector informs that the macroblock moved, it did not move. So, it may be assumed that reference block does not contain a moving object.

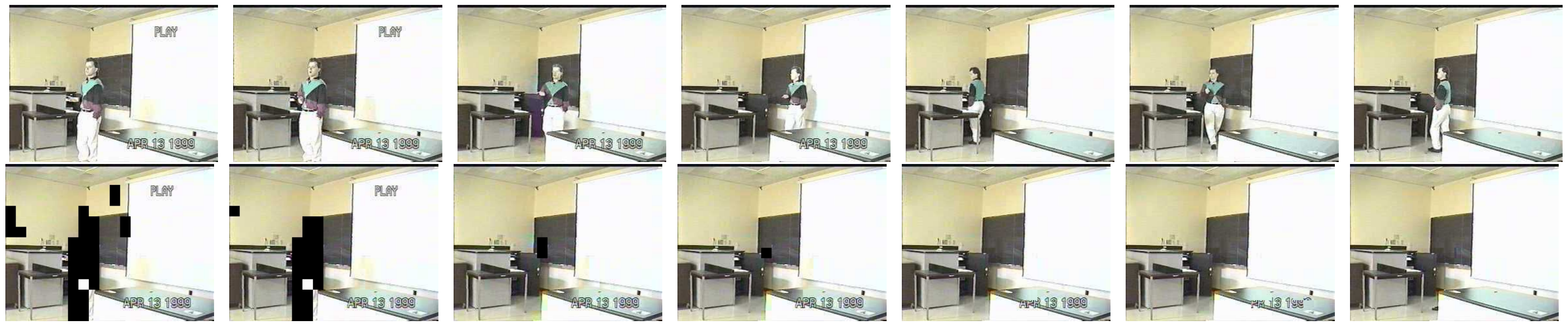
3.3.5 Sample Results

We have used video streams that are recorded in the lectures and MPEG-4 test sequence “Hall Monitor”. Each stream is stored as a MPEG-1 video stream. The coding pattern of streams is IBBPBBPBBPBBPBB and the frame rate is 15 frames per second. Figure 3.4.1 shows the phases of how the backgrounds are generated. The first rows display the frames that are encountered. The second rows show the phases of the background generation. Figure 3.4.1 (a) shows a lecture example where the parts of the instructor’s body may be occluded by the objects in the environment. Figure 3.4.1(b) shows an example where more than one object may appear and also objects like the bag may be occluded by other objects. For this example, first frames of the original video sequence are omitted.

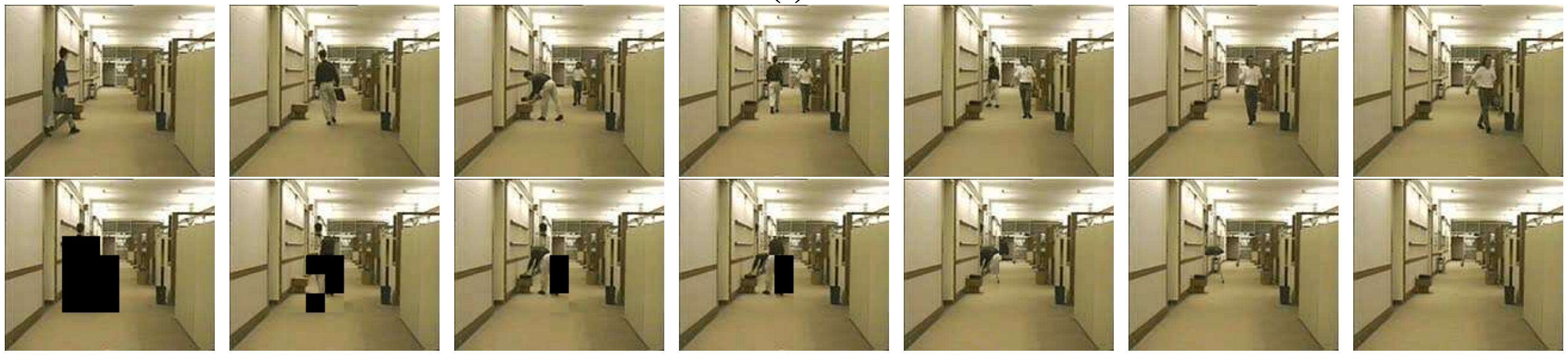
Our observations showed that if the objects move enough, the background can be constructed in the early frames of the clip. In the lecture example, the background is generated after processing 13 I frames. 120 B and 48 P frames are skipped. In the hall example, the background is generated after processing 18 I frames. 170 B and 68 P frames are skipped. Our results showed that chrominance must be included in the distance computation and the selection of the background macroblock from the cluster. The false macroblocks that have color distortions may be selected if only luminance coefficients are considered.

3.4 Video Object Segmentation

The features extracted from the compressed data can be used to identify significant regions before the segmentation. Since the compression technique in MPEG-1, MPEG-2, and MPEG-4 is DCT, we have proposed a reliable method to extract significant blocks by extracting features about the smoothness and boundaries from DCT compressed blocks.



(a)



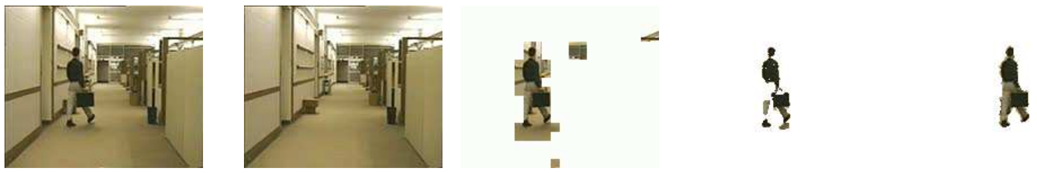
(b)

Figure 3.4.1: Video Sequences and Background Generation.

Features about smoothness and structure of boundaries are evaluated to determine the significant blocks from compressed video for object segmentation.

The general method for segmentation of video objects is to detect the boundaries of the object. Therefore, if a region does not have a visible boundary, it is very unlikely that the region contains the boundary for the object. The background model can be generated automatically or presented to the system. The feature vectors of blocks of the frames and the background model are compared to eliminate insignificant blocks. If the feature vectors are different, the frame block is likely to contain data about an object and considered as a significant block. The significant blocks are decompressed and used for further processing.

Firstly, the blocks in the background model and the frame are compared. Three thresholds, τ_{DC}, τ_{NZ}, and τ_{BV}, are used to compare DC coefficients, the number of non-zero AC coefficients, and boundary visibility of feature vectors. Let f_1 and f_2 be feature vectors of two blocks to be compared. If $|f_1(DC) - f_2(DC)| > \tau_{DC}$ or $|f_1(NZ) - f_2(NZ)| > \tau_{NZ}$, the blocks are considered as different blocks. If $f_1(BV) < \tau_{BV}$ and $f_2(BV) < \tau_{BV}$, blocks are assumed to have no visible boundaries. If only one of the blocks has a visible boundary (i.e., $\geq \tau_{BV}$), the blocks are considered different. If two blocks have visible boundaries, the boundary type and the darkness of boundaries are compared. If both are the same, they are treated as similar. The different blocks are considered as significant blocks. These blocks are decompressed and used in further processing. The frame 43 and background of ’Hall Monitor’ test sequence are shown in Figure 3.4.2 (a) and (b), respectively. The significant blocks that are selected by comparing coarse boundary features are displayed in Figure 3.4.2 (c). Two macroblocks are misdetected as significant blocks due to their complex patterns.



(a) (b) (c) (d) (e)

Figure 3.4.2: Experiments. (a) frame 43 (b) background image (c) decompressed (significant) blocks for edge detection (d) thresholded frame (e) video object.

Secondly, edges in the significant blocks are extracted to detect the boundaries of a video object. The Canny [20] edge detector is used to extract the edges. The most distinguished feature of an edge is its gradient, ∇g. Edge matrix E maintains the edges in an image, which is denoted by

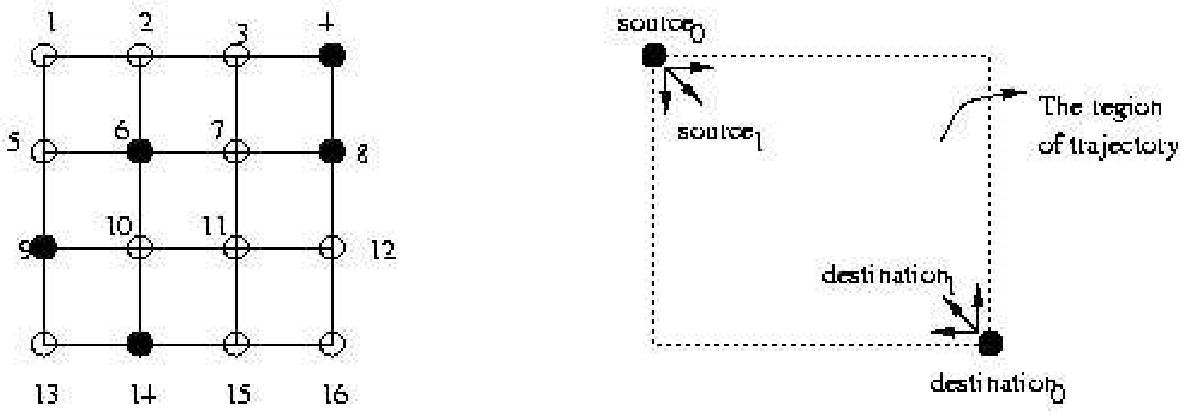
$$E = \begin{cases} E_{xy} = 1 & \text{if there is an edge at (x,y)}(\nabla g > \tau_{Edge}) \\ E_{xy} = 0 & \text{otherwise} \end{cases} \tag{3.4.1}$$

Let E^{β} and $E^{\mathcal{F}_i}$ represent the edge matrix for the background and frame i. The difference edge matrix (DE^i) holds the different edges that exist in the frame but not in the background where $DE^i = ((E^{\mathcal{F}_i} \text{ XOR } E^{\beta}) - E^{\beta})$. DE^i may also contain noisy edges due to the illumination change in the environment. $\Phi(DE^i)$ denotes the edges after noise removed. The removal of the noise is performed based on the image produced with thresholding (Figure 3.4.2 (d)). It is possible that significant blocks may be missed in the initial comparison phase. A significant block may be missed because of a weak boundary close to the borders of a block. This kind of a block has similar DC, NZ_{AC}, and BV values with the background model. If there is an edge on the border of an insignificant block, the insignificant block is also decompressed before further processing.

Thirdly, as a result of the previous steps the edges may be disconnected. The edges of the video object may be removed since they overlap with the background edges or may be removed as noise mistakenly. If the gradients of nodes are below the threshold for edge, the edges may again be disconnected. Linking of edges creates new edges and is composed of two phases: linking of close edges and linking of distant edges. Let $\Phi(DE^i_{xy}) = 1$ and $\Phi(DE^i_{pq}) = 0$ where (x,y) and (p,q) are coordinates, $x-1 \leq p \leq x+1$, and $y-1 \leq q \leq y+1$. Let N_{pq} denote the set of 8-connected nodes of a node at (p,q). For simplicity, assume that $E = \Phi(DE^i)$. E_{pq} is converted to an edge if

$$\begin{aligned} E_{xy} = 1 \wedge \exists p \exists q (E_{pq} = 0 \wedge \exists s \exists t (E_{st} = 1 \wedge (E_{st} \in N_{pq}) \\ \wedge \neg \exists u \exists v (E_{uv} = 1 \wedge (E_{uv} \in N_{xy}) \wedge (E_{st} \in N_{uv})) \end{aligned} \tag{3.4.2}$$

is true. Informally, a node is converted to an edge if the node connects edges that are not connected directly or through their neighbors. In this context, a node is an edge if its gradient is higher than the edge threshold. For example, in Figure 3.4.3 (a), nodes 4, 6, 8, 9 and 14 denote the edges. We want to link node 6 to another node. The 8-connected neighbors of node 6 are nodes 1, 2, 3, 5, 7, 9, 10, and 11. Node 9 is already an edge. Nodes 1 and 2 do not connect node 6 to any other edge. Therefore these nodes are ignored. If node 3, 7, or 11 becomes an edge, then nodes 4 and 8 are reachable from node 6. Nodes 10 and 11 connect node 6 to node 14 but node 14 is already reachable from node 6 through node 9. So, the only candidates are nodes 3, 7, and 11. Once one of nodes 3, 7, or 11 is chosen, all the edges in the figure are reachable from node 6.



(a) (b)

Figure 3.4.3: Edge Linking. (a) close edge linking, (b) distant edge linking.

Linking distant edges is different since none of the neighbors connects the edges to the distant edges. Assume that we want to link $source_{xy}$ to $destination_{pq}$. The main idea is to follow the nodes having high ∇g. There are 2 significant problems: 1) if the edge is also connected to a strong edge in the background, the trajectory from the source may stray, and 2) if we start from the source and try to reach the destination by following the gradients of nodes, the edges nearby the destination may be misdetected and the real edges may be eliminated. We use a heuristic to solve the first problem. If the source is at (x,y), the destination is at (p,q), and (s,t) is an edge on the trajectory from the source to the destination, then $min(p,x) \leq s \leq max(p,x)$ and $min(q,y) \leq t \leq max(q,y)$. To solve the second problem, instead of starting from the source and reaching the destination, a single step from the source to destination and a single step from the destination to the source is taken at each iteration. A single step is to visit an neighbor of an edge (Figure 3.4.3 (b)). We use the Manhattan distance to measure the distance between edges. The condition $distance(source_{i+1}, destination_{i+1}) < distance(source_i, destination_i)$ must be satisfied after each iteration. The iteration continues until the source and the destination are 8-connected. The snake model [51] is proposed to extract open and closed contours of objects. It needs initial points for the contours and requires weights for adjusting the curves of the contours. The distant edge linking can also be performed using the snake model. Finally, the regions within the boundary of the object are filled to obtain the video object (Figure 3.4.2 (e)).

In our experiments, we observed that sorted AC coefficients are significant in detection and extraction of coarse boundaries. Since first two AC coefficients are usually the highest AC coefficients, the methods that depend on these coefficients may also yield good results in some cases. However, during comparisons of video frames, there are also blocks that have other significant coefficients that cannot be distinguished using the first 2 AC coefficients. This increased the number of blocks that are assumed to be similar in the preprocessing. NZ_{AC} is also not enough to describe the coarse boundary features when NZ_{AC} is quite small (i.e. less than 6). If NZ_{AC} is high, it is likely that there are many AC coefficients having small magnitude. NZ_{AC} is useful in comparisons if the difference between the number of non-zero AC coefficients is significant. By using combination of the prestated features, we obtained satisfactory results.

3.5 Summary

In this chapter, we have firstly presented a method to extract coarse boundary features from DCT compressed blocks. We have described how stationary background can be generated from MPEG-compressed video by using DC coefficients of blocks. This method relies on the displacement of the objects in the video. We have showed that DC coefficient, smoothness, boundary visibility, boundary type, and darkness are good features to determine significant blocks. The boundary features are used to eliminate the insignificant blocks for video object segmentation.

Chapter 4

Global Motion Estimation with Motion Sensors

Global Motion Estimation (GME) techniques have been developed and usually applied on video that motion takes place often. Although these methods produce accurate results where frequent motion occurs, they turn out to be inefficient if motion is not so often in the video as in semi-dynamic videos. In this chapter, we propose motion sensors that will indicate the existence of motion and yield quick approximation to the motion. Motion sensors reduce the computations of the hierarchical evaluation of low-pass filtered images in iterative descent methods [8]. We also propose how GME can be performed at intervals accurately.

4.1 Introduction

Global Motion Estimation (GME) techniques play an important role in video compression methods. GME is usually used to describe the camera motion in a video. The motion is usually modeled with perspective, affine, translation-zoom-rotation, or translational motion models. Most of the GME techniques developed concentrate on the accuracy of motion parameters of the chosen motion models [28, 90, 89]. These methods usually include an initial estimation of the subset of the motion parameters and then adjusting of the motion parameters using a hierarchical pyramid of low-pass filtered images. These methods are usually tested on dynamic video where there is almost always motion in the video. In semi-dynamic video applications, like distance education, the motion does not happen often. The motion usually happens at intervals and then the camera stabilizes.

A hierarchical gradient descent method that uses M-estimators has been used to perform GME [28, 90, 89, 57]. An initial matching is necessary to avoid being trapped in local minima. The iterative descent is used to adjust the motion parameters at each level of the pyramid. There are two drawbacks of this kind of approaches: error in initial estimation and hierarchical computation of iterative descent. First drawback causes the technique to be trapped in local minima and the next one introduces significant computation. Tomasi and Shi [86] presents feature selection process based on a dissimilarity of feature selection. The features are selected based on the initial frame and the current frame thus depending on the motion between two frames. There are features presented based on edges (high gradients), corners, blocks having high spatial frequency. Features are selected using the Laplace operator with its FIR filter coefficients $1, -2, 1$ [76]. This type of features is selected according to the neighboring pixels.

In our system, GME is used for the sprite generation and spatial browsing. The sprite should include every part of the scene that is visible throughout the video sequence. If there is no a priori motion information for a video sequence, the motion has to be estimated between each sequential frame. If there is a priori information the motion estimation can be performed at a number of frames. However, if the global motion becomes so significant, the motion model may not be able to estimate the motion. The global motion estimation at intervals is important to produce a clearer sprite. Therefore, the GME has to be tuned for increasing accuracy.

In this chapter, we propose motion sensors, which are sensitive to motion that may take place. The motion sensors are expected to displace their positions in any type of motion and should be enough to describe the motion. For example, 4 motion sensors should yield information about perspective motion and 3 motion sensors should yield information about affine motion. For each pixel within the initial frame, a block search is performed. We used two kinds of masks: square and circular. The motion sensors are not only edges having high gradients. They are obtained by using more general information than gradients and carry more information in case of motion. The global motion estimation is also performed at intervals to reduce blurring in sprite generation.

This chapter is organized as follows. Section 4.2 following section explains motion sensors. The GME with motion sensors is discussed in Section 4.3. Section 4.4 explains how global motion estimation is tuned for spatial browsing. The last section summarizes our chapter.

4.2 Motion Sensors

Our experiments showed that motion estimation methods that process all the pixels that are available are slow for video. So, rather a set of feature points is selected and they are tracked in each frame. Mapping feature points in two frames is not easy since there may be several feature points that share the same characteristics.

This method aims to match blocks that have motion sensors as their center points rather than mapping feature points. This approach does not require the exact mapping of the feature point. The error function used in block matching introduces flexibility in tracking of the motion sensors points even though feature points are not located as they are expected.

Motion sensors are feature points that are sensitive to motion. The motion sensors are expected to displace their positions in any type of motion. 2 motion sensors should yield information about translation-zoom-rotation motion and 1 motion sensor should be enough to detect translational motion.

Edges having high gradients are likely to be candidates for motion sensors. Unfortunately, gradient contains information within ± 1 pixel distance that is usually not enough to detect motion due to existence of patterns or aperture problem. Another problem with the edges is the mapping of edges in two frames. Another edge may possess similar information to the desired edge and makes it difficult to detect the motion.

It is very hard to find absolute motion sensors that will change their locations after every type of motion. The goal is to find the motion sensors that will displace their locations in most types of motion. In Fig. 4.2.1, there are two lines intersecting each other. Every point lying on lines l_1 and l_2 is likely to displace to their positions after a motion. Let the slopes for l_1 and l_2 be m_1 and m_2, respectively. If there is a motion in the direction of m_1, the points lying on line l_1 will be useless and moreover, make the motion estimation difficult. Similar statement is also true for line l_2. Which points carry the highest information for motion? The answer is the intersection of line l_1 and line l_2 because the p will always change its location either there is a motion in the direction of m_1 or m_2.

In real video, the existence of lines and their intersection are not guaranteed. Even though, the lines may exist, they may not be straight lines or may be hard to detect. Therefore, we propose a method that will work without detection of lines and using gradients (since they

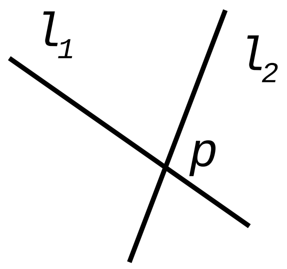


Figure 4.2.1: Intersecting lines.

carry limited information about the surrounding pixels).

4.2.1 Detection of Motion Sensors

A feature point is distinguishable by its surrounding pixels. A block matching process is performed for a block containing motion sensor *ms* as its center within the same frame. The block search operation is performed within distance of d. Each block size has a height and width of n pixels. For the block search operation, n-step search can be used. As an error function, we use the sum of absolute error differences (SAD)

$$\varepsilon(B^p, B^t)_{p \neq t} = \Sigma_{i=1}^{n} \Sigma_{j=1}^{j=n} |B^{ms}(i,j) - B^t(i,j)| \tag{4.2.1}$$

where B^p represents the block where p is its center and B^t represents a block within the search distance d. Let x_p and y_p denote the x and y coordinates of a pixel p. The sensitivity of a pixel is determined by $s(p)$ which is computed by

$$min_{(x_p-d) \leq x_t \leq (x_p+d),(y_p-d) \leq y_t \leq (y_p+d)}(\varepsilon(B^p, B^t)); \tag{4.2.2}$$

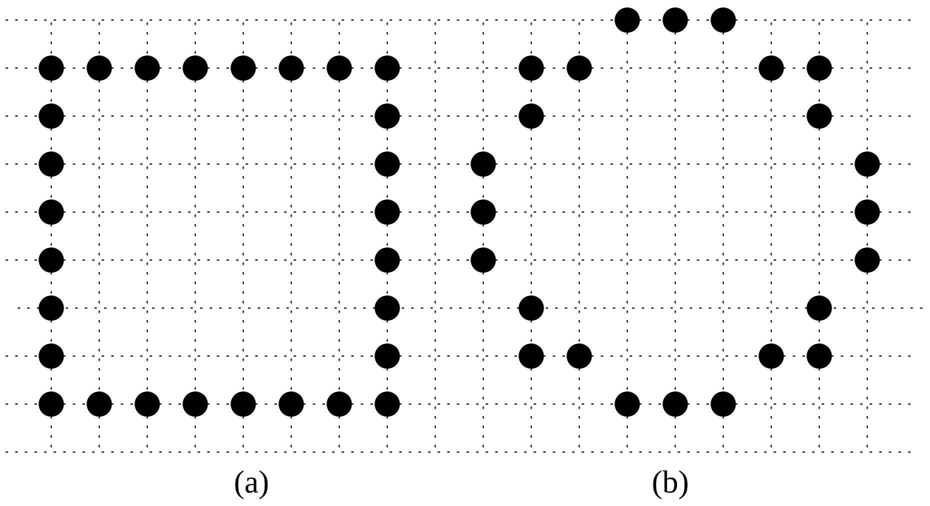


Figure 4.2.2: Square and circular masks.

We have used two different masks to detect motion sensors: square and circular. Figure 4.2.2 depicts 8x8 square mask and a circular mask of radius 4. Square mask contains 64 pixels whereas circular mask has 61 pixels. Circular mask is a better approximation than square masks, since pixels within a radius is considered. In real examples, the difference is not distinguishable when either of these masks is used. Fig. 4.2.3 shows motion sensors detected for a frame from a mobile & calendar frame by square and circular masks.

One problem in matching of the blocks is the significant displacement between two frames. In most cases, the motion between two consecutive frames is not huge.

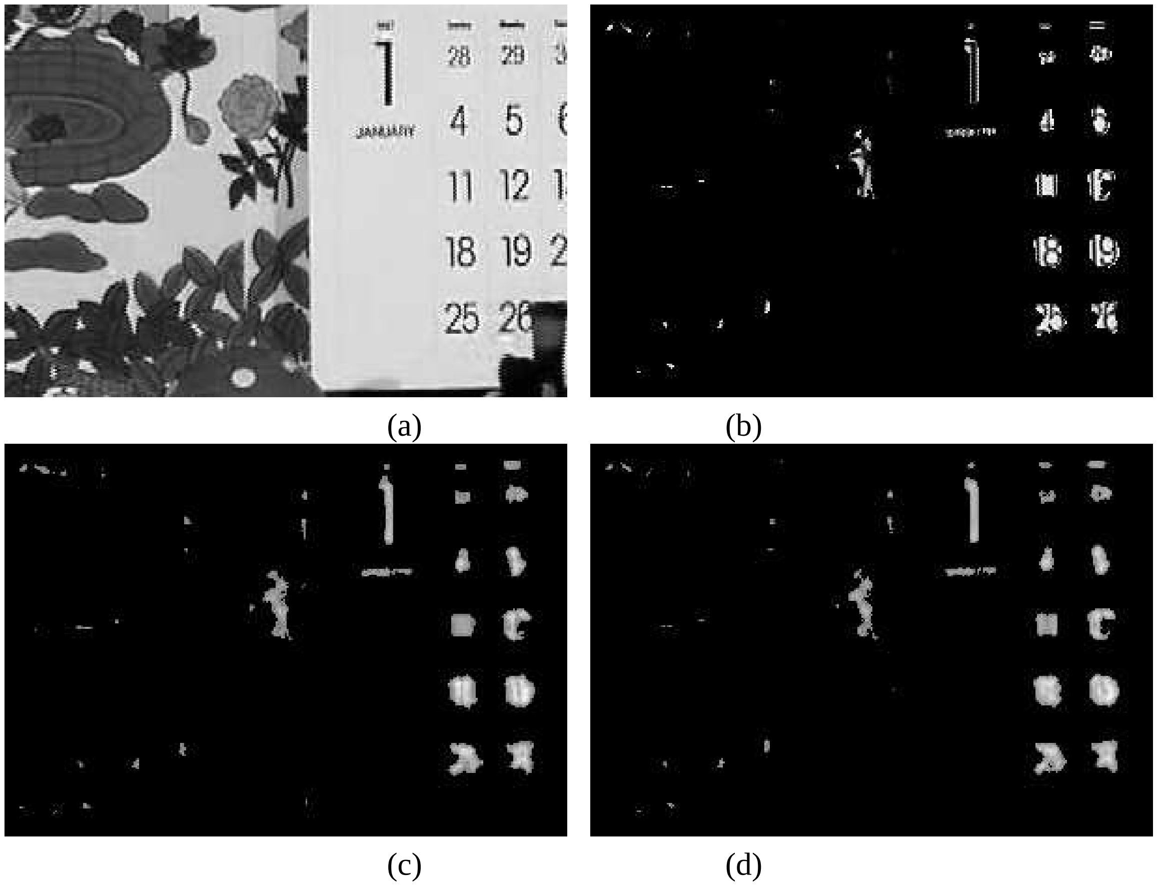


Figure 4.2.3: Mobile & Calendar example for motion sensors a) original frame b) motion sensors in the frame from circular mask c) application of square mask d) application of circular mask.

4.2.2 Optimization in Detection of Motion Sensors

Although motion sensor detection is performed at the beginning, the comparison of blocks is the most expensive part and it should be optimized. There are a couple of optimizations that can be performed. Since the goal is not to find all the motion sensors, there is no problem if some of them are missed. In most cases, a motion sensor has a neighboring motion sensor and an ordinary pixel has a neighboring ordinary pixel. Therefore, motion sensor detection can be performed at intervals. To increase the performance, every other pixel is skipped during detection.

Note that the distance between two blocks is symmetric. If the distance between two blocks is already computed, there is no need to compute the distance between those two blocks again. If the search distance is d, there are initially $(2d)^2 - 1$ SAD computations. Due to symmetry, this is reduced to $d^2 - 1$.

The sensitivity of a pixel will be high if it is a motion sensor. Otherwise, the pixel's region is similar to its surrounding. If the sensitivity is low when comparing the blocks, the further blocks do not need to be compared. That pixel can no longer be a motion sensor.

4.3 Global Motion Estimation Using Motion Sensors

There are different types of motion models that are used in GME depending on the camera operations and the structure of the scene. In this chapter, we detect camera motion that is parameterized by perspective motion model:

$$\begin{aligned} x_i' &= \frac{a_0+a_2x_i+a_3y_i}{a_6x_i+a_7y_i+1} \\ y_i' &= \frac{a_1+a_4x_i+a_5y_i}{a_6x_i+a_7y_i+1} \end{aligned} \tag{4.3.1}$$

where a_0, a_1, a_2, a_3, a_4, a_5, a_6, and a_7 are motion parameters and (x_i',y_i') is the transformed coordinate for (x_i,y_i). This model turns into affine motion when $(a_6 = 0,\ a_7 = 0)$, translation-zoom-rotation motion when $(a_4 = -a_3,\ a_5 = a_2,\ a_6 = 0,\ a_7 = 0)$, and translational motion when $(a_2 = 1,\ a_5 = 1,\ a_4 = 0,\ a_5 = 0,\ a_6 = 0,\ a_7 = 0)$.

The error between two frames can be declared as

$$\varepsilon = \sum^N e_i^2 \tag{4.3.2}$$

where $e_i = I'(x_i', y_i') - I(x_i, y_i)$, $I(x_i, y_i)$ is the intensity at (x_i, y_i) in the previous frame, and $I'(x_i', y_i')$ is the intensity at the transformed coordinate in the current frame. Error ε is computed for pixels overlapping in two frames.

The iterative descent methods are likely to be trapped in local minima when they try to minimize Equation 4.3.2. The hierarchical (iterative descent) approach is usually applied to detect the motion parameters. Initial estimation of translational parameters is necessary to avoid local minima. This method requires generation of low-pass filtering of a frame, computation of gradients and computing gradient descent for each layer. In our experiments, motion sensors approximately give the motion parameters without generation of hierarchical pyramid.

The mapping of a motion sensor is computed using block matching. For finding the best matching block, the full searching or n-step searching can be applied. Since the number of feature points is usually very few, the process of full searching is still fast and sometimes is desired if accuracy is needed. For perspective, affine, translation-zoom-rotation and translational motion at least 4, 3, 2 and 1 motion sensors are required, respectively. Extra motion sensors can be used to check the correctness of motion.

In our implementation, we choose 3 motion sensors to estimate the motion. Since 3 motion sensors are used, only the parameters for affine motion can be estimated. The parameters for translational and translation-zoom-rotation are subsets of these parameters. Since perspective motion is sensitive to slight changes in the parameters and iterative descent methods can be trapped in local minima, the initial guess of perspective motion parameters is not performed by motion sensors. The general structure of the motion estimation algorithm is depicted in Fig. 4.3.1.

If the motion detection by motion sensors is confirmed, affine motion is estimated using Levenberg-Marquardt(LM) iterative nonlinear minimization algorithm. Once the parameters are estimated for affine motion, these parameters are again fed into LM algorithm to estimate the perspective motion parameters.

If the motion sensors cannot uniquely identify a global motion, the motion estimation is determined as in [28]. After the motion parameters are obtained, motion sensors that do not conform to global motion are eliminated and not used in the subsequent motion detection.

It is possible that the motion sensors may be occluded by moving objects or may disappear from the scene due to camera motion. In those cases, motion sensors will significantly

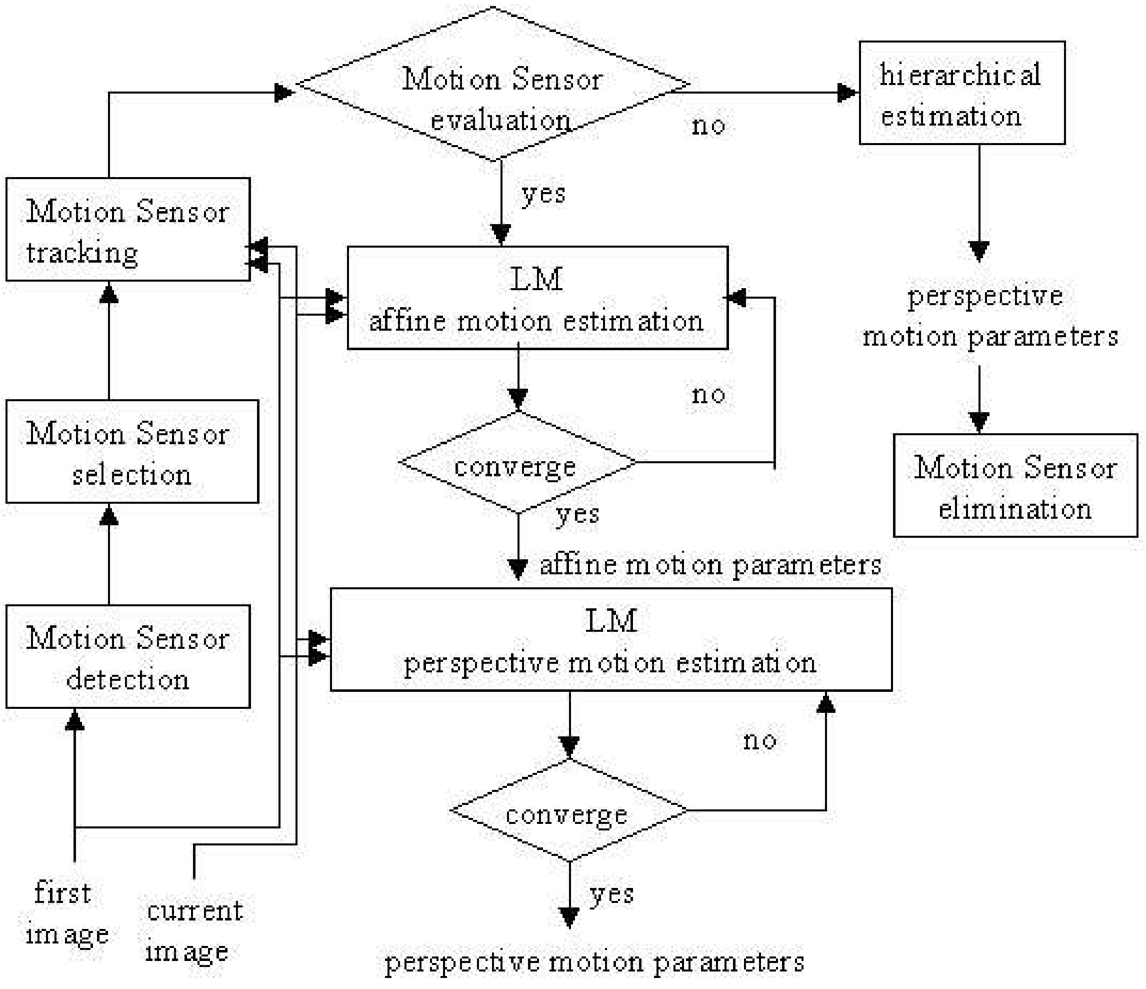


Figure 4.3.1: Global Motion Estimation Algorithm.

increase the error in Equation 4.3.2. The threshold t_{error} determines that the maximum average e_i^2 could be between consecutive frames. In our test environment t_{error} is chosen as 100. Once the error exceeds t_{error}, motion sensors are recomputed for the new frame.

4.4 Global Motion Estimation for Spatial Browsing

The accuracy of the motion parameters is important for accurate sprite generation. The scenes may have objects at different depth and shapes or moving objects. The motion parameters are affected by the moving objects and aperture problem. This causes some deviation from the original values of motion parameters. When the motion estimation is performed from frame to frame, the error accumulates and propagates to the later motion estimation and warping. Since the scene may not be modeled accurately by the motion model, it is not always possible to obtain the actual motion parameters. If the motion is estimated between each sequential frame, the error is accumulated for sprite generation when consequent frames are warped. Instead of using the original frame, we generate the frame from the sprite to use in global motion estimation [90]. Therefore, there will not be error accumulation. This process is shown in Figure 4.4.1.

The iterative descent methods are likely to be trapped in local minima. Our sprite generation method skips some of the frames to reduce blurring in the sprite. The motion estimation is performed between each sequential frame and also between frames at specific intervals. Motion estimation between farther frames is more prone to errors due to the initial estimation and possible large displacement. Accumulated motion parameters or relative motion with respect to the initial frame in the interval is a good approximation of the motion parameters. We decrease this entrapment by considering the relative motion

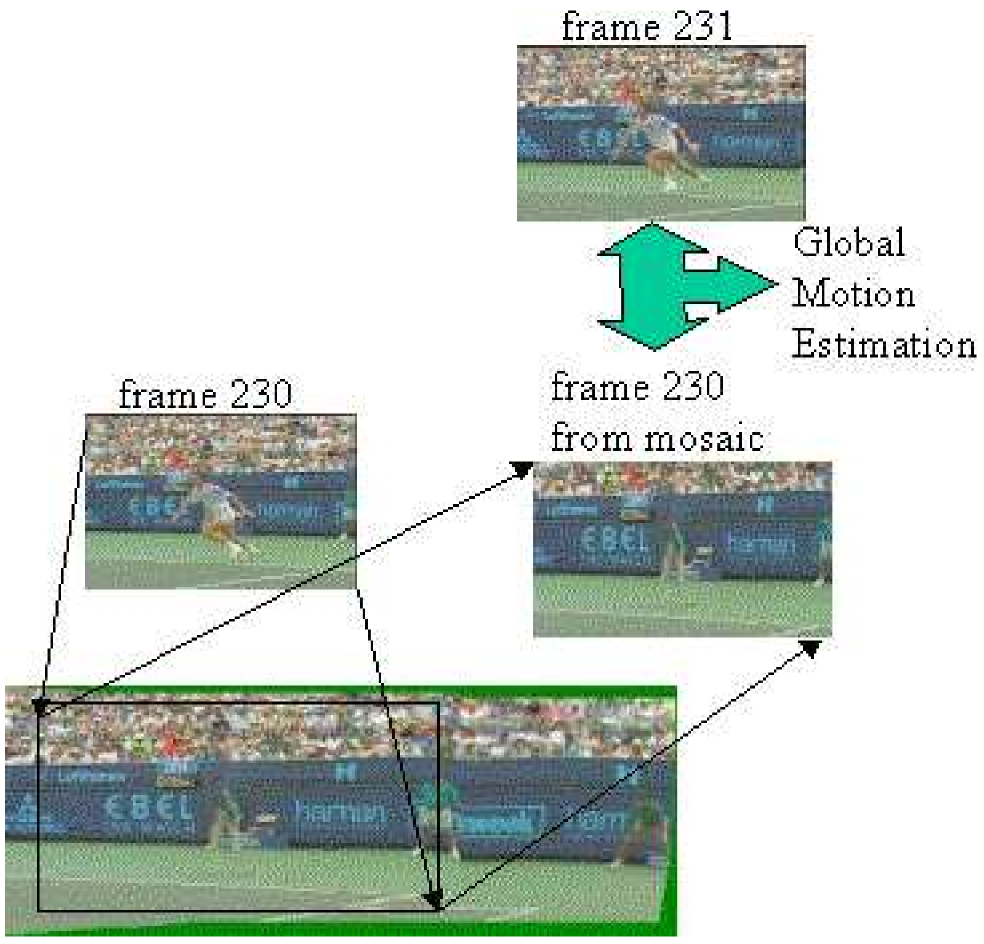


Figure 4.4.1: Global Motion Estimation.

between consecutive frames. So, there will not be an assumption on the upper-bound of translational parameters. We rather make an assumption on the speed of the camera movement. For example, if displacement is 20 pixels at frame f_t and the threshold on the camera speed is 5 pixels, the new expected displacement will be in [15,25]. This will reduce the error in initial estimation and remove the assumption on maximum displacement.

In matrix form, affine motion estimation can be written as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_2 & a_3 \\ a_4 & a_5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}. \tag{4.4.1}$$

More generally, this can be written as

$$v' = Mv + t \tag{4.4.2}$$

where M contains the motion parameters for the first matrix and t contains the translational parameters. The relative motion is computed as

$$v'' = M'(Mv + t) + t' = M'Mv + (M't + t') \tag{4.4.3}$$

where v'' is the vector for the new transformed coordinates; M' and t' hold the current motion parameters; and M and t hold the motion parameters up to the current frame.

If there is no segmentation mask is used and the scene includes moving objects, we use M-estimators [28, 90] to increase the robustness of motion estimation. In this case, the error is expressed as:

$$\sum^{N} \rho(e_i) \tag{4.4.4}$$

where $\rho(e_i) = e_i^2$ in the original formulation. Since this function gives more weight to large errors, it is biased by local motion (which are outliers for global motion). To decrease the effect of outliers, the truncated quadratic motion is used:

$$\rho(e_i) = \{ \begin{array}{ll} e_i^2, & \text{if } |e_i| \leq t \\ 0, & \text{if } |e_i| > t \end{array} \tag{4.4.5}$$

where t is a threshold selected according to the histogram of the errors.

4.5 Summary

In this chapter, we introduced a method to perform the global motion estimation method with motion sensors. If the video does not contain continuous motion, the existence of the motion can be detected by motion sensors. Moreover, motion sensors also give good approximation to the motion model parameters. This initial estimation reduces the number of computation at the levels of pyramid. Although initial detection of the motion sensors is costly, it is usually done once at the beginning and can be optimized by the methods given in Section 4.2.2. The GME can also be performed at intervals and accurate results are obtained.

Chapter 5

High Resolution Sprite Generation with Sprite Pyramid

The sprite generation methods benefit from recent global motion estimation (GME) methods, which yield almost accurate estimation of motion parameters. However, the generated sprites are usually more blurred than original frames due to image warping stage and errors in motion estimation. The transformed coordinates resulting from GME are generally real numbers whereas images are sampled into integer values. Although GME methods generate proper motion parameters, a slight error in motion estimation may propagate to subsequent sprite generation steps. In this chapter, we propose a method to generate clearer sprites from video. The temporal integration of images is performed using the *histemporal* filter based on the histogram of values within an interval [11]. The initial frame in the video sequence is registered at a higher resolution to generate high resolution sprite. Instead of warping of each frame, the frames are warped into the sprite at intervals. This reduces the blurring in the sprite. We also introduce a new sprite called *conservative sprite*.

The ordinary sprite generation techniques focus on camera movement, accurate motion estimation, alignment, and integration. These techniques ignore the resolution of original images and the regenerated images from the sprite are likely to have lower resolutions than the original ones. Especially, if the scenes have finite depth and zoom-in and zoom-out operations occur, the segments of the scene are captured at different resolutions. The traditional sprite generation methods either blur the sprite by integrating lower resolution segments or use unnecessary large storage for the sprite. The *sprite pyramid (or layered sprite)* allows efficient storage of images or video clips of overlapping scenes at different resolutions. Moreover, the images or video frames can be reconstructed from the sprite pyramid at the necessary resolutions.

In this chapter, we firstly propose a method for generating high resolution sprite from video. The frames are integrated using the *histemporal* filter. The *histemporal* filter is a generalized filter and keeps the histogram of values that map to a specific interval. The initial sprite is maintained at a higher resolution to reduce the blurring due to real-valued transformed coordinates. The frames are warped into the sprite at intervals. Since warping of frames is performed using bilinear interpolation, a low-pass effect is introduced. Therefore, ignoring unnecessary frames yields clear sprite generation. After developing our method, experiments are conducted on standard MPEG test sequences.

A *sprite pyramid (or layered sprite)* allows the regeneration of the video at the proper resolutions. Each layer of the sprite pyramid corresponds to a different resolution. The sprite pyramid allows the regeneration of different segments at different resolutions as they were captured. The sprite pyramid is created if the scene has finite depth and there are zoom-in and zoom-out operations.

This chapter is organized as follows. High resolution sprite generation and *histemporal* filter are presented in Section 5.1. Section 5.2 explains the structure of the sprite pyramid. The experiments and results are reported in Section 5.3. The last section summarizes the chapter.

5.1 Sprite Generation

5.1.1 Histemporal Filter

The linear temporal filters like averaging, recursive filters like Kalman filter [77], and order-statistic filters like median filters have been used for noise reduction or removal in image sequences. Temporal averaging yields a blurred sprite, if the video includes moving objects or motion cannot be estimated accurately. Median filters require enormous storage to detect the temporal median filter. Moreover, temporal median may yield erroneous results, if the expected median can take several values. For example, the frequency of pixel values 100 and 101 is 20 and 22 for a pixel coordinate in the sprite, respectively. Although this difference may result from the illuminance change in the environment, they are treated as different. If the frequency of another pixel value is 25, this value will be chosen by mistake. In fact, averaging (of 100 and 101) would yield a better result. *Histemporal* filter is a *temporal* filter that is based on the *histogram* of intensity values within a specific interval.

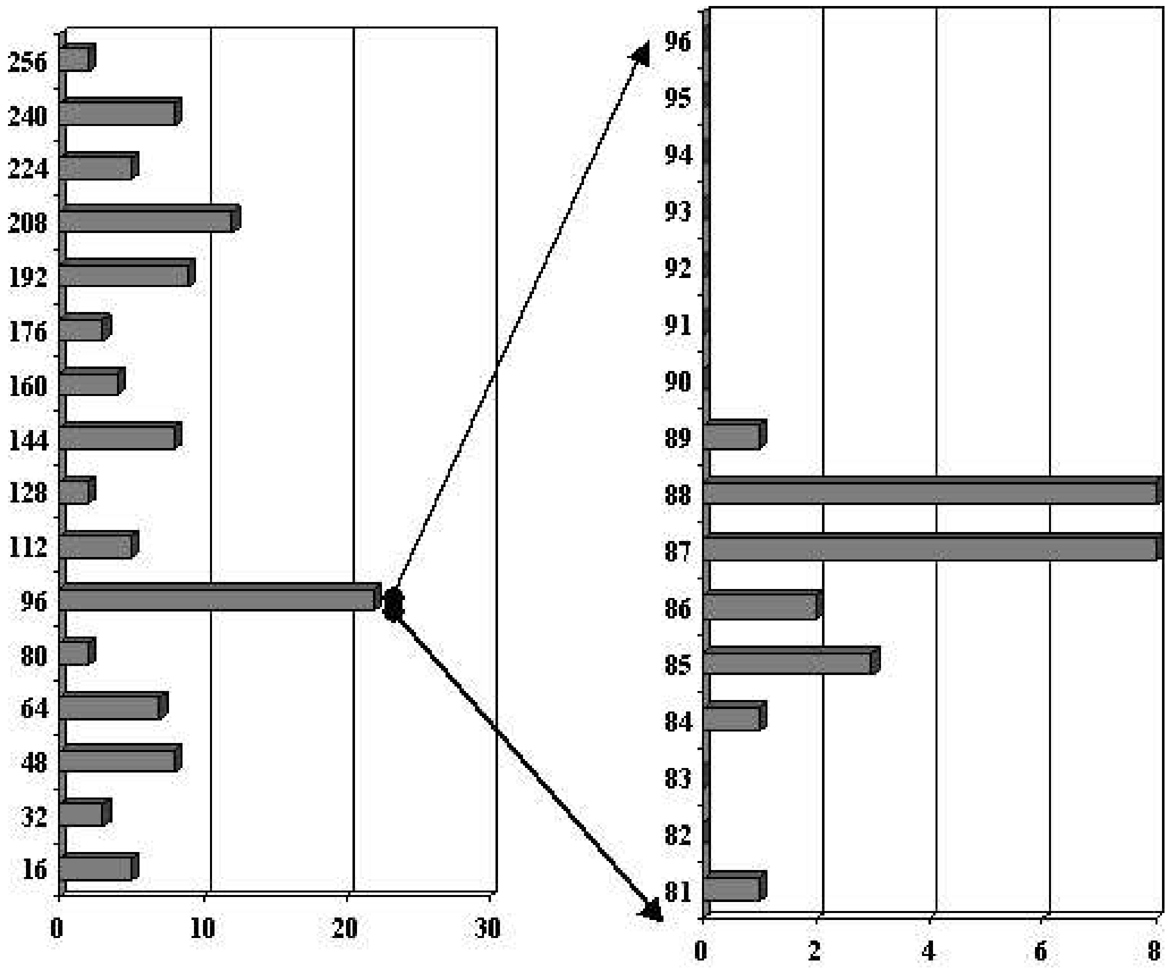
The $interval$ determines the precision of temporal integration in sprite generation. For a 8-bit per pixel gray-scale image, all the pixels lay in $[0, 255]$. There will be $\lceil \frac{256}{interval} \rceil$ slots in the histogram. If $interval$ is 256, histemporal filter becomes temporal averaging. If $interval$ is 1, the temporal interval becomes temporal median filter.

Two data structures are used to obtain the histemporal filter: *frequency* array and *average* array. *Frequency* array keeps the frequency of each interval of the histogram. As the frames are processed, the frequency of an interval is increased for each pixel value belonging to the interval. *Average* array maintains the average of the values as new values are processed for each slot. Histemporal filter returns the average value of the interval having the highest frequency. Figure 5.1.1 (a) shows a histogram where *interval* is 16. The interval $[81,96]$ has the highest frequency. Figure 5.1.1 (b) displays the frequencies of the values that lay in this interval. During histemporal filter computation, the average of these values is computed as they arrive.

5.1.2 Conservative Sprite

Different representations of mosaics like static, dynamic, and synopsis mosaic have been investigated in [47]. Direct methods are applied to align images and to generate the mosaic. A high resolution mosaic is generated by sliding the mosaic and warping the next frame into the mosaic [89]. Since warping occurs for every frame, the generated mosaic can still be blurred. Moreover, temporal integration methods are used according to the type of the mosaic that will be generated. The temporal integration methods also cause blurring in the mosaic.

The sprite generation is a lossy process and the resulting sprite is usually more blurred than the original sprite. This results from the inaccuracy of the motion estimation and warping stage. Especially, if the camera in the actual scene is moving fast as in between frames 230 and 270 of ’stefan’ MPEG test sequence, the camera cannot capture the environment at a good quality. Moreover, warping blurs the original image and integration of the blurred image into the sprite worsens the sprite. Figure 5.1.2 shows an example of blurred sprite



(a) Histogram for intervals

(b) Weigted average of values in the interval are taken

Figure 5.1.1: Histemporal filter.

generated from frames 260 to 280 of 'stefan' video. Especially note the blurring in the spectators area.

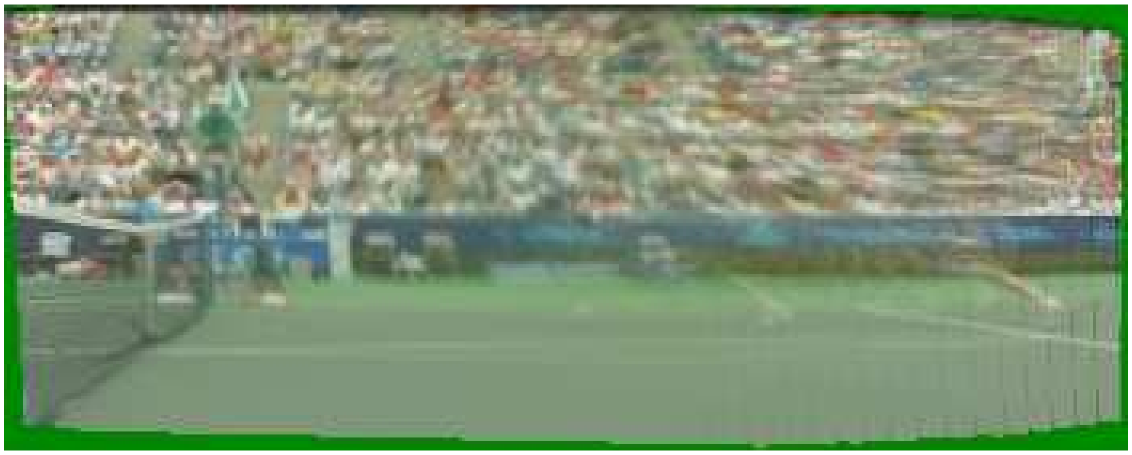


Figure 5.1.2: Blurred sprite generated from frames 260 to 280 of 'stefan'.

Static sprite displays the stationary parts in the scene. The dynamic sprite shows the sprite with the most recent frame. Synopsis sprite contains shadows of the moving objects. All these sprite generation types cause blurring of the image. We will define a new type of sprite termed as "conservative sprite" to avoid blurring of the sprite. However, conservative sprite generation requires the segmentation mask for moving objects. In conservative sprite generation, the new pixel from the new image is integrated into the sprite if no pixel was integrated onto that location before. Figure 5.1.3 shows the conservative sprite generated from frames 260 to 280 of 'stefan' video.

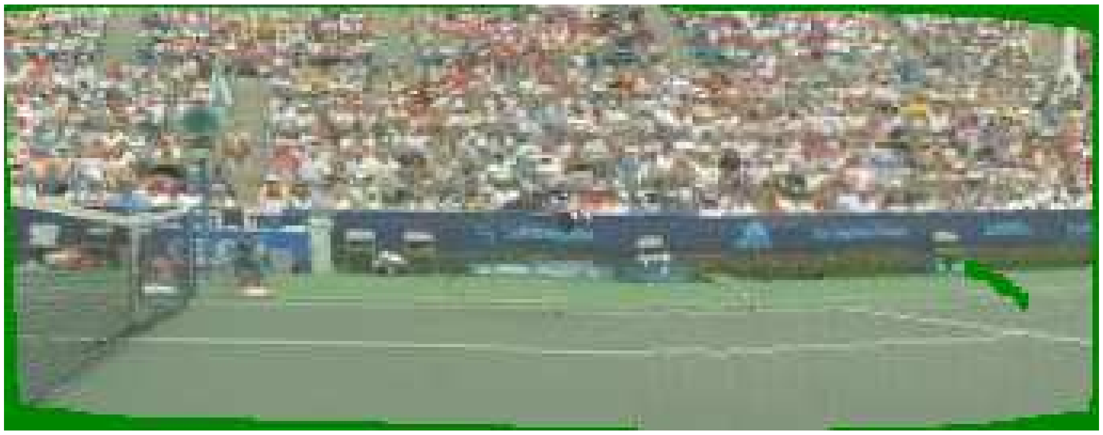


Figure 5.1.3: Conservative sprite generated from frames 260 to 280 of ’stefan’.

5.1.3 High Resolution Sprite Generation

Motion parameters that are obtained from Equation 4.3.1 are usually real numbers and yield real-valued transformed coordinates. The original images are sampled into integer domain. The ordinary techniques create a sprite having a resolution of the initial frame in the sequence. The pixel locations in the sprite may not correspond to the integer-valued pixel locations in the new frame. Approaches like bilinear interpolation are used to estimate the pixel value at the location. Bilinear interpolation takes the weighted average of the closest pixels and blurs the image.

A high resolution video mosaicing approach is proposed in [91]. A high resolution mosaic is generated where a mosaic also contains half-pel data. When a new frame is processed, a shift (diagonal, vertical, or horizontal) on the mosaic is assumed, and the frame is warped into the corresponding area in the mosaic. This usually preserves the original sharpness of the image. However, this approach does not consider the precision of the transformed coordinates and warping still occurs at a low resolution because of shifting. In our case, warping occurs at high resolution (Figure 5.1.4). Every pixel in the warping region is updated during warping.

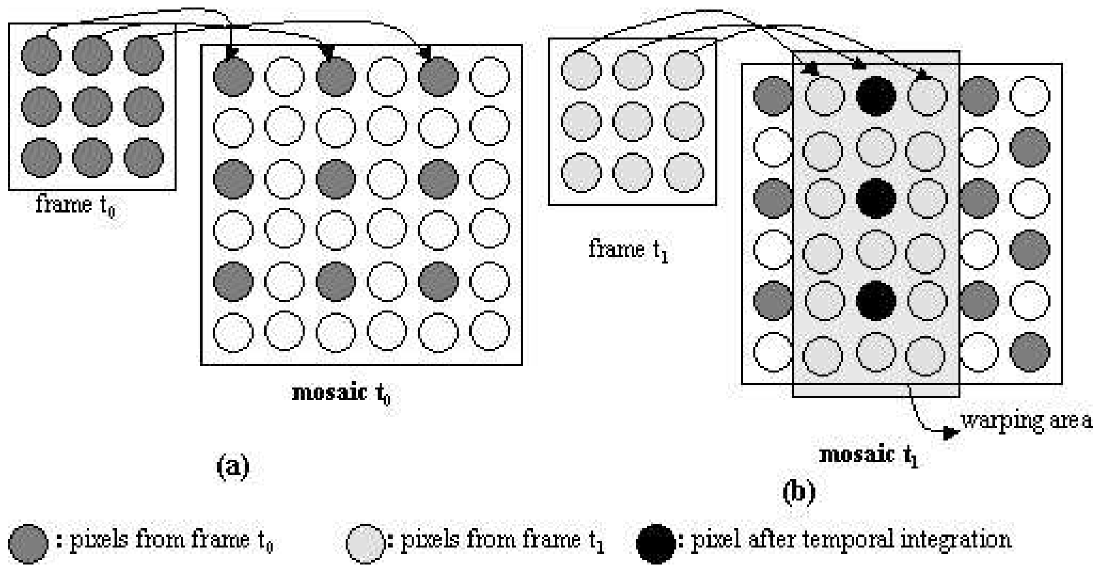
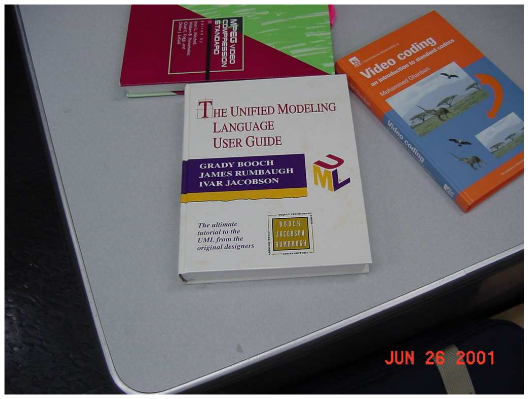

frame t_0
mosaic t_0
(a)
frame t_1
warping area
mosaic t_1
(b)
: pixels from frame t_0
: pixels from frame t_1
: pixel after temporal integration

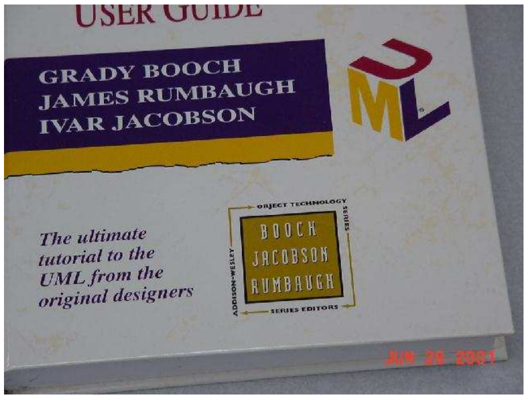
Figure 5.1.4: High resolution sprite.

The motion parameters are also affected by the moving objects and aperture problem. This causes some deviation from the original values of motion parameters. When the motion estimation is performed from frame to frame, the error accumulates and propagates to the later motion estimation and warping. In addition, warping at every frame also introduces blurring. Thus, instead of warping at every frame, the frames are warped into the sprite at intervals. But the motion estimation has to be performed for each sequential frame. The previous frame is mapped from the sprite to avoid error accumulation. Three thresholds are used: maximum accumulated displacement (mad), maximum scale factor (msf) and maximum interval length (mil). The motion between consecutive frames are accumulated until the displacement is less than *mad* and scale (zooming) factor is less than msf. Otherwise, the motion between the first frame and the last frame in the interval may increase significantly, and motion estimation methods may yield less accurate parameters. If there is no significant motion in the sequence, the relative motion is computed for at most *mil* frames. This upper bound is needed to remove the objects from the background sprite. The frame is also warped into the sprite when the direction of camera motion changes.

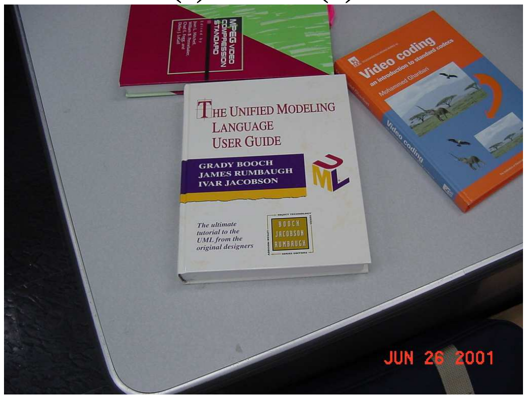
5.2 Sprite Pyramid

In traditional mosaicing methods, mosaics are generated by mapping onto a predetermined single space. The order of images is important in mosaic generation. In most cases, the images are mapped according to the first image in the sequence. If the image has the lowest resolution, then a low resolution mosaic is generated and if the first image has the highest resolution, a high resolution mosaic is generated. In the first case, if the images are generated from the mosaic, they will have lower resolutions than those of the originals. For example, Figure 5.2.1 gives an example of such an image alignment where the first image has a lower resolution. Figure 5.2.2 shows an example of image regeneration from a low resolution mosaic. If the first image has high resolution, after the images are aligned, the





(a) (b)



(c)

Figure 5.2.1: Image alignment for different resolutions: (a) first image (b) new image (c) mosaic

final mosaic will be huge to reserve the resolution of the first image. The goal is to provide a method to generate mosaic without losing resolution while maintaining efficient storage.

A sprite pyramid consists of L layers $(0 \leq l < L)$, where the lowest layer contains the highest resolution and the highest layer contains the lowest resolution. Laplacian pyramid [19] is a hierarchical way of representing an image usually at low resolutions at the high levels and high resolutions at the low levels. Each layer contains the same image at different resolutions. Since the sprite is not viewable at all resolutions, some layers of the sprite pyramid

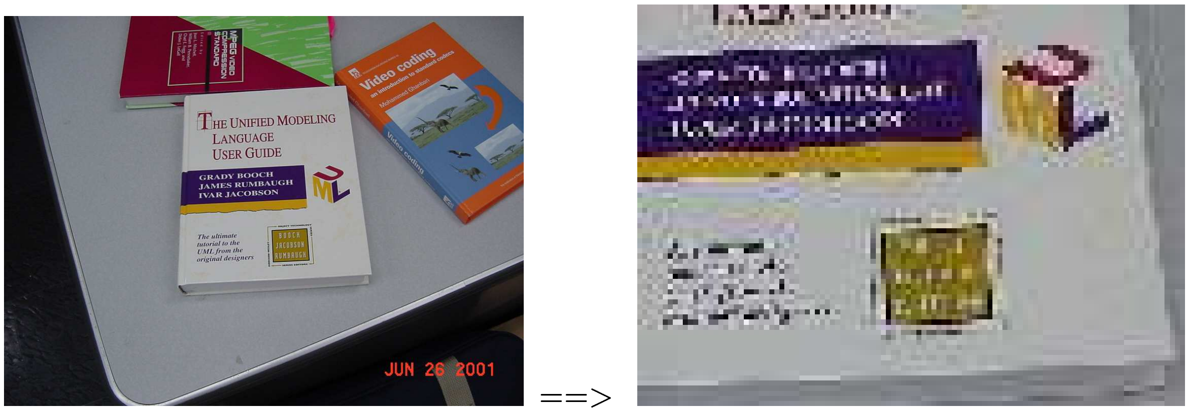


Figure 5.2.2: Regeneration.

may contain images having holes. Irregular shapes occurring as holes are caused by the rotation of the camera. A hole also occurs when a segment of an object is not captured at that resolution. Existence of the holes is the main difference from traditional image pyramids used in the literature where each layer contains an image at different resolution. Each layer l of pyramid σ has a zooming factor σ_l. The structure of a sprite pyramid is shown in Figure 5.2.3. The advantage of this sprite pyramid is that it keeps all the data visible at its resolution. So, when a video frame or image has to be regenerated, the video object is generated from the layers having closer zooming factors.

The generation of sprite pyramid from a group of images can be performed efficiently since the number of images is few or the camera motion is not contiguous. In a video, the camera motion is usually continuous. Creating a layer for each frame is time consuming and not efficient. If zooming is not significant, frames are mapped onto the current layer. If zooming is greater than 1 and significant, it is mapped onto the lower layer. Otherwise,

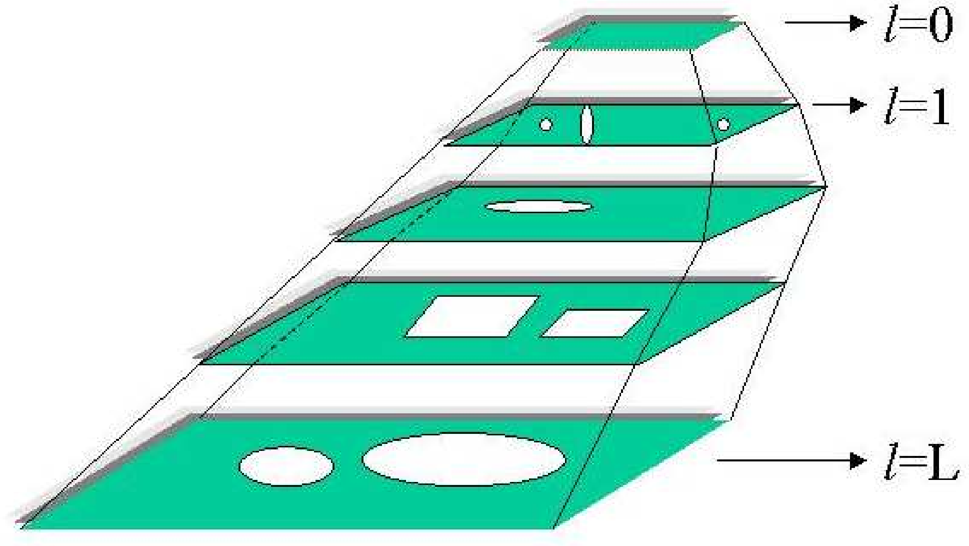


Figure 5.2.3: The sprite pyramid.

it is mapped onto the upper layer.

In most cases, the reason of zoom-in is to focus on the interesting object in the scene. Therefore, mapping can be ignored until the zooming operation stops. In that case, there is an interesting object and that scene has to be kept at high resolution. If new parts of the scene are visible during zooming, those regions are mapped onto the current layer of the sprite.

All the images are aligned at their own layers. The images are also aligned at the lowest resolution. At level 0, the sprite contains the largest view of the scene. This sprite may also be used to display the big picture of the object.

5.3 Experiments

5.3.1 High Resolution Sprite

In our experiments, the resolution of the sprite is twice as the initial frame of the sequence, thus resulting in half-pel accuracy. The *mad* and *mil* are both selected as 10. Figure 5.3.1 shows an ordinary blurred sprite generated from ’coastguard’ MPEG test sequence. If the motion can be modeled using translational model, the images can be warped according to the precision of transformed coordinates. MPEG-4 test sequence ’coastguard’ can be modeled using translational model. Figure 5.3.2 shows the high resolution background sprite generated after 300 frames. No segmentation mask is used in the generation. The water texture is smoothed because of temporal texture, and has been removed from the sprite. The right side of the figure includes parts that are not filled by frames. Therefore the right side looks darker. These locations are filled with bilinear interpolation. The smoothed regions in the ordinary sprite are clearly visible in the high resolution sprite.

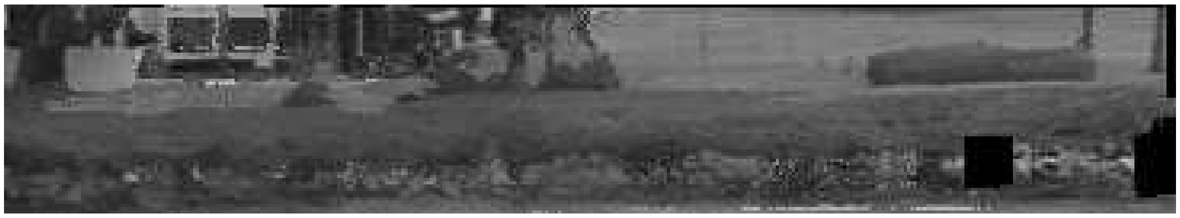


Figure 5.3.1: Ordinary Sprite.

There are no standardized performance tests for generating sprites. The most common

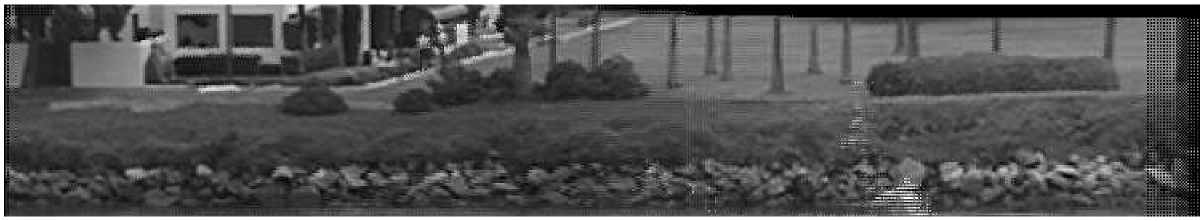


Figure 5.3.2: High resolution sprite from coastguard.

method is averaging PSNR values for a video. Although PSNR is a good indication of similarity between images, average of PSNR values is not always a good measure for video. Figure 5.3.3 shows the sprite generated for 'foreman' MPEG test sequence from frames 195 to 240. The corresponding PSNR values for frames that are generated from high resolution sprite and ordinary sprite are given in Figure 5.3.4. We have used affine motion model for 'foreman' sequence.

5.3.2 Sprite Pyramid

To show the effectiveness of the sprite pyramid, we give an example from a distance education application. Digital lectures contain zoom-in and zoom-out operations. Zoom-in usually happens when the instructor points an important data on the board or the slide show. When zoom-in happens, important data are displayed. When zoom-out happens, the general view is presented. Figure 5.3.5 shows three scenes from the lecture before zoom-in, after zoom-in and after zoom-out.

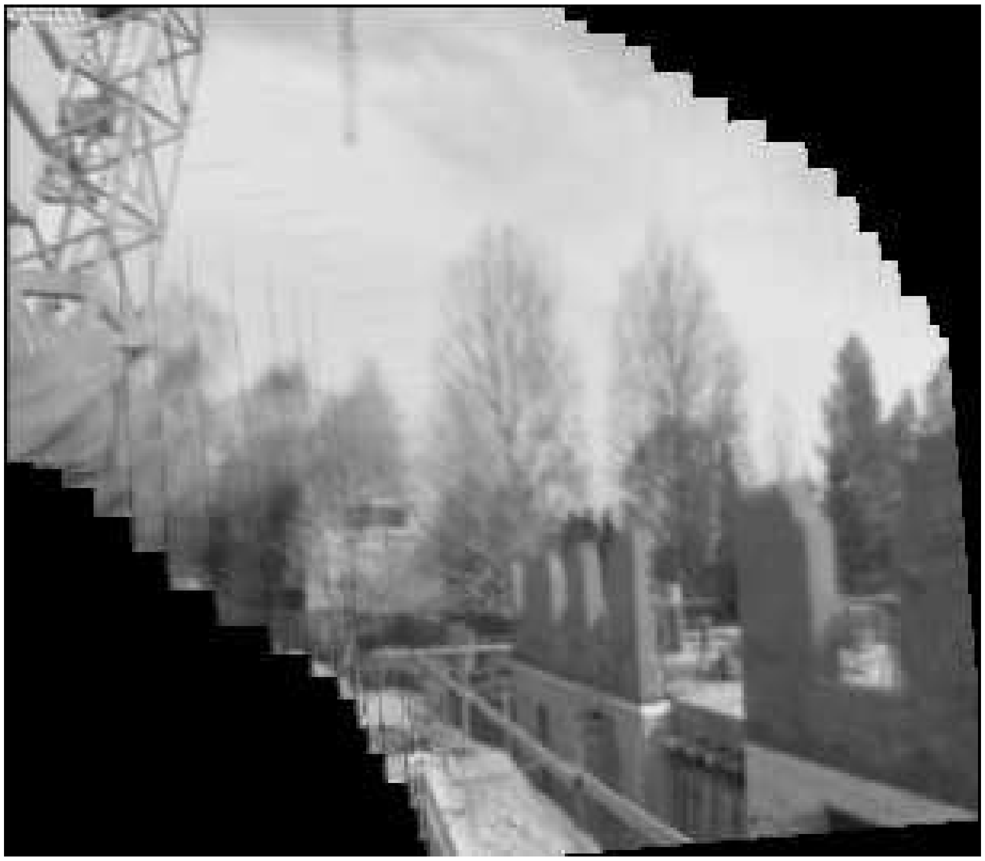


Figure 5.3.3: Sprite generated from ’foreman’.

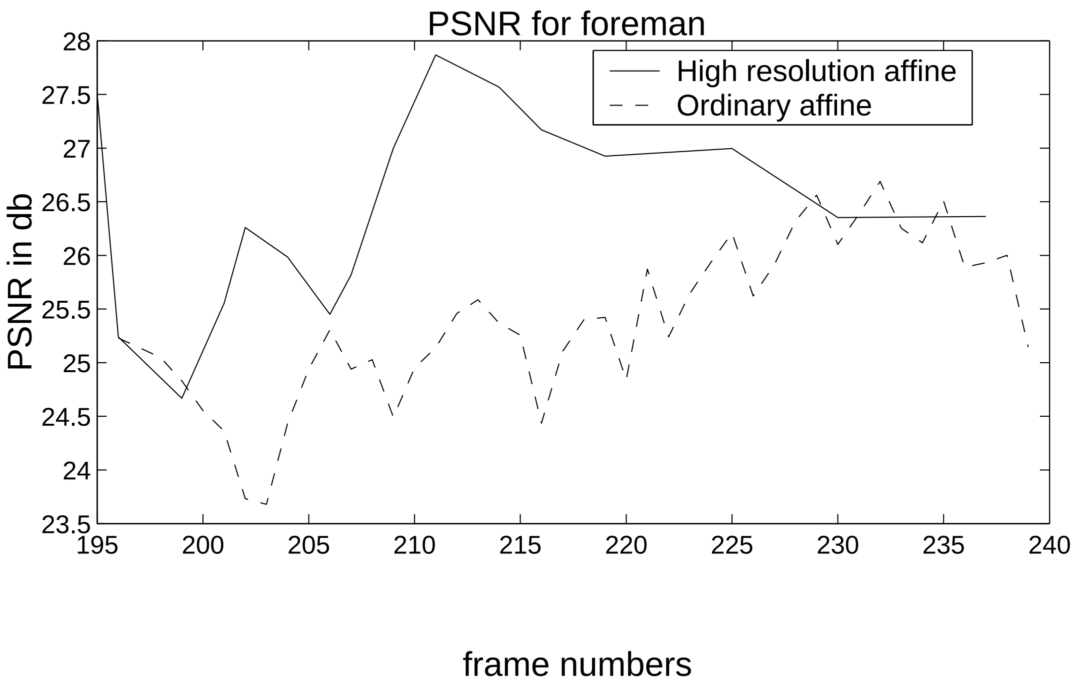
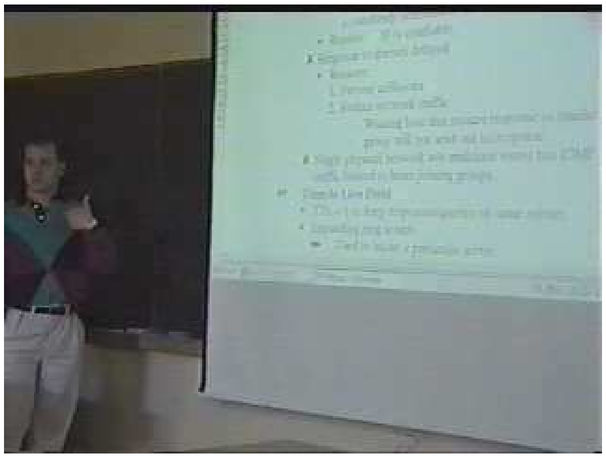
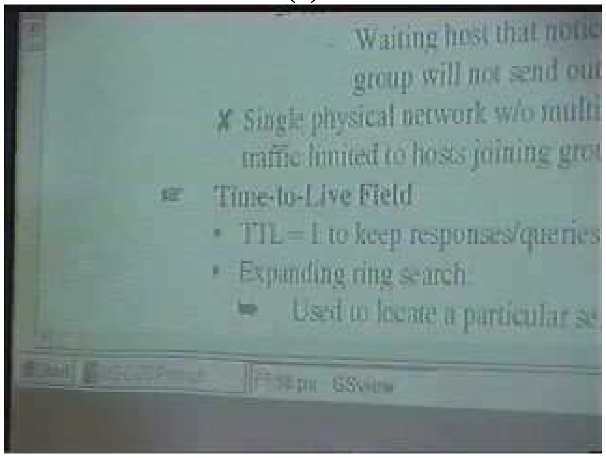


Figure 5.3.4: PSNR for foreman sequence.

Figure 5.3.6 displays the a_2 parameter of affine motion model with respect to the initial frame in the clip. It is about 20 seconds clip starting from frame 32400 to frame 32900. We only present a_2 since a_2 is nearly equal to a_5; and a_3 and a_5 is very close to 0. In the figure, p_1, p_3, p_5, and p_7 are peak points (local maxima). At these points, zoom-in reaches its final point. These are the possible points that the significant object is being captured at the required resolution. In the figure, p_2, p_4, p_6, and p_8 are the local minima where the zoom-out stops. In region R_2, there is a continuous zoom-in, so this part can be ignored in sprite generation. In R_4, there is a continuous significant zoom-out. In region R_3, p_5 is very close to the neighboring local minima, i.e., p_4 and p_6. There is no significant zoom-in operation at this part. In region R_3, p_4 and p_6 are very close to their neighboring local

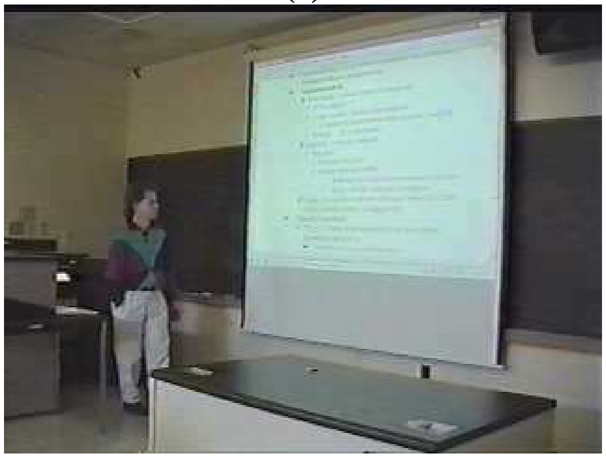


(a)



Waiting host that notic
group will not send out
✗ Single physical network w/o multi
traffic limited to hosts joining gro
☞ Time-to-Live Field
• TTL = 1 to keep responses/queries
• Expanding ring search
➥ Used to locate a particular se

(b)



(c)

Figure 5.3.5: Zoom operations in a lecture (a) frame 32400 (before zoom-in) (b) frame 32550 (after zoom-in) (c) frame 32900 (after zoom-out)

maxima. Therefore, there is no significant zoom-out. Region R_3 focuses on the important object in the scene. In regions R_1 and R_5, the general view of the environment is shown in the clip.

Separate sprites are generated for R_1, R_3, and R_5, and form the layers of the sprite. In traditional sprite generation, single sprite would be generated. Since temporal integration is performed at different resolutions, the resolution of the sprites is not degraded (blurred) by integration of lower resolution frames. In Figure 5.3.6, a_2 at p_7 is nearly three times of a_2 at p_8. If the sprite were generated, the final sprite would be 9 (3x3) times larger than the original frame. In this case, there will be three layers of sprite: for R_1, R_3, and R_5. R_2 and R_4 only contributes to the sprite if they cover some segments that are not covered in R_1, R_3, and R_5. Since frames in R_1, R_2, R_4, and R_5 are not integrated on frames in R_3, the sprite for R_3 is not blurred by low resolution data integration.

During our experiments, we have faced problems when using traditional mosaic generation methods. Sprite generation from the whole sequence is erroneous since the image (at high resolution) is considered as a pattern in a low resolution image. During error computation and motion parameter estimation, there are candidate regions in low resolution image which give less error than the original region. In that case, sprite cannot be generated properly and long-term sprite generation methods [90] fail. On the other hand, frame-based motion parameter estimation yields error accumulation in sprite generation. Figure 5.3.7 shows

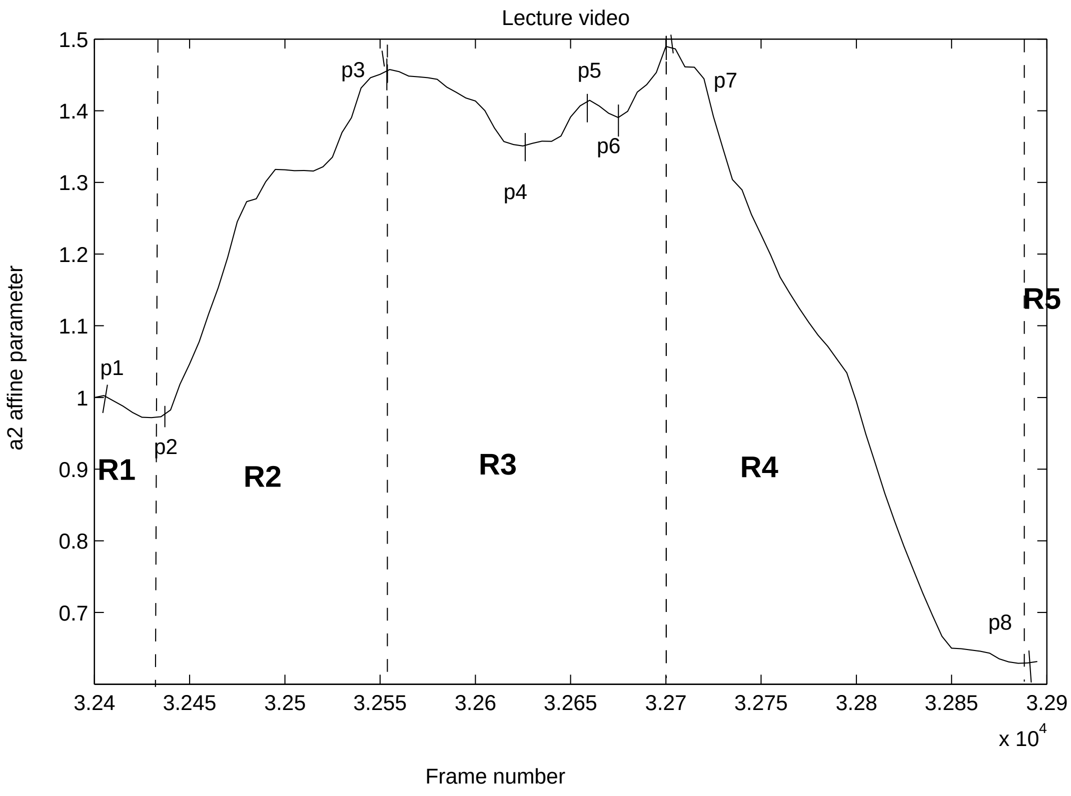


Figure 5.3.6: Affine motion parameter a_2 from a video clip of a distance education video.

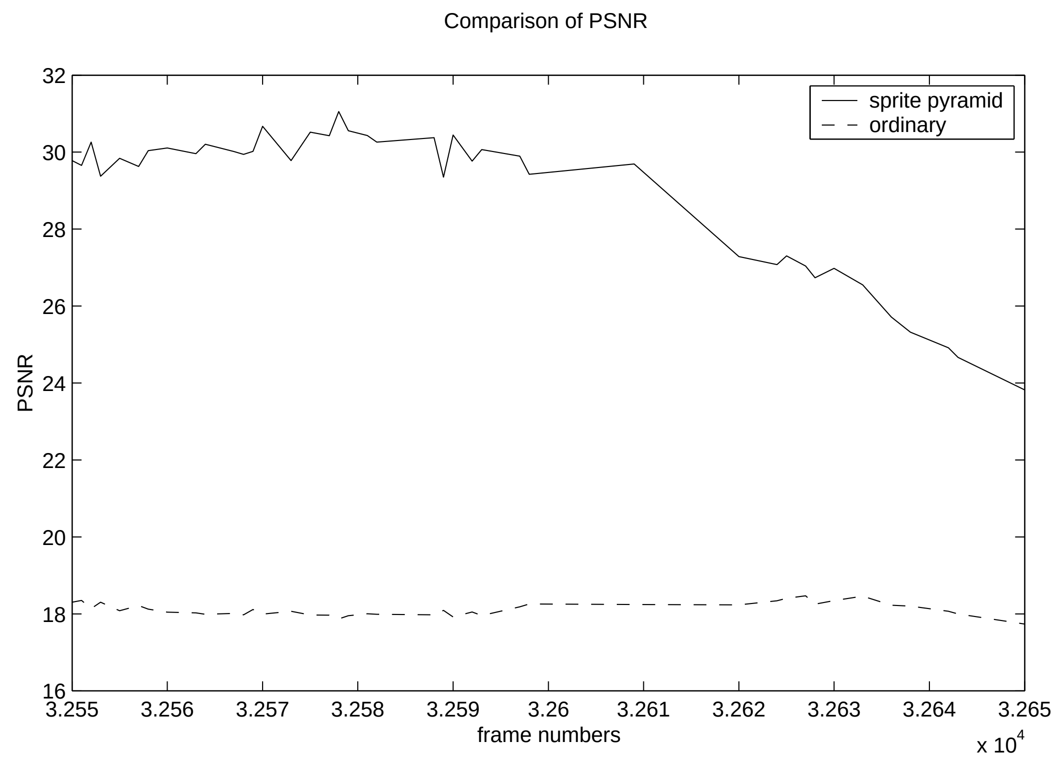


Figure 5.3.7: Comparison of PSNR values for ordinary sprite generation and from sprite pyramid.

the comparison PSNR values for the high resolution sequence of a video by using sprite pyramid and an ordinary sprite generation technique. This figure shows the efficiency of using sprite pyramid.

5.4 Summary

In this chapter, firstly, we have presented a method for high resolution sprite generation from video. Motion estimation is performed between each consecutive frame not to miss visible areas in the sequence for sprite generation. We introduce a new sprite named as *conservative* sprite that is clearer than traditional sprites. The blurring in the sprite generation is reduced by warping at intervals and at a higher resolution. Although high resolution sprite warping increases elapsed time, this is compensated by warping at intervals.

We have also presented the sprite pyramid for videos and images having finite-depth scenes. In applications like distance learning, zoom-in and zoom-out are common camera operations. The original sprite is only appropriate for applications having no zooming. Traditional mosaicing techniques usually ignore these basic operations and cause blurred or very large mosaics. This problem can be resolved by mapping the frames on a pyramid where layers reflect different resolutions. More importantly, this sprite pyramid model allows the regeneration of the video frames and objects at the resolution they were captured.

Chapter 6

VideoCruise: Spatial Browser for Recorded Digital Video

Spatial browsing of video documents enables the viewers to visualize interesting objects from their perspective. In this chapter, we introduce a system, termed as *VideoCruise*, to spatially browse the video documents. VideoCruise requires accurate global motion estimation and accurate sprite generation. Although there have been methods developed to perform these operations, the output of these techniques can only be used with motion compensation. To measure the quality of the sprite, a new measure called sharpness is used to estimate the blurring in the sprite. After motion parameters are detected and the precise sprite is generated, VideoCruise manipulates the sprite and the original frames to allow interactive spatial browsing which enables panning, tilting, and zooming. We have conducted experiments on standard MPEG-4 test sequences and discussed the output of our experiments.

Although spatial browsing is enabled in virtual environments, the scene is generated by using computer graphics tools as in interactive walk through applications [103]. However, these applications do not give the feeling of browsing through real images. In real life, it is not possible to define every kind of scene by geometric primitives. Another problem is that the environment has to be well-defined. In [2], a spatial navigation system is proposed by modeling the original video using VRML (Virtual Reality Modeling Language). The user will be able to navigate using VRML and access spatial locations. There have been intelligent cameras developed to follow the predefined objects. An intelligent camera system is developed to track lecturer and audience in [79]. Panoramic cameras are used to capture the environment in [96, 53]. The region of interests are detected from the panoramic scene and these parts are presented [96]. This approach is not applicable to previously developed videos. A spatially indexed camera is explained for navigation in [53].

Our goal is to enable spatial browsing of recorded video. We do not assume that the scene is predefined. We consider scenes that are captured using a single camera. The user can browse the scene by using camera operations like pan left and right, tilt up and down, and zoom in and out. The background mosaic or sprite has to be extracted to provide these interactions. However, the ordinary sprite generation methods have some handicaps to be used directly in spatial browsing. For example, in MPEG-4 [87], the generation and playback of the new generated video depend on the motion compensation algorithms since the motion cannot be estimated accurately. The user can spatially browse the video and release browsing to see the actual video at any frame display. We call this system as *VideoCruise*. In this chapter, we describe the system and evaluation of the experiments.

This chapter is organized as follows. In the following section, the components of the system are explained briefly. The spatial browsing issues are expressed in Section 6.2. The experiments are explained in Section 6.3. Section 6.4 discusses the limitations and the future work. The last section concludes our chapter.

6.1 System Components

VideoCruise has two components: preprocessing module and spatial browsing module (Figure 6.1.1). In the preprocessing phase, the global motion estimation between frames and the generated sprite is performed; and the sprite is generated. Since the sprite is used in spatial browsing, our goal is to generate a high resolution sprite while reducing blurring.

The motion parameters for each frame is stored and made available for spatial browsing. The user can browse the video using panning, tilting, and zooming operations. The frames

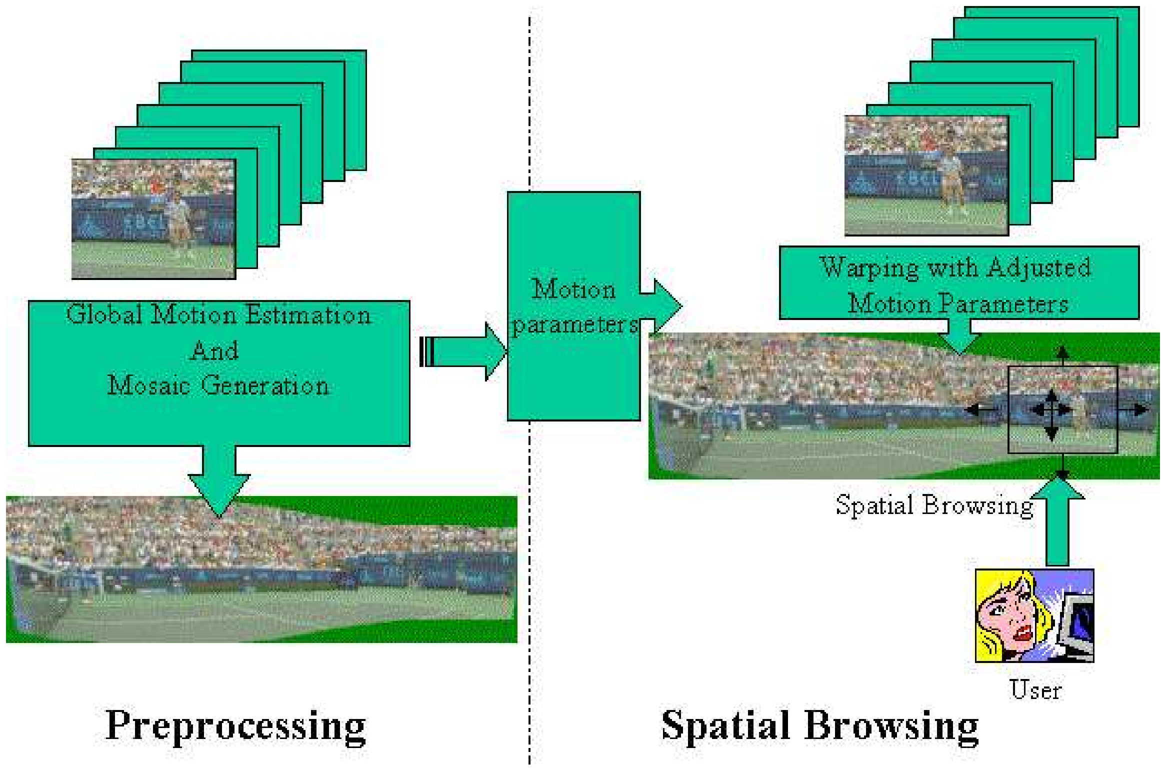


Figure 6.1.1: Components of VideoCruise.

and the sprite are merged by adjusting motion parameters and enabling browsing.

The videos that we experiment have color components. Each pixel in a frame has YUV components. We have performed GME only on luminance values (Y component). The U and V components usually do not contribute much to the GME and moreover, it brings additional processing cost.

6.2 Spatial Browsing

Spatial browsing requires integration of sprite and the frames. In [96], since the original scene is a panoramic scene, the region of interest from the scene is detected and presented to the user. In their case, there is no need to integrate frames and the background sprite. In our case, the sprite is generated using the frames and it is important how these frames are related spatially.

The video can be seen in two ways at any time of the video: original display and with camera controls. The user can gain the camera control at any frame display at any time. The user can also release the camera control and the watch the original video. When the user gains the camera control, the camera can be moved left, right, up, and down and zoomed in and out (Figure 6.2.1). This yields camera stabilization and spatial browsing.

Spatial browsing can be performed using the segmentation masks of the objects if they are available or using the complete frames. When objects are available, they are warped into the background. If the objects are not available, original frames are warped into the background.

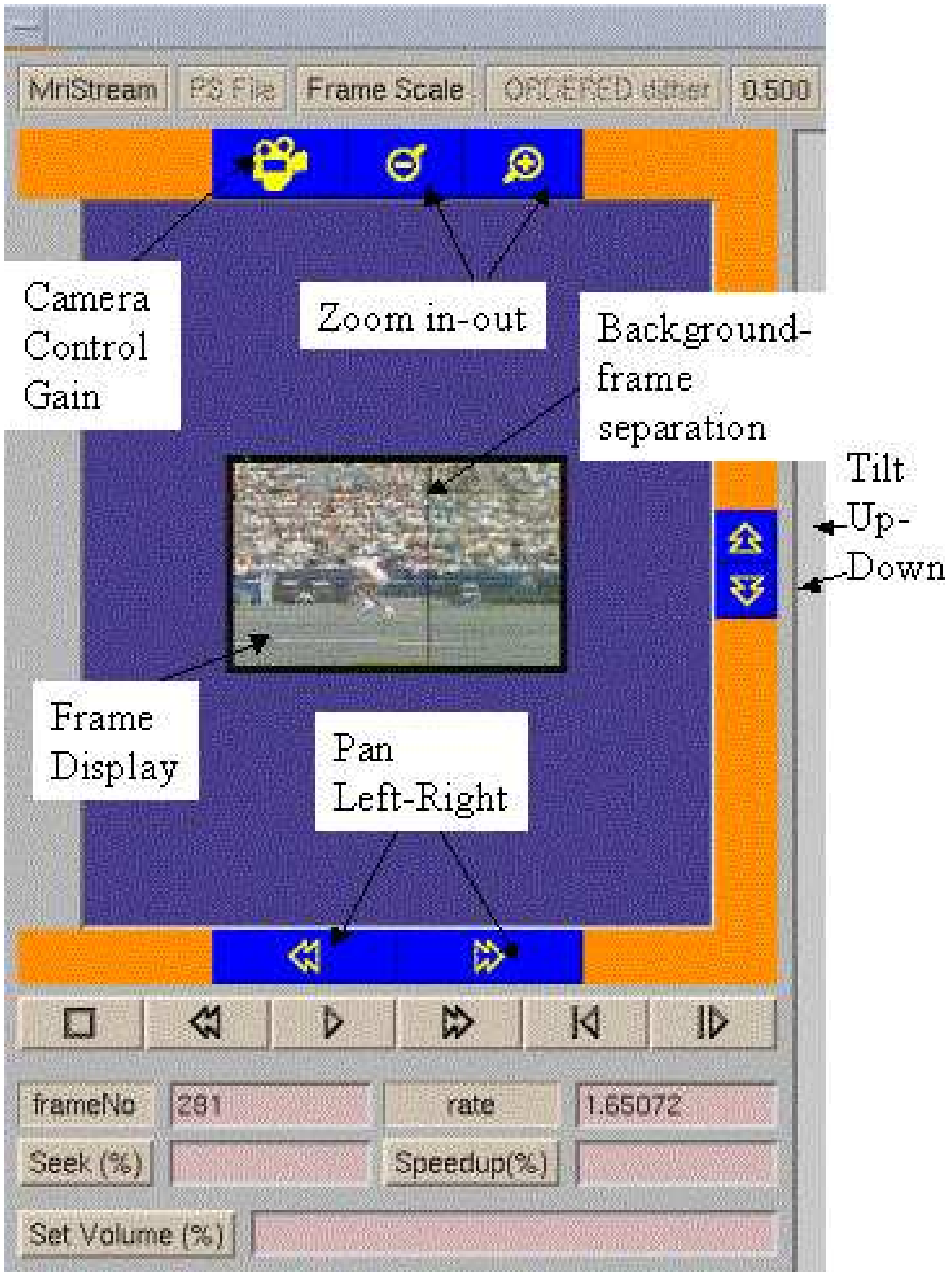


Figure 6.2.1: User interface for spatial browsing.

6.2.1 Camera Stabilization

When the user gains the camera control, the camera is stabilized. In this case, since the camera is static, the object might enter and leave the scene if the original video is tracking the object.

Let f_c frame be the frame where the user gained the camera control. Since all the motion parameters are kept according to the initial frame in the video, the relative motion with respect to the f_c has to be calculated. Let $M^m(t)$ be the motion parameter in frame f_t with respect to frame f_m. For affine motion in Cartesian coordinate system, the parameters for frame f_t are calculated using

$$M^c(t) = (M^0(c))^{-1} \times M^0(t) \tag{6.2.1}$$

If each frame f_t is warped according to the new relative motion parameters, the camera is stabilized.

6.2.2 Camera Operations

The user can perform pan left and right, tilt up and down, and zoom in and out. These operations are performed in the transformed coordinates. We have 3 parameters to handle panning, tilting and zooming: $panLeftRight$, $tiltUpDown$, and $zoomInOut$. We also have 3 thresholds to determine the strength of the interaction: τ_p, τ_t, and τ_z. Whenever the user clicks pan left button, $panLeftRight \leftarrow panLeftRight + \tau_p$. If the user clicks pan right button, $panLeftRight \leftarrow panLeftRight + \tau_p$. If the user clicks pan tilt up button, $tiltUpDown \leftarrow tiltUpDown - \tau_t$. If the user clicks zoom in button, $zoomInOut \leftarrow zoomInOut/\tau_z$. If the user clicks zoom out button, $zoomInOut \leftarrow zoomInOut * \tau_z$.

The environment should be set according to the new user settings. The background and the foreground should be updated. Panning and tilting require modification on the translational parameters. The transformation matrix becomes

$$\begin{bmatrix} a_2^0(c) * zoomInOut & a_3^0(c) * zoomInOut & a_0^0(c) - panLeftRight \\ a_4^0(c) * zoomInOut & a_5^0(c) * zoomInOut & a_1^0(c) - tiltUpDown \\ 0 & 0 & 1 \end{bmatrix} \tag{6.2.2}$$

6.2.3 Data for Spatial Browsing

The motion parameters must be available for the spatial browsing. In MPEG-4 sprite coding, the coordinates of trajectories are maintained for generation of motion parameters. If perspective, affine, translation-zoom-isotropic rotation, and translational motion are used 4 points, 3 points, 2 points, and 1 point are required to generate the motion parameters, respectively. To support real time processing, the number of processes needs to be decreased. Instead of using coordinates, the original motion parameters are maintained.

The upcoming frames are warped according to the first frame in the sequence. For each frame, the motion parameters are estimated and stored in the database. After warping each frame, the center of the frame in the sprite frame is also maintained.

Since the frames are warped onto the original frame, the relative motion parameters are maintained instead of the motion parameters between the sequential frames. The motion transformation can be written briefly as:

$$v' = Mv \tag{6.2.3}$$

where M contains the motion parameters for the first matrix; v is the coordinate in the previous frame; and v' is the transformed coordinate. The relative motion is computed as

$$v'' = M'v' = M'(Mv) = (M'M)v \tag{6.2.4}$$

where v'' is the vector for the new transformed coordinates; M' and t' hold the current motion parameters; and M and t hold the motion parameters up to the current frame.

For each frame f_i, we have the feature set $F_i = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, center_x, center_y\}$ where a_i is the motion parameters and $center_x$ and $center_y$ are the center coordinates of the frame in the sprite.

6.3 Experiments

We have tested our tool on MPEG test sequences ’stefan’, ’foreman’, ’coastguard’ and lecture examples. For stefan, we also have the segmentation mask and used it in the sprite generation of ’stefan’ sequence. The affine motion model is used in the motion estimation of ’foreman’, ’stefan’ and lectures. Translational motion model is used in the estimation of ’coastguard’. We have used these motion models to be comparable with other work in the literature.

6.3.1 Accuracy of Sprite

The common method to measure the quality of a sprite is to regenerate the frames from the sprite and then to compute the PSNR (Peak Signal-to-Noise Ratio) between the generated frames and the original frames. Higher PSNR mostly indicates higher quality. However, PSNR may not always be a good measurement. Usually PSNR is used to measure the quality of a distorted image by using the original image. Video can be considered as a series of images. The quality of a generated sprite in this case is measured by the average PSNR. The average PSNR obfuscates some problems in the sprite generation. Humans can also evaluate the generated sprite. Sometimes, the sprite is generated and the quality is measured by the viewer.

PSNR is based on the error function between images (more specifically Equation 3). The sprite is generated based on the motion vectors. There are two factors that affect error in the function. Firstly, during sprite generation, pixels may be shifted in a wrong way. Secondly, the sprite is blurred due to warping and temporal integration. Shifting problem lowers PSNR whereas blurring may increase PSNR. When the image is blurred, all the pixels at a specific location in all frames will be closer to the mean value. If one of them was selected, the image would not be blurred, but the error would increase. We conclude that PSNR alone is not a good measure of the quality of a sprite. Another factor is the blurring of the image or in contrast, how *sharp* the image is. PSNR with *sharpness* of the sprite for each frame is a better indication of the sprite quality.

In real life, blurring is usually caused by the distortion in the lens of a camera. We will compute the sharpness of the image using the gradient of the edges. Let $\bigtriangledown g_{ij}^{t}$ be the gradient at location (i, j) of frame f_t. The gradients are truncated if they are less than the edge threshold τ_e ($g^t_{ij} = 0\ if\ \bigtriangledown g^t_{ij} < \tau_e$). Let f_t be the frame that sharpness will be computed and f_o be the original frame. The sharpness of a frame f_t is determined by

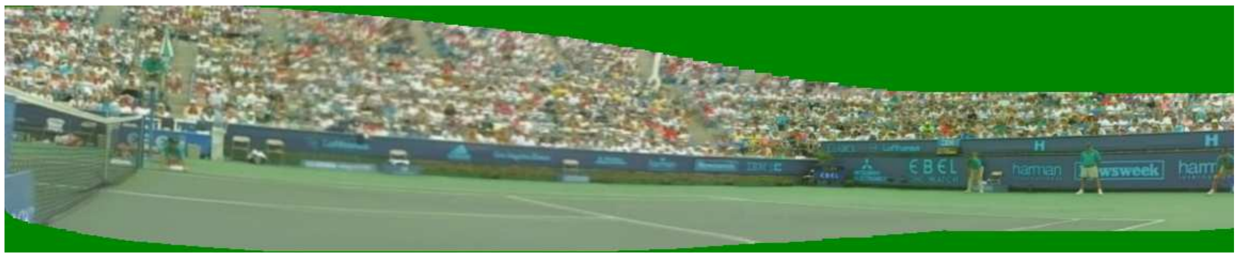
$$\gamma_t = \frac{\sum_i^N \sum_j^M \bigtriangledown g^t_{ij}}{\sum_i^N \sum_j^M \bigtriangledown g^o_{ij}}. \tag{6.3.1}$$

This equation measures how much the original edges are smoothed. The number of edges could also be used in sharpness computation; however, the gradient also keeps the information on how much the edge is smoothed.

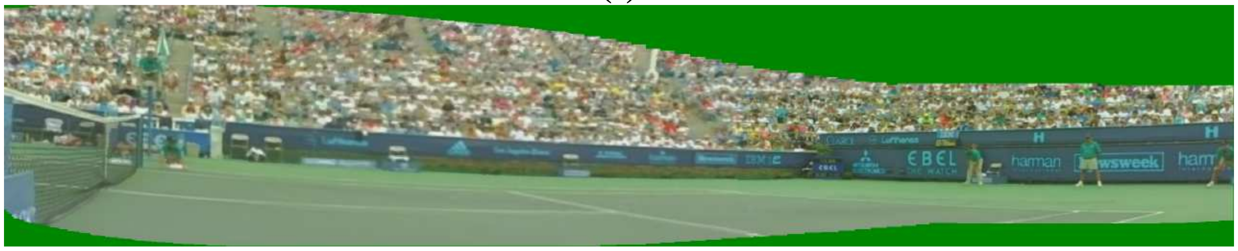
During our experiments we have computed motion vectors, sharpness, and PSNR for each frame. We have generated conservative sprite and stationary sprite for different sprite details 1 and 2, histogram sizes 1, 2, 4, 8, and 16 for the histemporal filter. Due to space limit, we will explain our results on stefan sequence in detail and give summary of the other results.

6.3.2 PSNR

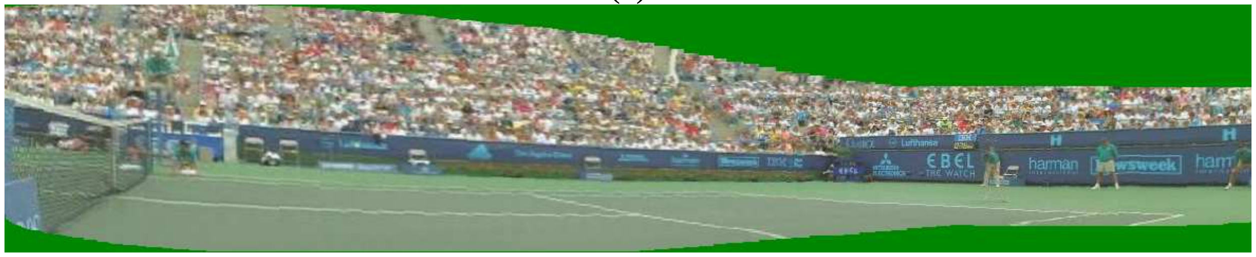
Different sprites generated for the 'stefan' sequence is shown in Figure 6.3.1. We used the segmentation mask for generation of the mask. Without segmentation mask, we also get a similar sprite since the foreground object displaces its location frequently. The average PSNRs values are listed in Table 6.3.1. The PSNRs for histogram sizes 1, 2, and 4 are



(a)



(b)



(c)

Figure 6.3.1: Sprites for ’stefan’ (a) histogram size=1 and detail=1 (b) histogram size=2 and detail=4 (c) conservative sprite.

shown in Figure 6.3.2. When the histogram size increases, there is slight improvement on the PSNR. When the detail is increased, there is a significant increase in PSNR. Figure 6.3.3 displays the PSNR results for detail=2.

The sprite can be created conservatively. Figure 6.3.4 displays the PSNR values for the conservative sprite.

For the foreman sequence the sprite cannot be extracted since the background is covered

detail	1	1	1	2	2	2
histogram size	1	2	4	1	2	4
PSNR	21.16	21.52	21.54	22.26	22.55	22.66

Table 6.3.1: Average PSNR values for ’stefan’

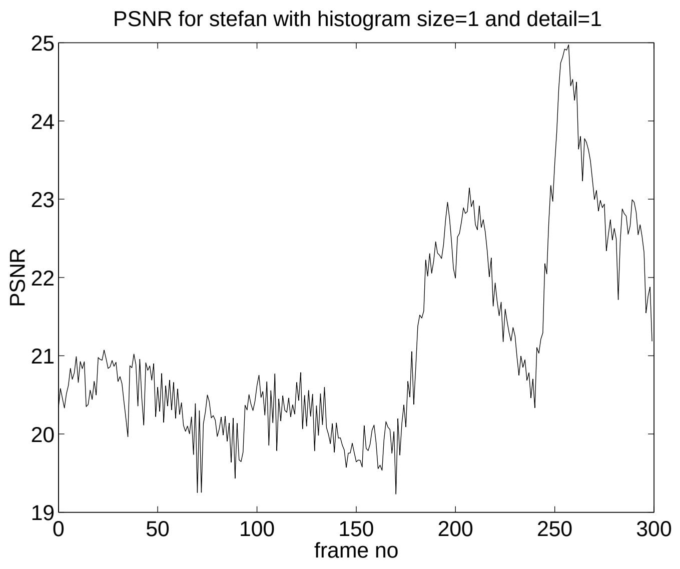
detail	1	1	2	2
histogram size	1	16	1	16
PSNR	24.38	26.37	25.20	27.44

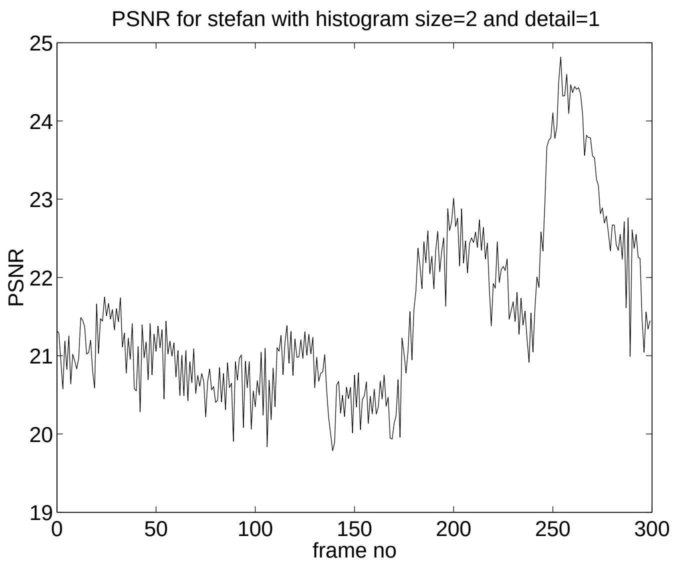
Table 6.3.2: Average PSNR values for ’foreman’

with the object. However, after frame 190 the object disappears from the scene and sprite can be generated from frames 190 to 300 (Figure 6.3.5). Figure 6.3.6 depicts the PSNR values. The average PSNR values are given in Table 6.3.2. For ’foreman’ sequence, the increase caused by histogram size is more significant than detail. When histogram size becomes 16, the PSNR approximately increase by 2. On the other hand, when detail becomes 2, the PSNR increases by 1.

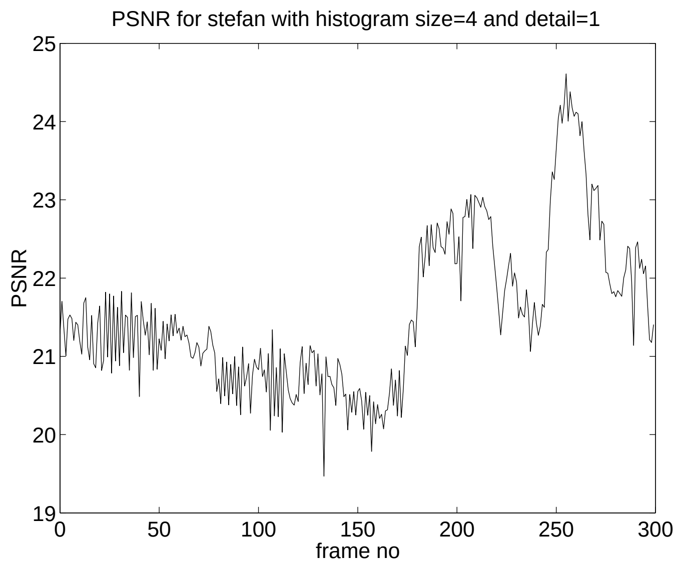
6.3.3 Sharpness

To measure the sharpness of an image, the edges in the original and the generated frame have to be extracted. The edges are detected using the Sobel operator [37]. The gradient $\bigtriangledown g$ is obtained by adding horizontal gradient $\bigtriangledown g_x$ and vertical gradient $\bigtriangledown g_y$. The edge



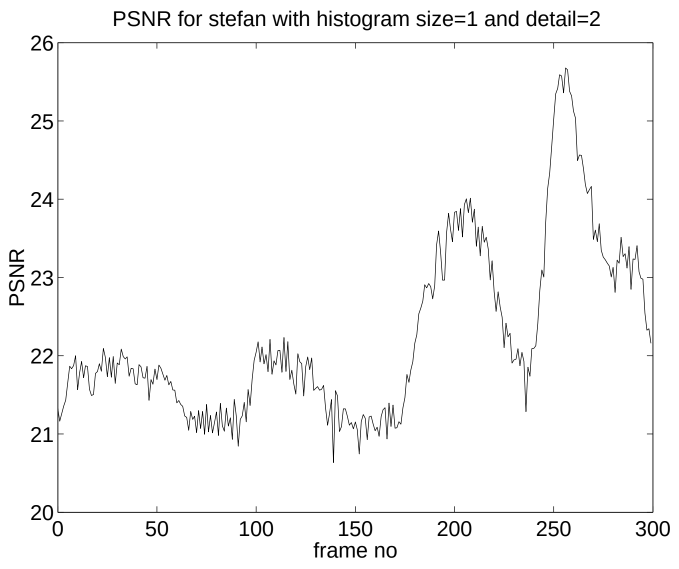


(a) (b)

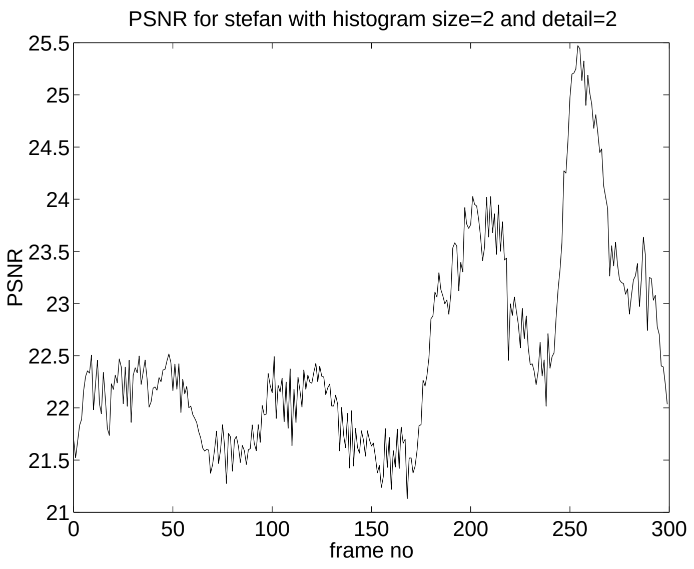


(c)

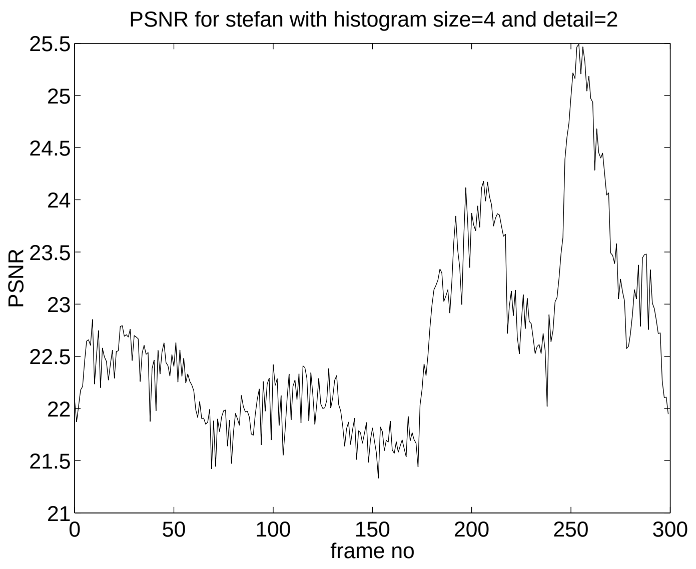
Figure 6.3.2: PSNR for stefan for detail=1 (a) histogram size=1 (b) histogram size=2 (c) histogram size=4.



(a)



(b)



(c)

Figure 6.3.3: PSNR for stefan detail=2 (a) histogram size=1 (b) histogram size=2 (c) histogram size=4.

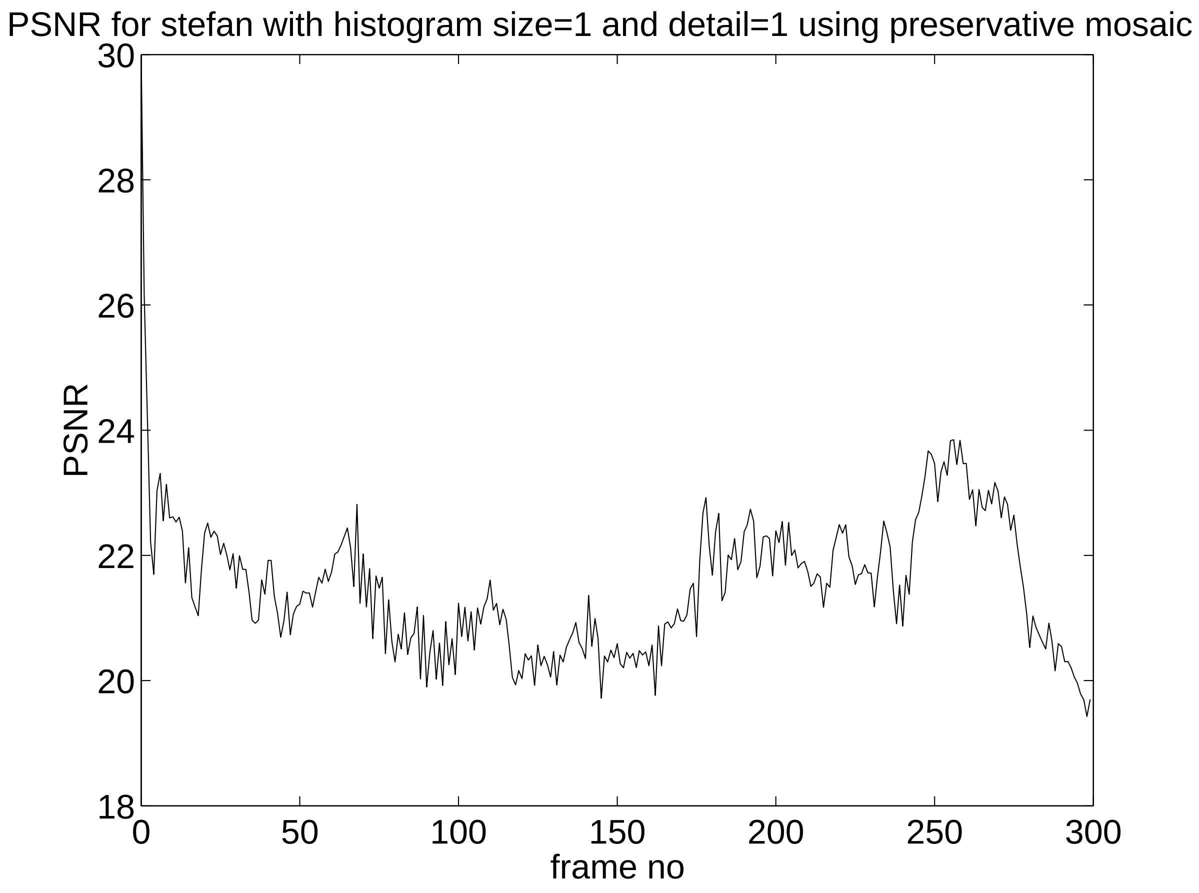


Figure 6.3.4: PSNR for stefan using conservative sprite.

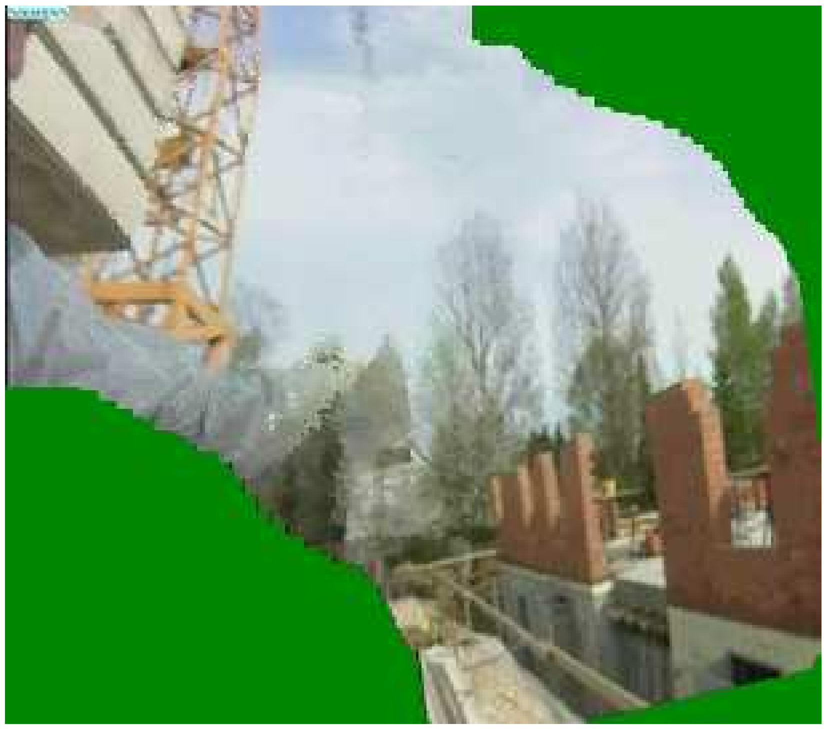
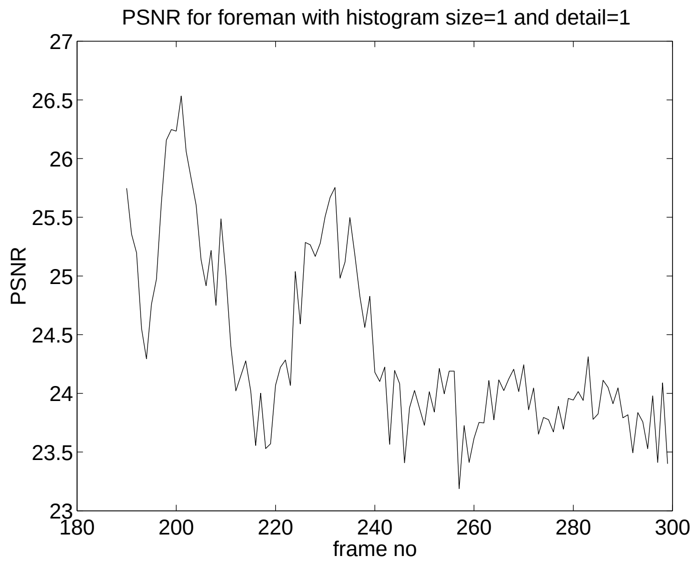
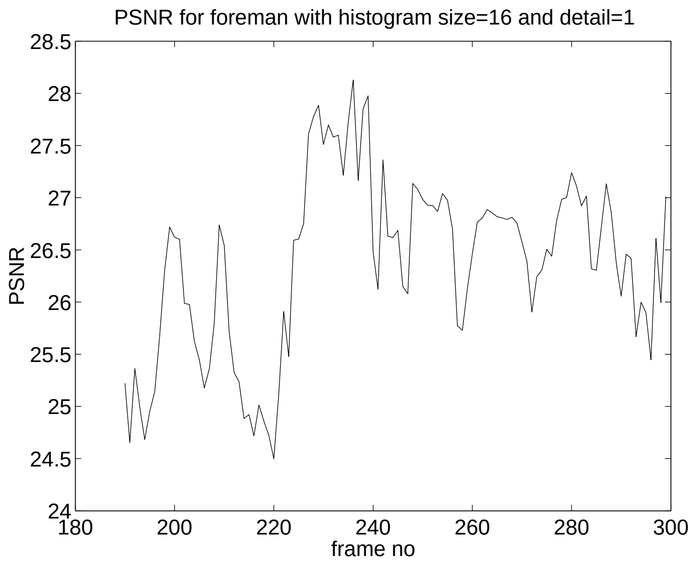


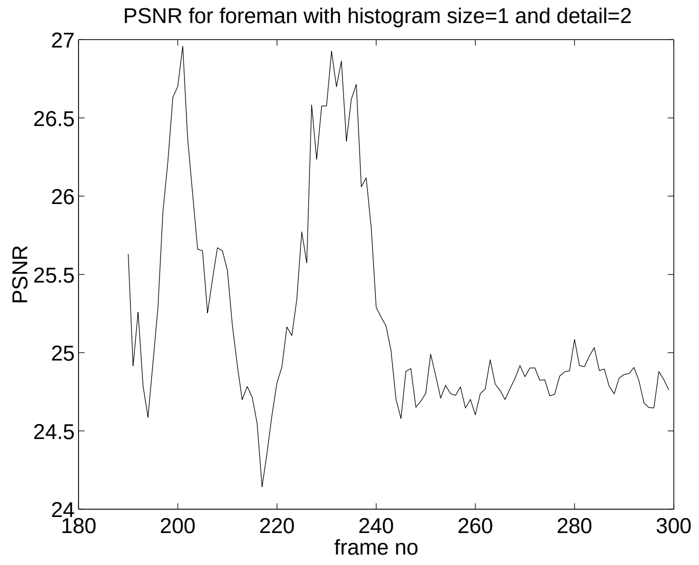
Figure 6.3.5: Sprite for ’foreman’ video from frames 190 to 300.



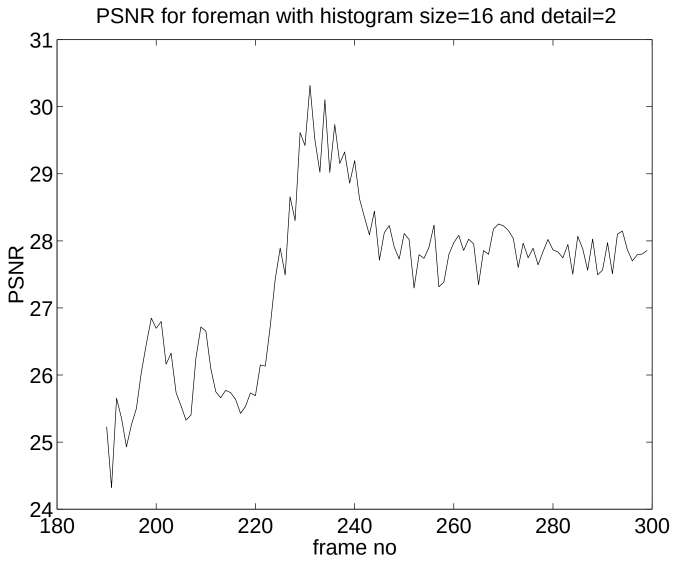
(a)



(b)



(c)



(d)

Figure 6.3.6: PSNR for foreman sequence from frame 190 to 300 (a) detail=1 histogram size=1 (b) detail=1 histogram size=16 (c) detail=2 histogram size=1 (d) detail=2 histogram size=16.

detail	1	1	1	2	2	2
histogram size	1	2	4	1	2	4
Sharpness	0.45	0.50	0.55	0.52	0.55	0.59

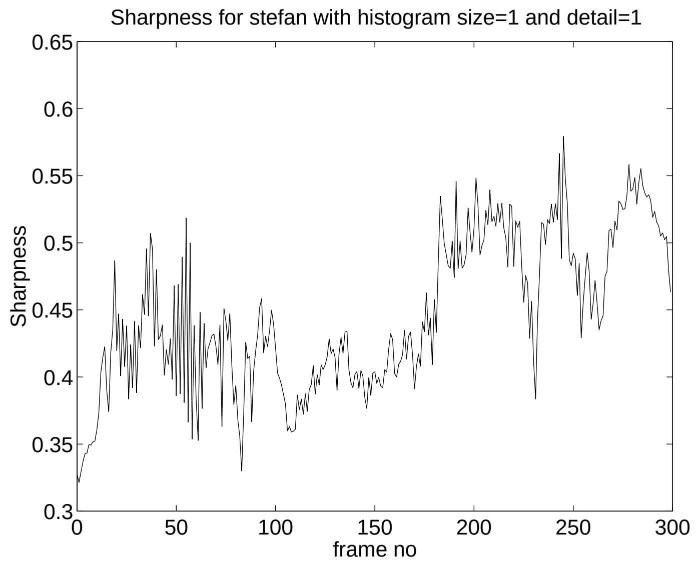
Table 6.3.3: Average sharpness values for 'stefan'

threshold τ_e should not be so high. If τ_e is high, only blurring at significant edges are detected. In our experiments, we set τ as 30. Figure 6.3.7 displays the sharpness of generated frames of 'stefan'. The average sharpness is given in Table 6.3.3. Figure 6.3.7 shows the sharpness values for each frame in each case. The average sharpness for conservative sprite is 0.83. As can be seen from Table 6.3.3, both detail and histogram size are effective in increasing sharpness.

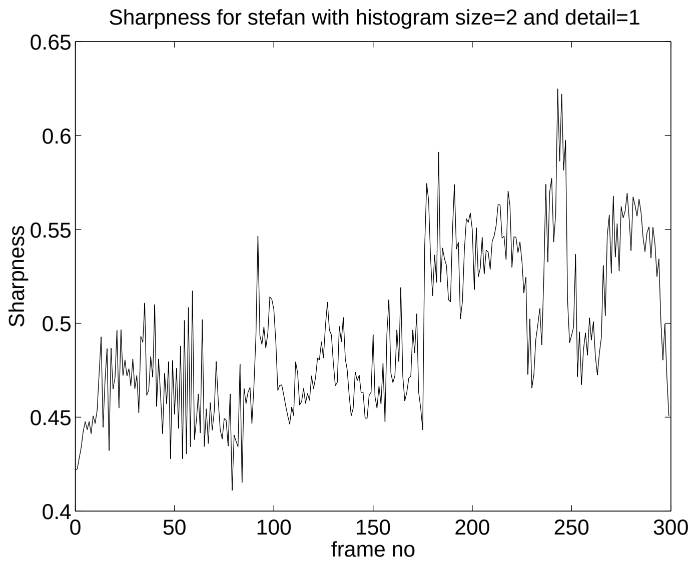
6.3.4 Discussion

According to Table 6.3.1, the histogram size did not have significant contribution to PSNR. However, according to Table 6.3.3, the histogram size has clear positive effect on the sharpness of the sprite. Although histogram size may not contribute much to PSNR, it contributes to the sharpness of the image. This positive improvement is not detectable by PSNR.

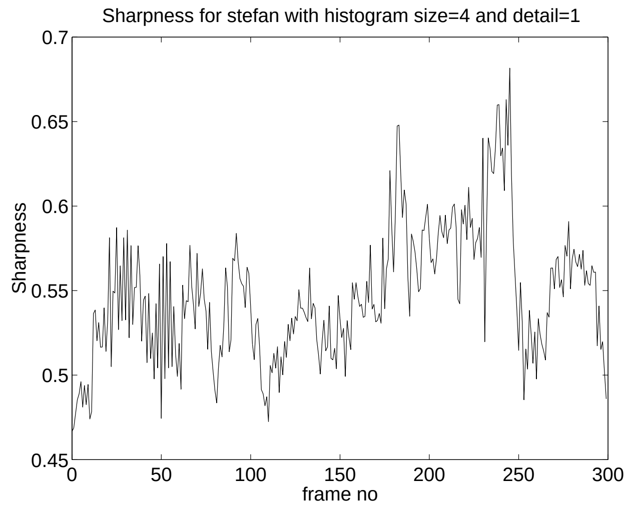
There is a strict relationship between the PSNR and the motion of the camera. Whenever the object visits a new place, the PSNR starts increasing. Whenever the object visits previously visited place, PSNR starts decreasing. For instance for frames 0 to 30, the object is



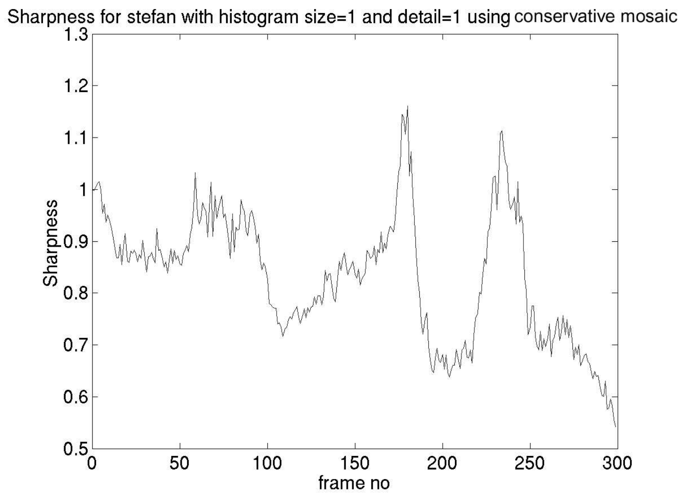
(a)



(b)

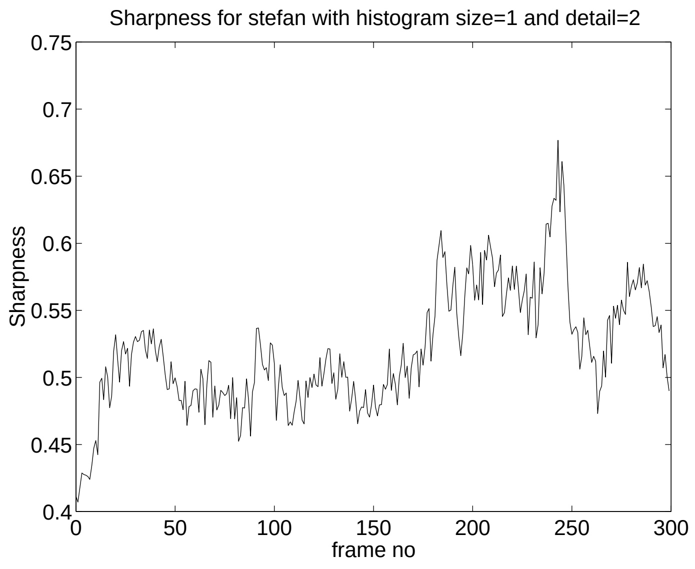


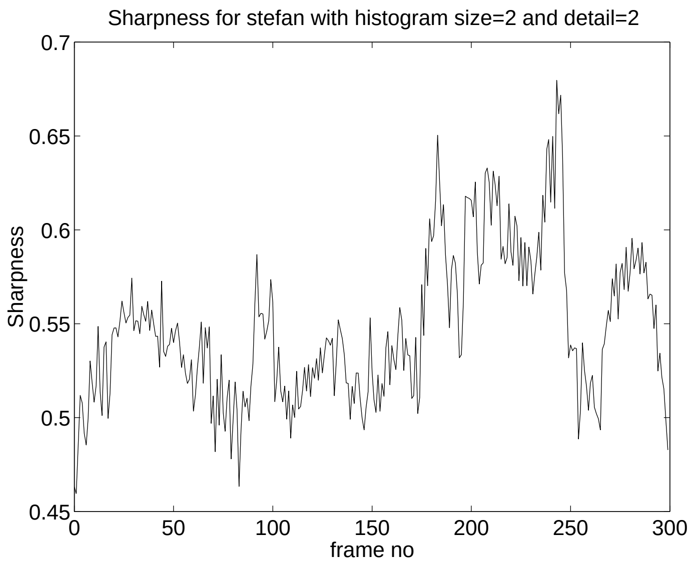
(c)



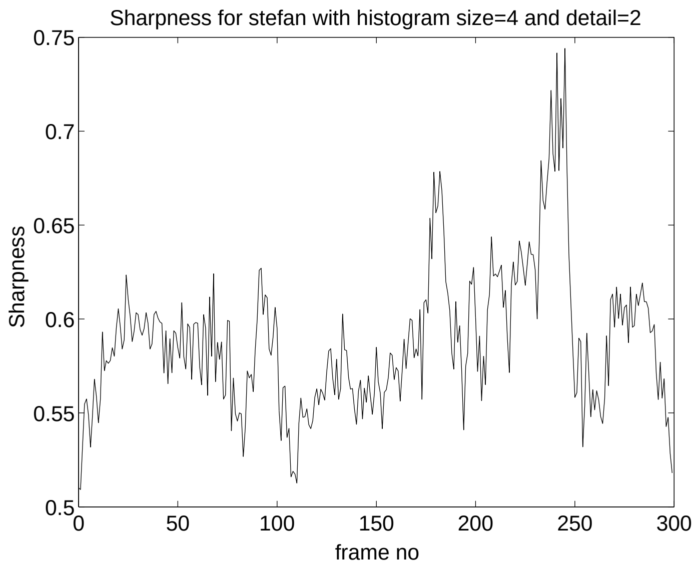
(d)

Figure 6.3.7: Sharpness for stefan sequence with detail=1 (a) histogram size=1 (b) histogram size=2 (c) histogram size=4 (d) conservative sprite.





(a) (b)



(c)

Figure 6.3.8: Sharpness for stefan sequence with detail=2 (a) histogram size=1 (b) histogram size=2 (c) histogram size=4.

visiting new places (Figure 6.3.9). From frames 30 to 80, again previously visited palaces are visited. From frames 80 to 120, new places are visited. From frames 120 to 180, old places are visited. From frames, 180 to 210 new places are visited. From frames 210 to 250 new places are visited. From frames 250 to 290, new places are visited. We did not consider the position in y direction since the motion in y direction is not significant. However, the increase and decrease in PSNR are closely related with visiting the new or old places. In fact, this is related with the number of visits to a specific position. As a future work, to increase the PSNR, the frames that visit previously visited places may not be used in the sprite generation.

6.4 Limitations

The moving objects in the video have to be captured completely. If a video object does not appear completely in a video frame, during spatial browsing the invisible parts will still be invisible. There are a couple of ways to avoid this defect. One of them is to limit spatial browsing when the video object is not captured completely.

We assume that the camera does not make significant rotational motion around its axis. In such a case, the generated sprite should be mapped to a cylindrical sprite. The approach can also be applied for Video $360'$ applications. In those cases, the sprite has to be static.

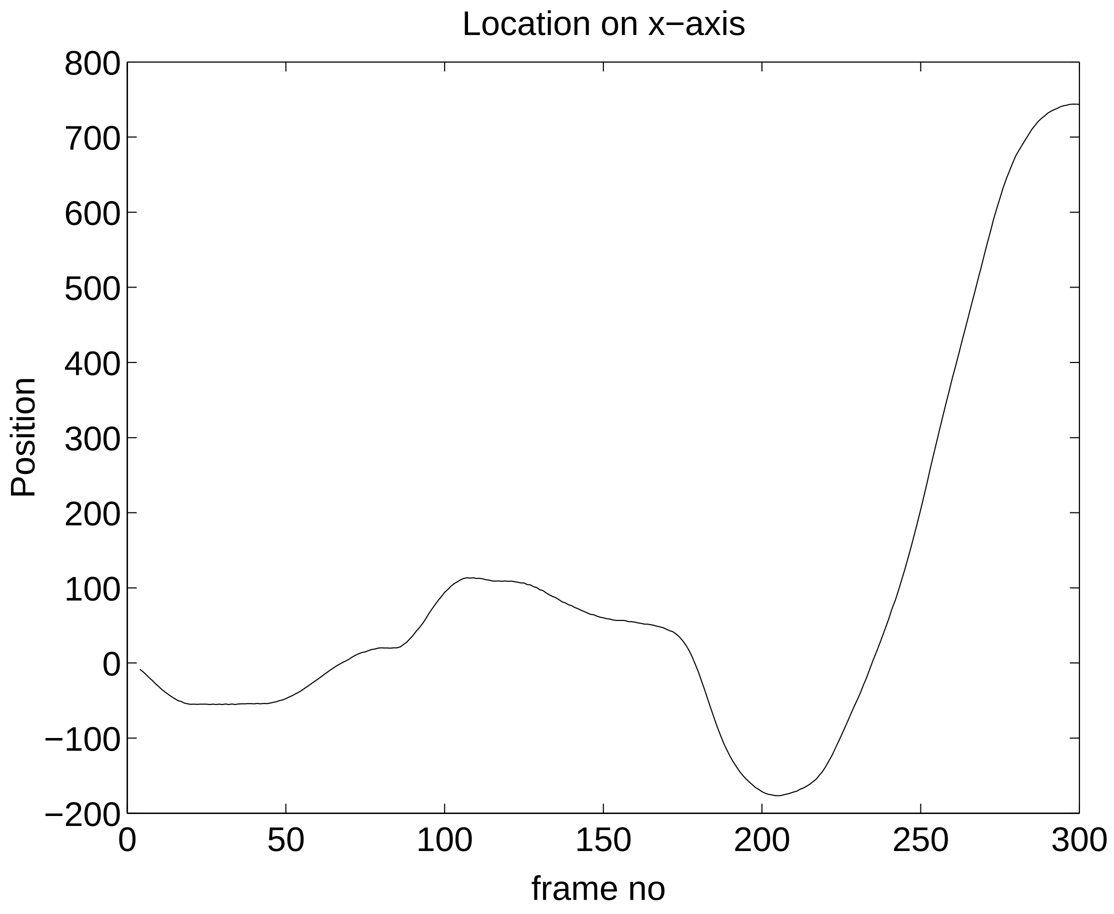


Figure 6.3.9: Position of object in x axis

6.5 Summary

In this chapter, we have presented an interactive spatial browser, VideoCruise, for recorded digital video. The VideoCruise requires high quality sprite generation. Our experiments have showed that average PSNR is not always a good indicator of quality by itself. PSNR does not consider blurring in the sprite. Sharpness measure is an indicator of blurring in the sprite. We have obtained different sharpness measures for the same average PSNR. We demonstrated examples from standard MPEG test sequences. Once the motion vectors and the sprite are generated, VideoCruise provides interactive spatial browsing. VideoCruise provides panning, tilting and zooming interactions. VideoCruise allows the use to gain and release the camera control at any frame display. When the camera control is gained all, frames are mapped according to the frame which camera control is gained. In addition to browsing, it enables camera stabilization.

Chapter 7

An Introduction to Multimedia Synchronization

Multimedia synchronization deals with the synchronization of media streams in a presentation. Synchronization is classified as *intra-stream* synchronization and *inter-stream* synchronization. The intra-stream synchronization manages the presentations of streams at a required rate (e.g. playing video 30frames/second). The inter-stream synchronization manages the relationships among the streams. There are two types of inter-stream synchronization: *fine-grained* synchronization and *coarse-grained* synchronization. Fine-grained synchronization requires a tight synchronization between each segment of two streams like a lip-synchronization between audio and video. Most of the research in fine-grained synchronization aims at lip-synchronization between audio and video [92]. Coarse-grained synchronization handles the relationships among streams and determines when streams start and end. Synchronization specification languages like SMIL [88] focuses on the synchronization requirements for coarse-grained synchronization.

Composition of media objects such as audio, video, and image play an important role in today's multimedia systems and databases. A multimedia presentation model should support both event-based and time-based actions to satisfy flexible specification and presentation requirements. The complex synchronization requirements should be supported by the model to maintain the integrity of the synchronization among individual media streams in case of delay or loss of data over the network. The model should allow VCR-type functions like play, pause, resume, (fast-slow) forward and backward, skip and these user interactions should be provided for browsing capabilities of the presentation and should not increase the complexity of the specification of the multimedia presentation. The backward operation provides browsing and the skip functionality allows presenting specific segments of the presentations. Forward and backward (fast or slow) operations enable quick scan of the presentation. For example, in an education multimedia system, the user may not understand the concepts that are stated in a lecture video (and audio) and he may need to replay that section until he understands. So, a presentation may need to be started from any point.

If the relationships among media streams are specified by using constraints, condition statements, or event relationships, the synchronization can be handled in a more consistent way. In event-based models, no modification on start times and durations of streams is needed for interactions such as play, pause, resume and (fast-slow) forward. But skip and backward operations are complex, because the constraints, condition statements and events for these functionalities must also be available to the system. If the backward presentation is supported, the length of the specification doubles. So, event-based models do not support backward presentations. Some support skip operations with a lengthy specification. The event-based models have more semantics than time-based models by imposing relationships among streams. When an event is signaled, it means that there is a relationship between the source and the destination. Time-based models do not have this kind of semantics. In an event-based model, the start of a stream depends on an event signal. So, it is unnecessary to keep parameters such as start time and duration, and to modify these values. Thus, using events give us more flexibility to model the synchronization of a presentation. Previous work on event-based models assumes that the events will certainly be generated. So, they did not need to investigate when events can be signaled.

Nevertheless, to our knowledge, there is no implemented flexible model that supports backward functionality. The main difficulty behind backward and skip is that these interactions change the course of the presentation. Although conceptual models have been proposed to handle backward and skip [107, 73], the implemented systems could not use these models since they require the authors to spend enormous time on modeling.

7.1 Reasons for Limitations on User Interactions

More generally, we believe that these implemented systems ignored skip and backward due to following reasons:

1. Most of the presentations included compressed video stream like MPEG. Compressed video is designed for forward display and important frames appear first in the nominal presentations. But they appear the last in the backward direction thus making the backwarding very difficult. If video cannot be played in backward direction, there is no need to backward a presentation. Skipping is not easy since each frame in the video has a different size. It is not possible to skip to a specific frame since its exact location in the video is unknown. Nowadays, by using buffering and preprocessing techniques it is possible to perform backwarding and skipping to any point in the video [108].

2. Most of the presentations also included audio. Playing audio in the backward direction does not make any sense. Nowadays, audio is accompanied with closed caption. Even though audio is not played, the text can be displayed in the backward direction.

3. Inherent deficiency of the models could not deal with these operations. Especially,

it is hard to manage these user interactions in event-based models since some events may be skipped or signaled again yielding incorrect presentations. For the backward presentations, it is even unknown what to perform in the backward direction.

4. It is assumed that management of these functionalities is easy and they can be incorporated into the system easily later. Management of these functionalities looks easy but is actually hard. At first sight, it looks like reversing the relationships yields the backward presentation, which is not true. There has been a lot of time spent for proper specification of forward presentations to capture the synchronization requirements. Since, presentations are not delivered in a perfect world, each minor difference in the specification may yield a different presentation. From a given forward presentation specification, a number of backward presentations may be specified. The problem is to determine the best one, which is compliant with the author's forward presentation.

7.2 Related Work

Allen [1] introduced 13 primitive temporal relationships for time intervals. This model describes how the time instants and presentation durations of two temporal intervals are related. It does not quantify the temporal parameters for time instants and duration of temporal intervals. This work forms the basis for most of the synchronization models. Little et. al. [56] extended these with n-ary and reverse relationships. The n-ary model has the same power as the Allen's model but it is easier to create composite objects. The reverse relationships can declare relationships for backward presentations. *Overlaps, during,* and *finishes* can only be expressed with time values in these models. The verification of a multimedia presentation usually consists of verification of these relationships. If there is a problem in presenting any media object, it is not clear how the presentation of other media objects will be affected if the temporal relations are defined by using *before, overlaps, during,* and *finishes*. This kind of temporal compositions of media objects is good on local systems where distribution over network is not needed.

7.2.1 Time-based Models

One important factor in modeling of presentations is whether the model is time-based or event-based. A lot of work has been done on time-based modeling. Timed Petri Nets are first introduced for multimedia presentations in OCPN [55] and extended with user interactions in [107]. The modeling of user interactions using Petri Nets has been covered in [73]. The backward and skip has also been covered but for each possible skip and backward operation (including the current position of presentation and where skip is performed), a Petri Net has to be constructed. As these Petri Nets are not connected to each other, verification and consistency of the whole system is difficult to handle. So, a model where all operations can be incorporated and verified is needed. Gibbs [36] proposed a way of composing objects through BLOB stored in the databases and created objects by applying interpretation, derivation and temporal composition to a BLOB. The temporal composition is based on the start times of media objects on a timeline. Hamakawa et al. [40] has an object composition and a playback model where the constraints can be defined only as pair-wise. A time-based synchronization model that considers master-slave streams having at least one master stream is explained in [46]. NSync [12] is a toolkit which manages synchronous and asynchronous interactions, and fine-grained synchronization. The synchronization requirements are specified by synchronization expressions having syntax *When* {*expression*} {*action*}. The synchronization *expression* semantically corresponds to "whenever the expression **becomes** true, invoke the corresponding *action*". It does not allow backward presentations and declaration is harder than typical declarations since the author has to specify the necessary information for possible skip operations. The specification with NSync is complex, since the user may need to update variables such as pointers of the presentation that causes the user to think about pointer updates so that the presentation is consistent and the user may need to specify more conditions due to skip functionality.

Time-based models usually keep the start time and duration of each stream and these models modify the duration and start time after each interaction of the system or the user.

7.2.2 Event-based Models

It has been shown that event-based models are more robust and flexible for multimedia presentations. A disadvantage of the event-based models is the inapplicability of the model in case there is a change in the course of the presentation (like backwarding and skipping). In an event-based model, the start of a stream depends on an event signal. Events for multimedia applications are discussed in [99] and a model that includes temporal and spatial events is given in [68]. SSTS [68] is a combination of Firefly's [18] graph notation and transition rules of OCPN [55]. SSTS has AND-start, AND-end, OR-start and OR-end nodes to satisfy the synchronization requirements of multiple streams. The relationships among streams are based on binary relations. SSTS does not cover any user interactions. DAMSEL [75] has an event-based model that considers activation of two events such that "occurrence of an event will cause the occurrence of another event *t* time units later". The occurrence of an event may also defer the occurrence of another event. Temporal constraints that are used in Madeus [50] are based on Allen's temporal relations [1]. FLIPS [83] is an event-based model that has barriers and enablers to satisfy the synchronization requirements at the beginning and the end of the streams. It does not have complex user interactions such as fast-forward and fast-backward but it has a limited skip operation that moves to the beginning of another object. The functionalities of an application are classified as pre-orchestrated or event-based in [99]. The pre-orchestrated actions are the actions that are known prior

	Model		Synchronization Requirements		User Interactions	
	Time based	Event based	Satisfaction	Specifi cation Complexity	Play, Pause, Resume, Forward	Backward Skip
Gibbs [36]	√	–	Low	Low	–	–
FLIPS [83]	–	√	High	High	√	No backward Limited skip
NSync [12]	√	Synchronization expressions	High	High	√	No backward
PREMO [41]	√	√	High	High	No forward	–
Hamakawa [40]	√	constraints	Moderate	Moderate	–	–
SMIL [88]	–	–	High	Moderate	–	–
RuleSync	√	√	High	Moderate	√	√

Table 7.2.1: Comparison of existing methods.

to the presentation whereas the event-based ones are triggered by events. A timeline approach with event-based modeling is proposed in [43]. User interactions are considered but not VCR functions such as fast-forward, fast-backward, or skip. They emphasize the synchronous and asynchronous events. PREMO [41] presents an event-based model that also manages time. It has synchronization points that may also be AND synchronization points to relate several events. Time for media is managed with clock objects and time synchronizable objects that contain a timer. They do not discuss any user interactions in their model. Multimedia synchronization using ECA rules is covered in [5, 4, 9]. These papers show how synchronization requirements can be modeled using ECA rules and form the basis of ECA rule modeling in this paper. A hierarchical synchronization model that has events and constraints is given in [24].

Table 7.2.1 gives the comparison of the some of the existing synchronization methods.

7.2.3 Synchronization Languages

SMIL [88] is a mark-up language for publishing synchronized multimedia presentations via the Internet. It is unclear how the user's input affects the presentation. Marcus and Subrahmanian [59] consider presentation creation depending on the consecutive queries and constraints submitted by the user. The user specifies how the query results will be presented.

Chapter 8

RuleSync: Robust Flexible Synchronization Model Using Synchronization Rules

In this chapter, we introduce a synchronization model, termed *RuleSync*, for management of robust, consistent, and flexible presentations that include a comprehensive set of interactions. RuleSync manipulates synchronization rules that are managed by Receiver-Controller-Actor (RCA) scheme where receivers, controllers and actors are objects to receive events, to check conditions and to execute actions, respectively.

8.1 Introduction

Multimedia presentation management has drawn great attention in the last decade due to new emerging applications like video teleconferencing, collaborative engineering, asynchronous learning and video-on-demand. Applications like video teleconferencing are live presentations and user interactions are usually limited in terms of accessing the presentation. On the other hand, applications like asynchronous learning and collaborative engineering may exploit recorded presentations and users may later access and interact with these multimedia presentations. There have been challenging problems confronted when multimedia presentations enable user interactions and are transmitted over networks shared by many users. The loss and delay of the data over the networks require a comprehensive specification of the synchronization requirements. The user interactions that change the course of a presentation either increase the complexity of the specification or are not allowed.

Multimedia presentation management research started with organization of streams that participate in the presentation and VCR-based user interactions are incorporated at different levels at the later research. Initial models only considered simple interactions like play, pause, and resume. Flexible models do not enforce timing constraints and temporal organization is rather performed by relating events in the presentation. For example, *stream A starts after (meets) stream B.* There is no enforcement on media clock time like *stream B* has to end at an instant and at that instant *stream A* has to start. Since there may be delay in the play of *stream B*, to start *stream A* after *stream B* brings flexibility by not enforcing timing constraints. Speed-up and slow-down operations are included at a later stage in initial models. Skip and backward interactions are able to be incorporated in time-based models. Flexible models could not incorporate these functionalities since it is not clear how these interactions affect the presentation in these models. There are only few flexible models considering skip operation. These models have some restrictions on the application of skip functionality.

The following section explains the synchronization rules by introducing events, conditions, and actions for multimedia presentations. Section 8.3 presents the components of the synchronization model where middle-tier, receivers, controllers, actors, and timeline are covered. The last section summarizes the chapter.

8.2 Synchronization Rules

Synchronization rules form the basis of the management of relationships among the synchronization rules. Each synchronization rule is based on the Event-Condition-Action (ECA) rule.

Definition 8.2.1 *A synchronization rule is composed of an event expression, condition expression and action expression, which can be formulated as:*

on *event expression* ***if*** *condition expression* **do** *action expression.*

A synchronization rule can be read as: When the *event expression* is satisfied if the *condition expression* is valid, then the actions in the *action expression* are executed.

8.2.1 Events, Conditions and Actions for a Presentation

In a multimedia system, the events may be triggered by a media stream, user, or the system. Each media stream is associated with events along with its data and it knows when to signal events. When events are received, the corresponding conditions are checked. If a condition is satisfied, the corresponding actions are executed.

Definition 8.2.2 *An event expression manages the relationships among the events and can be defined in the following format:*

```
eventExpression =       eventExpression && eventExpression
                    |   eventExpression || eventExpression
                    |   ( eventExpression )
                    |   event;
```

Definition 8.2.3 *An event is represented with* $source(event\_type[, event\_data])$ *where source points the source of the event, event_type represents the type of the event and event_data contains information about the event.*

The event expression enables the composition of events may be required to trigger actions instead of a single event. Composite events can be created by boolean operators && (AND) and || (OR). The AND composition requires all events in the composition and not necessarily at the same time.

The goal in inter-stream synchronization is to determine when to start and end streams. The start and end of streams depend on multimedia events. The hierarchy of multimedia events are depicted in Fig. 8.2.1. The user has to specify information related to the stream events. Allen [1] specifies 13 temporal relationships. Relationships *meets, starts* and *equals* require the *InitPoint* event for a stream. Relationships *finishes* and *equals* require the *EndPoint* event for a stream. Relationships *overlaps* and *during* require *realization* event to start (end) another stream in the mid of a stream. The relationships *before* and *after* require temporal events since the gap between two streams can only be determined by time. Temporal events may be *absolute* with respect to a specific point in a presentation (e.g. the beginning of a presentation). Temporal events may also be *relative* with respect to another event.

Event source can be the user or a stream. Optional event data contains information like a realization point. Event type indicates whether the event is *InitPoint*, *EndPoint* or *realization*

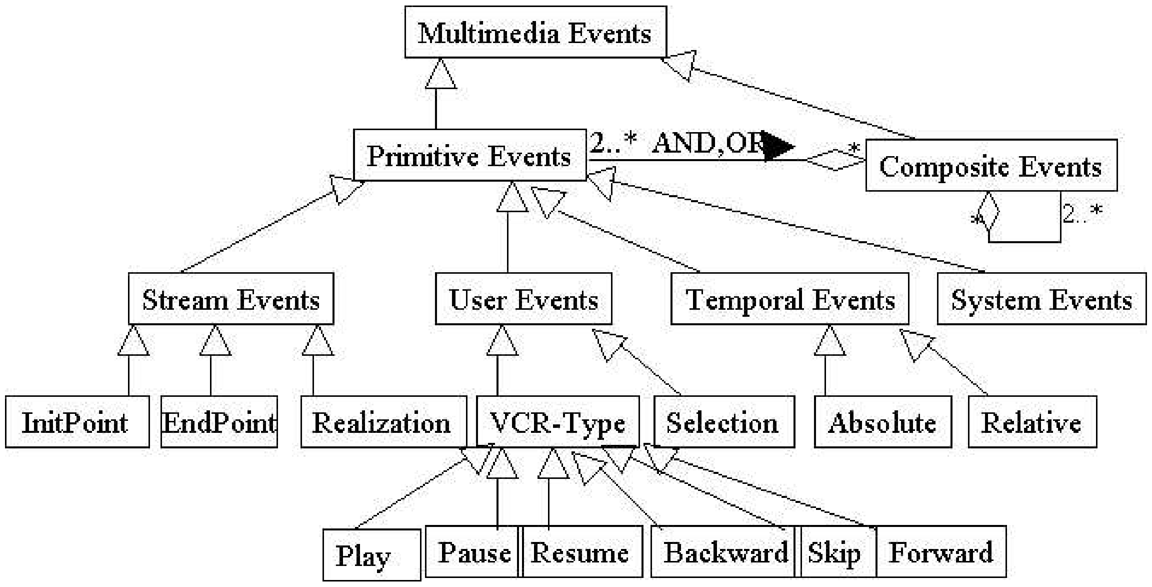


Figure 8.2.1: The event hierarchy.

if it is a stream event. Each stream has a series of events. Users can also cause events such as start, pause, resume, forward, backward and skip. These events have two kinds of effects on the presentation. Skip and backward change the course of the presentation. Others only affect the duration of the presentation. In our system, even a stream may be played multiple times, each instance of the stream has a different identifier. If there is no user interaction, each event is signaled once during the presentation. An event may be signaled multiple times if user interactions like backward and skip occur.

Definition 8.2.4 *The condition expression determines the set of conditions to be validated when the event expression is satisfied and has the following form:*

```
conditionExpression =      conditionExpression && conditionExpression
```

```
                    |   conditionExpression || conditionExpression
                    |    ( conditionExpression )
                    |   condition;
```

Definition 8.2.5 *A condition in a synchronization rule is a 3 tuple* $C = condition(t_1, \theta, t_2)$ *where* θ *is a relation from the set* $\{=, \neq, <, \leq, >, \geq\}$ *and* t_i *is either a state variable that determines the state of a stream or presentation or a constant.*

A condition indicates the status of the presentation and its media objects. The most important condition is whether the direction of the presentation is forward. The receipts of the events matter when the direction is forward or backward. Other types of conditions include the states of the media objects.

Definition 8.2.6 *The action expression is the list of the actions to be executed when condition is satisfied:*

```
actionExpression =     action | actionExpression;
```

Definition 8.2.7 *An action is represented with* $action\_type(stream[, action\_data], sleeping\_time)$ *where* $action\_type$ *needs to be executed for* $stream$ *using* $action\_data$ *as parameters after waiting for* $sleeping\_time$*. Action_data can be the parameter for speeding, skipping, etc.*

An action indicates what to execute when conditions are satisfied. *Starting* and *ending* a stream, and *displaying* or *hiding* images, slides and text are sample actions. For backward presentation, *backwarding* is used to backward and *backend* is used to end in the backward direction. There are two kinds of actions: *Immediate* Action and *Deferred* Action. *Immediate* action is an action that should be applied as soon as the conditions are satisfied. *Deferred* action is associated with some specific time. The deferred action can only start after this $sleeping\_time$ has been elapsed. If an action has started and had not finished yet, that action is considered as alive.

8.3 The Synchronization Model

The middle-tier is responsible for extraction of synchronization rules. In this section, firstly the role of the middle-tier is explained briefly. The elements of a multimedia presentation are explained along with SMIL expressions. The rule generation from SMIL expressions is covered with an example. Receivers, controllers, and actors are basic components of the synchronization model and responsible for management of synchronization rules. The presentation timeline is used to keep track of expected time of actions, receipts of events, satisfaction of controllers, and activation of actors. The presentation timeline is mainly used for user interactions that change the course of the presentation.

8.3.1 The Middle-Tier

The middle-tier for multimedia synchronization first handles the rules that can be extracted from the synchronization specification. Synchronization requirements are stored in rules since each synchronization rule is simple and can be processed easily to generate other rules. Once the rules from the specification are extracted, the synchronization rules for the backward presentation are generated. The extracted rules are fed into the synchronization model. The synchronization model contains a rule manager to manage these rules. The timeline for events and actions are generated in case the course of the presentation changes after user interactions like skip and change direction. When the presentation module receives an event from the user or one of the stream handlers, it informs the event and the current condition of the presentation to the synchronization model. The synchronization model determines if any of the rules are satisfied and if a rule is satisfied it informs the necessary actions to the presentation module. The framework is shown in Fig. 8.3.1.

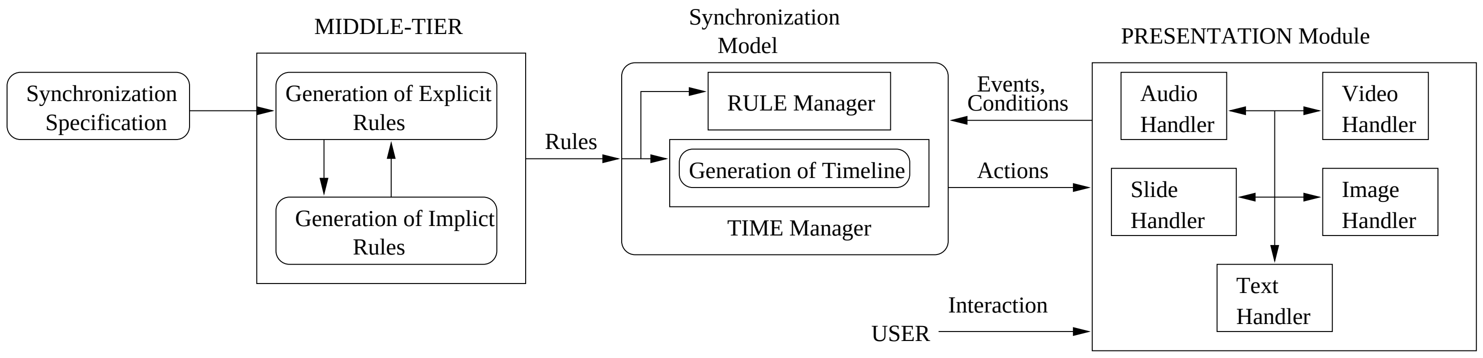


Figure 8.3.1: The role of middle-tier.

8.3.2 Elements of a Multimedia Presentation and SMIL

The basic component of a multimedia presentation is a stream. In our model, a multimedia presentation may have a *container* consisting of containers or other streams. This allows grouping of streams and creation of subpresentations. The containers can signal InitPoint and EndPoint events. This means that the container initiates its presentation and the container ends either one or more of its components end or is ended by other containers or streams.

Explicit rules are generated by processing the synchronization specification. In SMIL 1.0, there are two kinds of grouping, parallel and sequential. The beginning of a group is determined by an event signaled from another group, a stream or the user. If the group is the first group that is presented in the multimedia presentation, the user event *USER(Start)* determines the beginning of the presentation. The parallel grouping corresponds to the list of all actions that will start when the group starts. Thus, if the parallel grouping is like

```
<par>
    <... id=”id1” ...>
    <... id=”id2” ...> ...
    <... id=”idn” ...>
</par>,
```

the synchronization rule is as follows:

```
$R_{par}$ :  on      ...     if    direction=forward  do  start(id1)
                                                          start(id2)
                                                          ...
                                                          start(idn)
```

In the sequential grouping, the end of a stream triggers start of another stream. If the sequential group has *n* elements, there are n-1 rules for the group. Thus, if sequential grouping is like

```
<seq>
    <... id="id1" ...>
    <.. id="id2" ...> ... <... id="id(n-1)"> <.. id="idn" ...>
</seq>
```

The synchronization rules that will be generated are as follows:

```
$R_{seq1}$ :      on   ...                 if   direction=forward  do  start(id1)
$R_{seq2}$ :      on   id1(EndPoint)       if   direction=forward  do  start(id2)
                                                                       ...
$R_{seq(n-1)}$ :  on   $id_{n-1}$(EndPoint)  if   direction=forward  do  start(idn)
```

Notice that the direction is considered as forward in the condition part since the user specifies the requirements for the forward presentation. If time is associated with a start of a stream (e.g. start a stream after 2 seconds), time is considered part of the action rather than part of the event. Including time in the event expression increases the number of rules significantly (i.e. the same event may also trigger other actions).

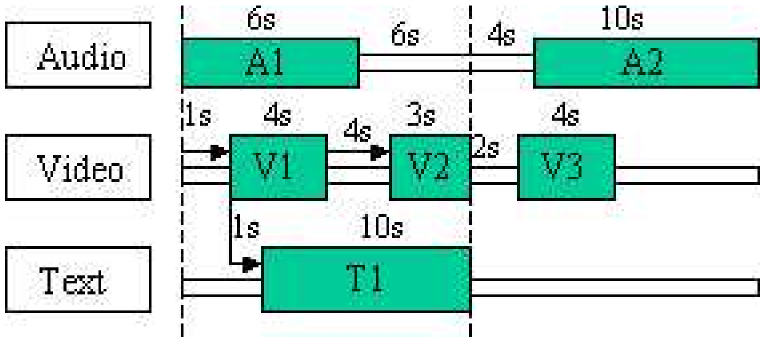


Figure 8.3.2: Sample presentation.

A sample presentation is depicted in Fig. 8.3.2. There are 6 stream elements: A1, A2, V1, V2, V3 and T1. A1 and A2 are audio elements. V1, V2, and V3 are video elements and T1 is a text element. Assume that the presentation is grouped according to the SMIL expression given in Fig. 8.3.3.

There are four containers in the presentation: sequential presentation of V1 and V2 (SEQ1), parallel presentation of A1, T1 and SEQ1 (PAR1), parallel presentation of A2 and V3 (PAR2) and sequential presentation of PAR1 and PAR2 (MAIN). We have the following synchronization rules given in Fig. 8.3.4. The event-action relationships for PAR1 container is depicted in Figure 8.3.5.

```
<seq>
        <par endsync=”last”>
                <audio id=”A1” src=”cnn.aiff”/>
                <seq>
                        <video id=”V1” src=”cnn1.mpv” begin =”1s”/>
                        <video id=”V2” src=”cnn2.mpv” begin =”4s”/>
                </seq>
                <text id=”T1” src=”leader title.html” begin =”id(V1)(1s)” dur=”10s”/>
        </par>
        <par>
                <video id=”V3” src=”cnn3.mpv” begin =”2s”/>
                <audio id=”A2” src=”cnn2.aiff” begin=”4s”/>
        </par>
</seq>
```

Figure 8.3.3: The SMIL expression.

```
(F1)    on  USER(Start)                          if  direction=FORWARD    do   start(MAIN)
(F2)    on  MAIN(InitPoint)                      if  direction=FORWARD    do   start(PAR1)
(F3)    on  PAR1(InitPoint)                      if  direction=FORWARD    do   start(A1)
                                                                               start(SEQ1)
(F4)    on  SEQ1(InitPoint)                      if  direction=FORWARD    do   start(V1,1s)
(F5)    on  V1(InitPoint)                        if  direction=FORWARD    do   start(T1,1s)
(F6)    on  V1(EndPoint)                         if  direction=FORWARD    do   start(V2,4s)
(F7)    on  V2(EndPoint)                         if  direction=FORWARD    do   end(SEQ1)
(F8)    on  (SEQ1(EndPoint) && A1(EndPoint)
            && T1(EndPoint))                     if  direction=FORWARD    do   end(PAR1)
(F9)    on  PAR1(EndPoint)                       if  direction=FORWARD    do   start(PAR2)
(F10)   on  PAR2(InitPoint)                      if  direction=FORWARD    do   start(A2,4s)
                                                                               start(V3,2s)
(F11)   on  (V3(EndPoint) && A2(EndPoint))       if  direction=FORWARD    do   end(PAR2)
(F12)   on  PAR2(EndPoint)                       if  direction=FORWARD    do   end(MAIN)
```

Figure 8.3.4: Forward synchronization rules.

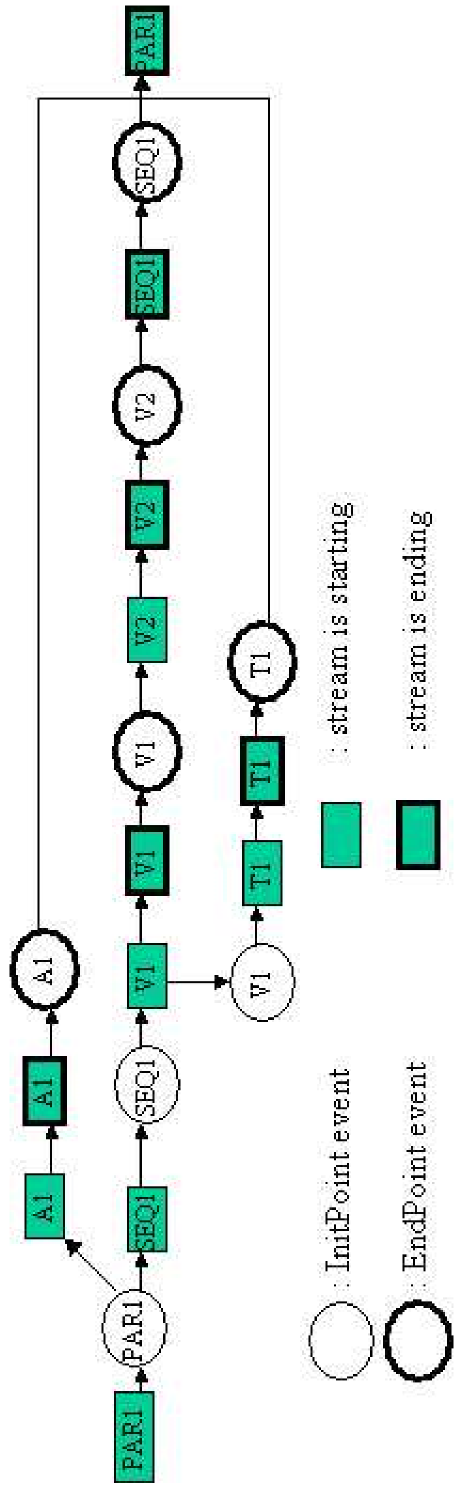


Figure 8.3.5: The event action relationships for PAR1.

8.3.3 Receivers, Controllers and Actors

The synchronization model is composed of three layers: the receiver layer, the controller layer, and the actor layer (Figure 8.3.6). Receivers are objects to receive events. Controllers check composite events and conditions about the presentation such as the direction. Actors execute the actions once their conditions are satisfied.

Definition 8.3.1 *A receiver is a pair $R = (e, C)$, where e is the event that will be received and C is a set of controller objects.*

Receiver R can question the event source through its event e. When e is signaled, receiver R will receive e. When receiver R receives event e, it sends information of the receipt of e to all its controllers in C. A receiver object is depicted in Fig. 8.3.7(a). There is a receiver for each single event. The receivers can be set and reset by the system anytime.

According to the synchronization rules given in Fig. 8.3.4, there are 13 receivers for each event specified in the event expression. These receivers are R1: user(START), R2: MAIN(InitPoint), R3: PAR1(InitPoint), R4: SEQ1(InitPoint), R5: V1(InitPoint), R6: V1(EndPoint), R7: V2(EndPoint), R8: SEQ1(EndPoint), R9: PAR1(EndPoint), R10: PAR2(InitPoint), R11: V3(EndPoint), R12: A2(EndPoint) and R13: PAR2(EndPoint).

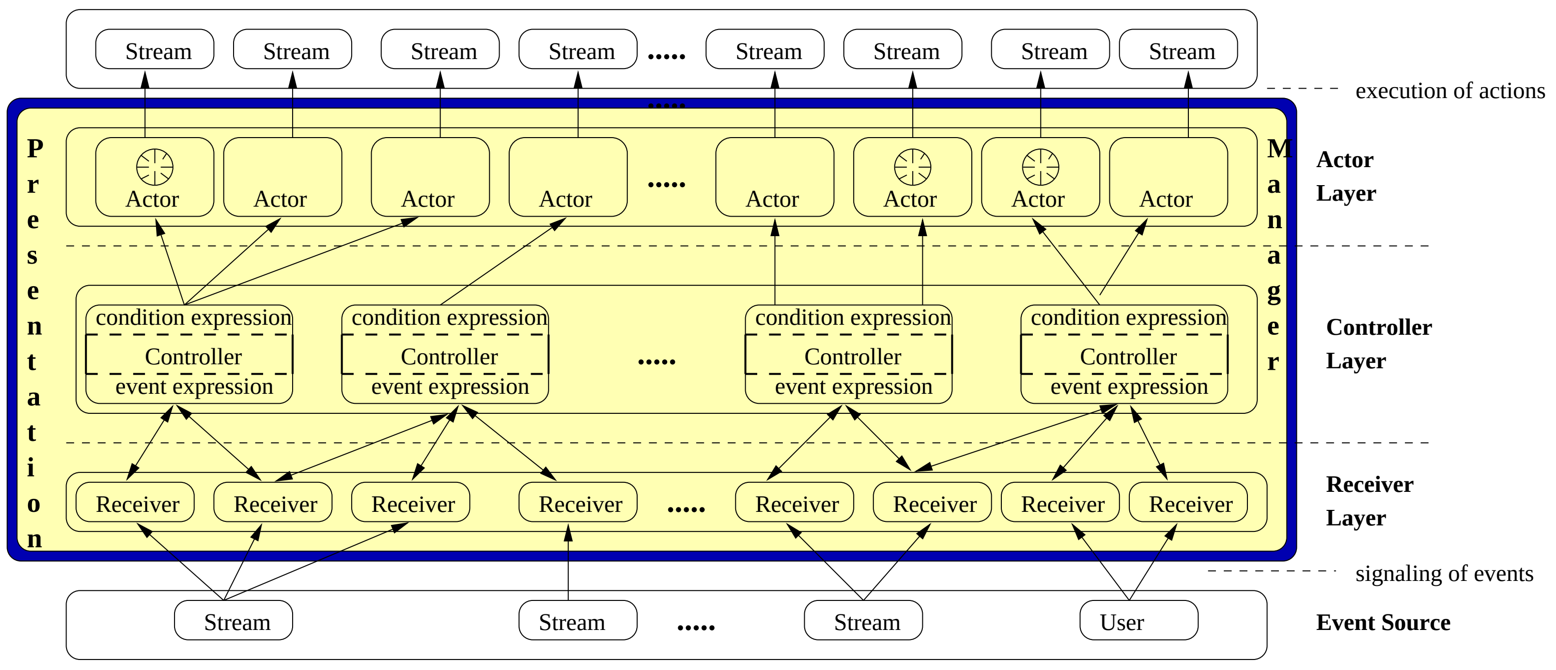


Figure 8.3.6: The layers of the synchronization model.

Definition 8.3.2 *A controller is a 3-tuple $C = (ee, ce, A)$ where ee is an event expression; ce is a condition expression; and A is a set of actors.*

Controller C has two components to verify, composite events *ee* and conditions *ce* about the presentation. When the controller C is notified, it first checks whether the event composition condition *ee* is satisfied by asking the receivers of the events. Once the event composition condition *ee* is satisfied, it verifies the conditions *ce* about the states of media objects or the presentation. After the conditions *ce* are satisfied, the controller notifies its actors in A. A controller object is depicted in Fig. 8.3.7(b). Controllers can be set or reset by the system anytime.

For the synchronization rules given in Fig. 8.3.4, we have a controller for each synchronization rule. So, we have 12 controllers (C1,C2,...,C12) which are listed in Fig. 8.3.8.

Definition 8.3.3 *An actor is a pair $A = (a, t)$ where a is an action that will be executed after time t passed.*

Once actor A is informed, it checks whether it has some sleeping time t to wait for. If t is greater than 0, actor A sleeps for t and then starts action a. If t is 0, action a is an immediate action. If $t > 0$, action a is a deferred action. An actor object is depicted in Fig. 8.3.7(c).

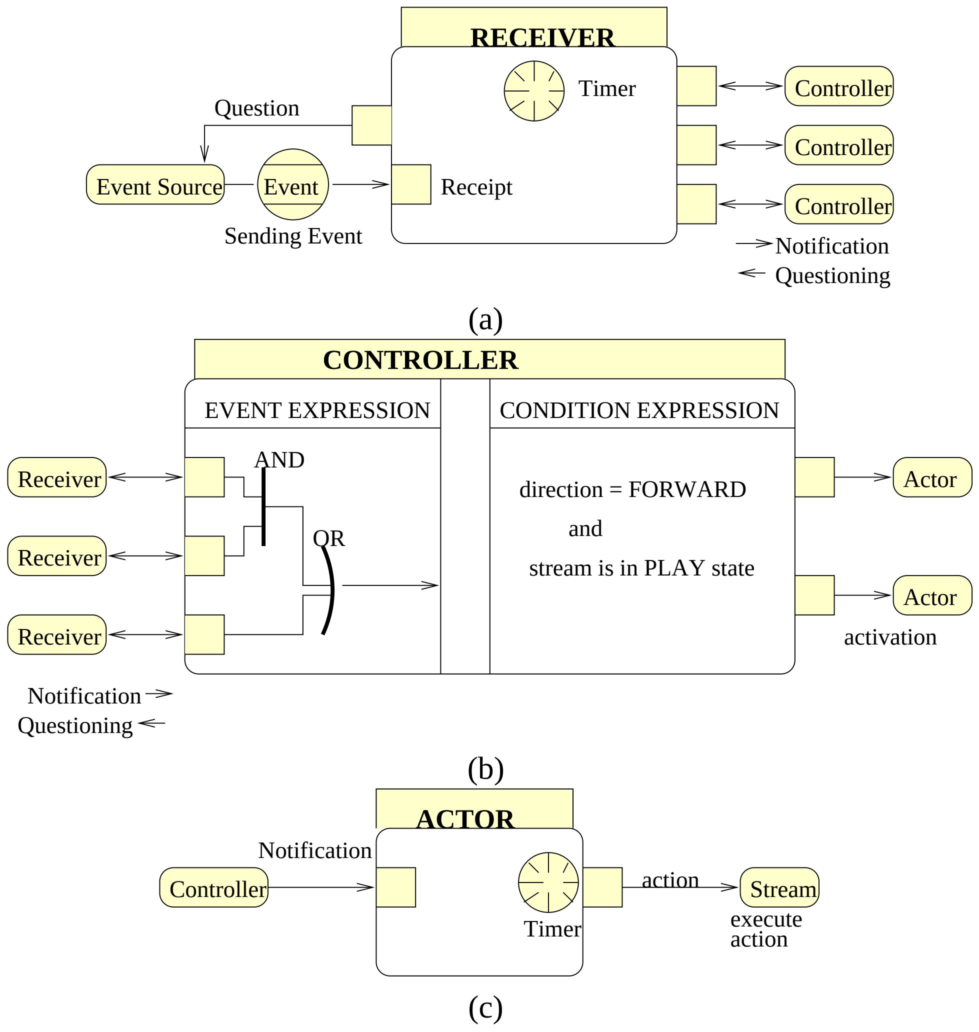


Figure 8.3.7: (a) A receiver object, (b) a controller object, and (c) an actor object object.

For the synchronization rules given in Fig. 8.3.4, we have one actor for each action. So, we have 15 actors (A1,A2,...,A15) which are listed in Fig. 8.3.8.

8.3.4 Timeline

If multimedia presentations are declared in terms of constraints, synchronization expressions or rules, the relationships among streams are not explicit. They only keep the relationships that are temporally adjacent or overlapping. The status of the presentation must be known at any instant. In our work, the presentation timeline object keeps track of all temporal relationships among streams in the presentation.

Definition 8.3.4 *A presentation timeline object is a 4-tuple* $T = (receiverT, controllerT, actorT, actionT)$ *where* $receiverT$*,* $controllerT$*,* $actorT$*, and* $actionT$ *are timelines for receivers, controllers, actors, and actions, respectively.*

The timelines $receiverT$, $controllerT$, $actorT$, and $actionT$ keep the expected times of the receipt of events by receivers, the expected times of the satisfaction of the controllers, the expected times of the activation of the actors, and the expected times of the start of the actions, respectively. Since skip and backward operations are allowed, alive actions, received or not-received events, sleeping actors and satisfied controllers must be known for any point in the presentation. The time of actions can be retrieved from the presentation timeline object.

The information that is needed to create the presentation timeline is the duration of streams and the relationships among the streams. The expected time for the receipt of *realization*, *InitPoint*, and *EndPoint* stream events only depend on duration of the stream and the start time of the action that starts the stream. Since the duration of a stream is already known, the problem is the determination of the start time of the action. The start of the action depends on the activation of its actor. The activation of the actor depends on the satisfaction of the controller. The expected time when the controller will be satisfied depends on the expected time when the event composition condition of the controller is satisfied. Algorithms 8.3.1 and 8.3.2 find the time of the receipt of an event for a receiver and start time of actions, respectively.

Algorithm 8.3.1 Time findReceiptOfEvent(Receiver R)

```
eventSource ← source of event $e$ (of receiver $R = (e, C)$)
if (eventSource=USER and eventType=START) then
    startTime ← 0
else
    find the $action$ (start or display) for eventSource (stream)
    startTime ← compute start time of $action$
    if (event type=END) then
        startTime ← startTime + duration of the stream
    else if (event type=REALIZATION) then
        startTime ← startTime + realization time of the stream
    end if
end if
return start time
```

Algorithm 8.3.2 findStartTimeOfActions()

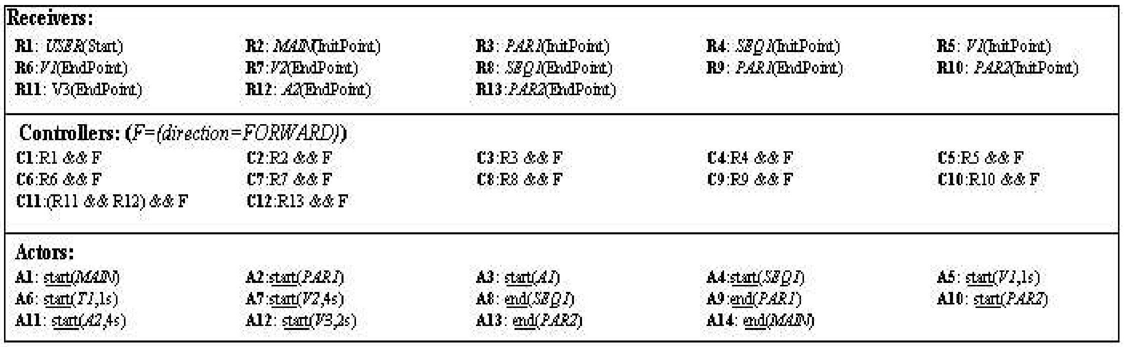
```
// $C_t$: expected time of controller $C$
// $a_t$: expected time of action $a$
for each controller $C = (ee, ce, A)$ do
    compute $C_t$
    for each actor $Actor = (a, t)$ in $A$ do
        $a_t \leftarrow C_t + t$
    end for
end for
```

The expected time for finding the satisfaction of a controller is the expected time of the satisfaction of its event expression. The expected time for the satisfaction of an event composition condition is handled using the composition type. In our model, events can be composed using && and || operators. Assume that ev_1 and ev_2 are two event expressions where $time(ev_1)$ and $time(ev_2)$ give the expected times of satisfaction of ev_1 and ev_2, respectively. Then, the expected time for composite events is found according to the predictive logic for WBT (will become true) in [12]:

$$time(ev_1 \,\&\&\, ev_2) = maximum(time(ev_1), time(ev_2))$$

$$time(ev_1 \,||\, ev_2) = minimum(time(ev_1), time(ev_2))$$

where *maximum* and *minimum* functions return the maximum and minimum of the two values, respectively. The presentation timeline for receivers, controllers, and actors for synchronization rules given in Fig. 8.3.3 are listed in Fig. 8.3.8. On top of the figure the receivers, the controllers and the actors for the presentation are listed. The four timelines



Receivers:				
R1: *USER*(Start)	**R2**: *MAIN*(InitPoint)	**R3**: *PAR1*(InitPoint)	**R4**: *SEQ1*(InitPoint)	**R5**: *V1*(InitPoint)
R6:*V1*(EndPoint)	**R7**:*V2*(EndPoint)	**R8**: *SEQ1*(EndPoint)	**R9**: *PAR1*(EndPoint)	**R10**: *PAR2*(InitPoint)
R11: V3(EndPoint)	**R12**: *A2*(EndPoint)	**R13**:*PAR2*(EndPoint)		
Controllers: (*F=(direction=FORWARD)***)**				
C1:R1 *&&* **F**	**C2**:R2 *&&* **F**	**C3**:R3 *&&* **F**	**C4**:R4 *&&* **F**	**C5**:R5 *&&* **F**
C6:R6 *&&* **F**	**C7**:R7 *&&* **F**	**C8**:R8 *&&* **F**	**C9**:R9 *&&* **F**	**C10**:R10 *&&* **F**
C11:(R11 *&&* R12) *&&* **F**	**C12**:R13 *&&* **F**			
Actors:				
A1: start(*MAIN*)	**A2**:start(*PAR1*)	**A3**: start(*A1*)	**A4**:start(*SEQ1*)	**A5**: start(*V1*,1*s*)
A6: start(*T1*,1*s*)	**A7**:start(*V2*,4*s*)	**A8**: end(*SEQ1*)	**A9**:end(*PAR1*)	**A10**: start(*PAR2*)
A11: start(*A2*,4*s*)	**A12**: start(*V3*,2*s*)	**A13**: end(*PAR2*)	**A14**: end(*MAIN*)	

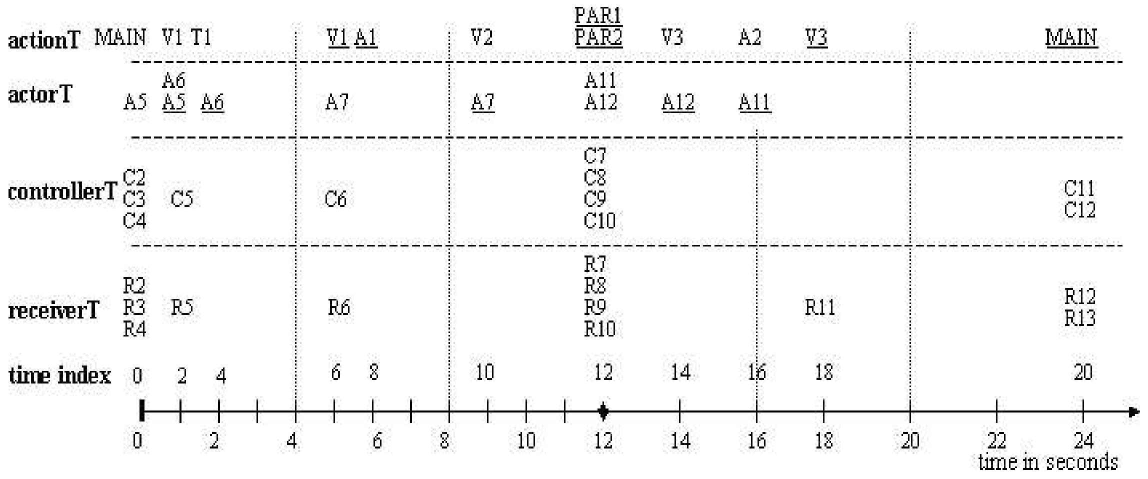


Figure 8.3.8: The presentation timeline.

are shown at the bottom side. The receivers and controllers are ordered according to their expected satisfaction time. Only actors that have a sleeping time greater than 0 are displayed. The name of the actor shows its activation (sleeping time) and underlined actor shows the ending of sleeping time. The actions are also displayed in the same way. The name of the container or the stream shows its starting time and if it is underlined it shows the ending time. At a time instant, if a stream or a container has the same starting time as its container, the main container is shown in the presentation timeline.

8.4 Summary

The RuleSync synchronization model is developed to support the NetMedia [108] system, a middleware design strategy for streaming multimedia presentations in distributed environments. The synchronization is handled by synchronization rules based on event-condition-action (ECA) rules. The middle-tier is responsible for extraction and preprocessing of rules. The synchronization rules can also be extracted from SMIL expressions.

Chapter 9

User Interactions

Specification as in SMIL usually considers forward presentation without considering interactions that change the course of the presentation like backward and skip. Also, management of the presentation constraints becomes harder when the course of the presentation changes. This chapter is organized as follows. The following section explains how basic VCR user interactions are performed. Section 9.2 explain how skip interactions is handled. Section 9.3 discusses how backward interaction is managed.

9.1 Basic User Interactions

The support of VCR functions such as play, pause, resume, forward (fast or slow), backward (fast or slow) and skip strengthens the browsing and access of multimedia presentations. Event-based models can handle play, pause, resume, speed-up, and slow-down operations easier than time-based models. Time-based models need to update the duration and start time of all the objects that participate in the presentation for the pre-specified operations. In event-based models, these interactions only need to inform active streams. In our case, the time is connected to actors. The speed of the presentation is 1 in nominal presentation. If the speed is greater than 1, it is a fast-forward operation. If the speed is less than 1 but greater than 0, it is a slow-forward operation. If the speed is negative, it is a backward operation. When an actor is notified, it only needs to sleep for $(sleepingTime)/(|speedOfPresentation|)$. Speeding up or slowing down only requires the update of the speed of the presentation.

9.2 Skip Operation

A multimedia presentation has a lifetime. The user interactions like skip or changing direction (backwarding when playing forward or vice versa) need to be handled carefully. When skip-forward is performed, some events may be skipped that may cause ignorance of future streams that depend on the receipt of these events. The problem can be solved by using the timeline of the presentation. In the timeline object, the expected time of the receipt of each event and the satisfaction of each controller is known. Therefore, it is known when events should have been received and when the controllers should have been satisfied by tracing the time-trackers of the timeline. It is not always reasonable to start the actors whose controllers are satisfied, since their actions must already have finished (to avoid restart of actions). So, only the actions that will be active at the skip point are started from their corresponding points. The actors whose *sleeping time* has not expired are allowed to sleep for the remaining time.

For example, if a skip is requested to the mid (12 seconds) of the presentation that is shown in Fig. 8.3.3, the timeline is followed in Fig. 8.3.8. Receivers R2, R3, R4, R5, R6, R7, R8, R9 and R10 are assumed to receive their events. Receivers R11, R12 and R13 are assumed that they did not receive their events. Controllers C2, C3, C4, C5, C6, C7, C8, C9, and C10 are assumed to be satisfied. C11 and C12 are assumed to be not satisfied. A satisfied controller cannot notify its actors. It is assumed that it already notified its actors. At the middle point, there is no sleeping actor. The actors A11 and A12 are activated. So, all the actors should be set to their original time. MAIN container should be active since it has elements that are still active. V1, T1, V2 and A1 should be idle. PAR1 should be idle too. PAR2 should be set to its InitPoint so that it can start the streams that it contains. V3 and A2 must also be idle.

9.3 Backward Presentation

If the direction of the presentation is modified, then receiver conditions, controllers, and actors still need to be updated. For example, if the direction is converted from forward to backward, the events that have been received are assumed to have not been received and the events that would have been received later should be set so that the earlier actions (in nominal presentation) can start again.

In our system, the event composition and other conditions for the backward presentation are automatically derived from the declaration of the rules of the forward presentation. So, the author does not have to consider the backward presentation or skipping, and this alleviates the specification of the presentation substantially. It is desirable that the backward synchronization rules are compliant with the forward presentation. Authors usually specify the relationships among streams for some specific reasons. We call these reasons as *author properties*. Assume A and B are streams; C is a container; and ev_1 and ev_2 are events in a presentation. The *author properties* can be listed as follows:

Author Property 9.3.1 ***Dependency*** *If A participates in starting B, B can be used for backwarding A. In this case, there is a dependency between A and B. If streams A and B are not overlapping, dependency property is used.*

Author Property 9.3.2 ***Master-Slave*** *If A influences B by starting or ending, the author* *considered A as a master stream over B. A should be master at this point in the backward direction. If A and B are overlapping, master-slave property is used.*

Author Property 9.3.3 ***Hierarchy*** *If C starts its elements, the end of its elements will participate in ending C in the backward direction. A container ends when all elements are played.*

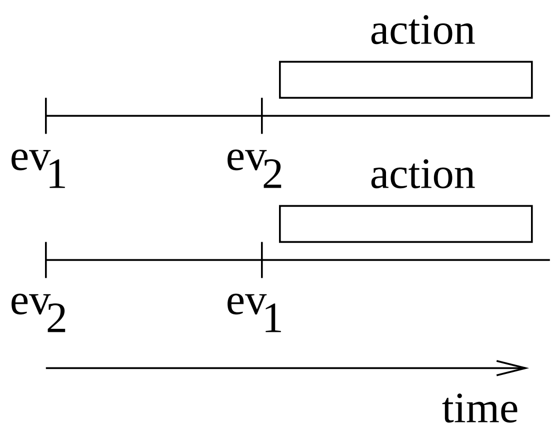


Figure 9.3.1: Co-occurrence.

Author Property 9.3.4 ***Co-occurrence*** *If (ev_1 && ev_2) influences action, their co-occurrence is effective in the forward direction. That is, the action will take place after both events are signaled (Fig. 9.3.1). The action should be terminated when one of the events is signaled in the backward direction (Fig. 9.3.1). This corresponds to self-occurrence in the backward direction.*

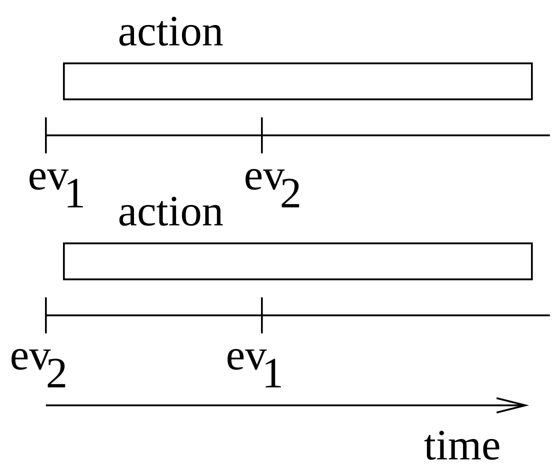


Figure 9.3.2: Self-occurrence.

Author Property 9.3.5 ***Self-occurrence*** *If ($ev_1 \mid\mid ev_2$) influences action, their self-occurrence is effective in the forward direction. That is, the action will take place after one of the events is signaled (Fig. 9.3.2). The action should be terminated when both events are signaled in the backward direction (Fig. 9.3.2). This corresponds to co-occurrence in the backward direction.*

Author Property 9.3.6 ***Realization*** *$A(realization, P)$ corresponds to the realization event of P. P is an ascending number for a stream during forward presentation. In a video, it corresponds to when frame P is displayed. If $A(realization, P)$ influences B in forward direction, then realization of $P-1$ is important for B in the backward direction. This corresponds to the end of playing P.*

The following logic is used for the generation of the backward presentation based on the

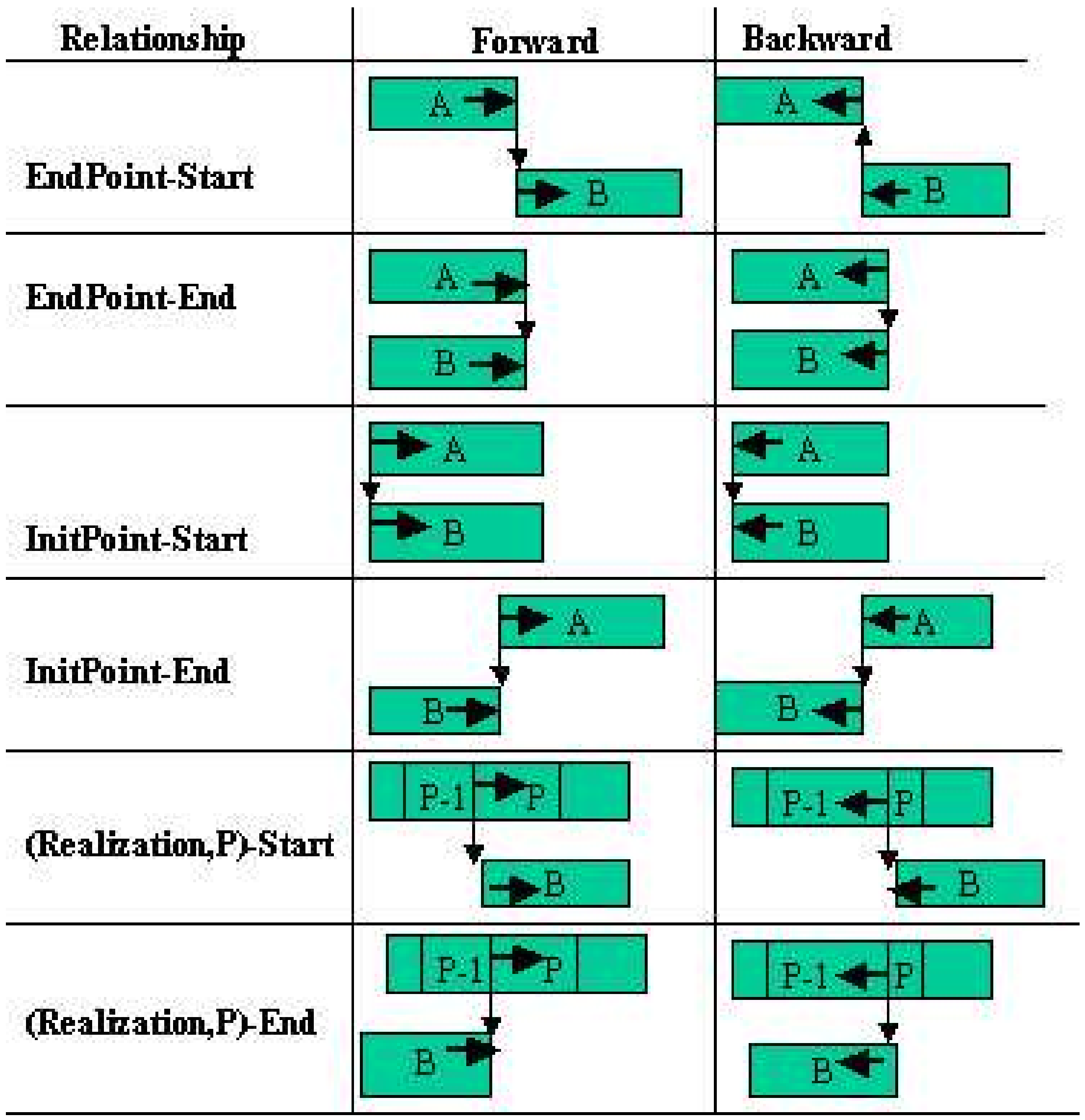


Figure 9.3.3: Forward-backward relationships without time.

previous *author properties*. The relationships are depicted in Fig. 9.3.3.

- **EndPoint-Start Relationship.** If the end of *A* participates in starting *B*, when *B* reaches its beginning in the backward presentation, it will participate in starting *A* in the backward direction *(Dependency property)*.

- **EndPoint-End Relationship.** If the end of *A* participates in ending *B*, when *A* starts in the backward direction, it will participate in starting *B* in the backward direction *(Master-Slave property)*.

- **InitPoint-End Relationship.** If the start of *A* participates in ending *B*, when *A* ends in the backward direction, it will participate in starting *B* in the backward direction *(Master-Slave property)*.

- **InitPoint-Start Relationship.** If the start of *A* participates in starting *B*, when *A* ends in the backward direction, it will participate in ending *B* in the backward direction *(Master-Slave Rule)*. If the start of container *C* starts its elements , when its elements reach their beginning in the backward presentation, they will participate in ending *C* in the backward direction (*Hierarchy property)*.

- **InitPoint and EndPoint Events in Composite Events.** If an InitPoint event or an EndPoint event exists in a composite event, they are treated individually whether the event composition is an AND or OR composition. Therefore, one of the previous rules are applied.

- **Realization-Start Relationship.** The realization points are monotonically increasing within a stream. If $A(realization, P)$ participates in starting B, then $A(realization, P-1)$ will participate in ending B in the backward presentation. At first sight it seems that P should cause the end of B which is not true. Consider the presentation of an image along with slides. Each slide has a duration of 1 minute. The image is displayed when the second slide is displayed. On the timeline, the display of the image will be at the beginning of the 2^{nd} minute. The image must be closed when the first slide is displayed in the backward presentation and must be visible during the second slide *(Realization property)*.

- **Realization-End Relationship.** Assume that the realization point is P again. If $A(realization, P)$ participates in ending B, then $A(realization, P-1)$ will participate in starting B in the backward presentation *(Realization property)*.

- **Realization Events in Composite Events.** Let $A(realization, P_1)$ and $B(realization, P_2)$ be realization events for streams A and B. If $(A(realization, P_1) \&\& B(realization, P_2))$ causes some actions in the forward presentation, $A(realization, P_1 - 1) \,||\, B(realization, P_2 - 1)$ will cause actions in the backward presentation. Because the actions become active when both of the events are realized in the forward direction, the actions should be active as soon as one of the events is realized in the backward direction. In this way, the integrity and the consistency of the presentation can be protected. If $(A(realization, P_1) \mid\mid B(realization, P_2))$ causes some actions in the forward presentation, $(A(realization, P_1 - 1)$ && $B(realization, P_2 - 1))$ will cause actions in the backward presentation *(Co-occurrence and self-occurrence properties)*.

9.3.1 Management of Time in Backward Presentation

The management of time relationships for backward presentation may introduce some ambiguities. These ambiguities can be solved by correcting the specification or making assumptions on the specifications. Figure 9.3.3 depicts the relationships that include time. EndPoint-Start and EndPoint-End relationships can be converted to backward relationships as shown in Figure 9.3.3. Backwarding InitPoint-Start and InitPoint-End relationships is not easy to handle. We denoted time with t^* for backward presentation if there is ambiguity. In fact, t^* should be equal to t. However, it is hard to satisfy this equality in the backward presentation. So, we will first explain the ambiguity and then propose a solution using realization events.

There is ambiguity in using time in the literature and in synchronization models. Assume a video stream has duration of 10 seconds. The question is whether expressions “7 seconds after the beginning” and “3 seconds before the end” are equivalent or not. In a network, the presentation of a video stream may have different durations due to delay and loss on

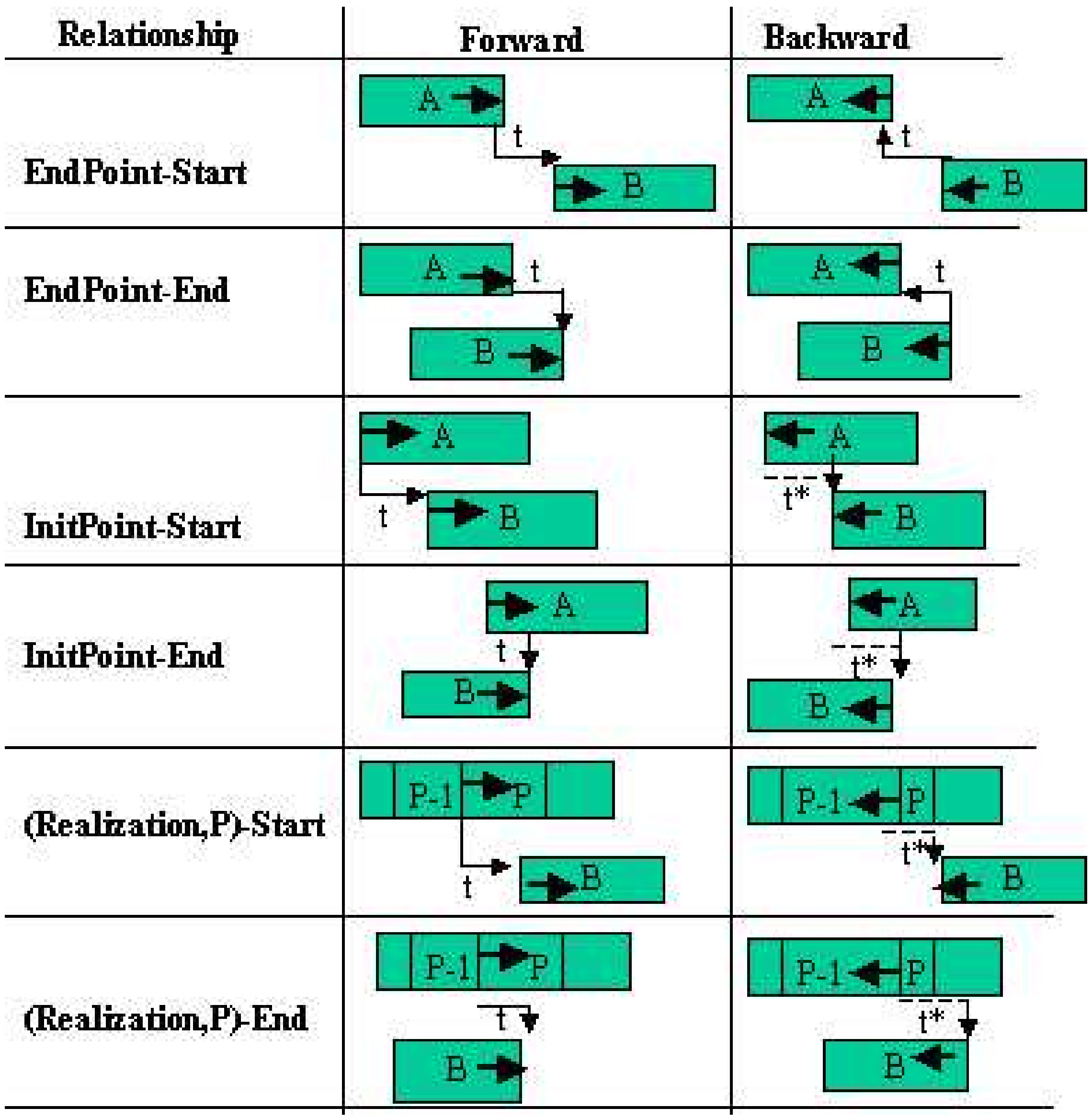


Figure 9.3.4: Forward-backward relationships with time.

the network. It is even harder to figure out when the video will end if the data has not arrived yet. In the specification, this may be considered equivalent. The author usually uses time because of its simplicity. In most cases, this kind of usage corresponds to the realization event. This should be clear either by default by the specification or should be stated explicitly. The expression “7 seconds after the beginning” often means “7 seconds after the beginning in nominal display”. Thus, in this one, it is 70% of the video stream. This clarification should be made for the forward presentation so that backward rules can be converted using realization events. Assume videos $V1$ and $V2$ are played at 30 frames per second. If V2 has to be played 1 minute after V1 starts, this requirement is converted to “V2 has to be started after frame 1800 of $V1$ is played”. In the backward presentation, V2 should be terminated when frame 1799 of $V1$ is played to make it consistent with the forward presentation.

There are two ways to handle the EndPoint-End relationship with time for backward presentation. Assume stream A and stream B both have duration of 5 seconds at 30 frames/second and t is 1 second. Stream A can be backwarded 1 second after starting backwarding stream B. If they are overlapping, a realization event can be used instead of time. In this case, stream A can be backwarded after frame 119 of stream B is displayed in the backward direction. Frame 120 is displayed after 1 second display of stream B in nominal backward presentation. A realization event is signaled at frame 119 instead of frame 120 due to realization event property. The same case applies for the InitPoint-End relationship.

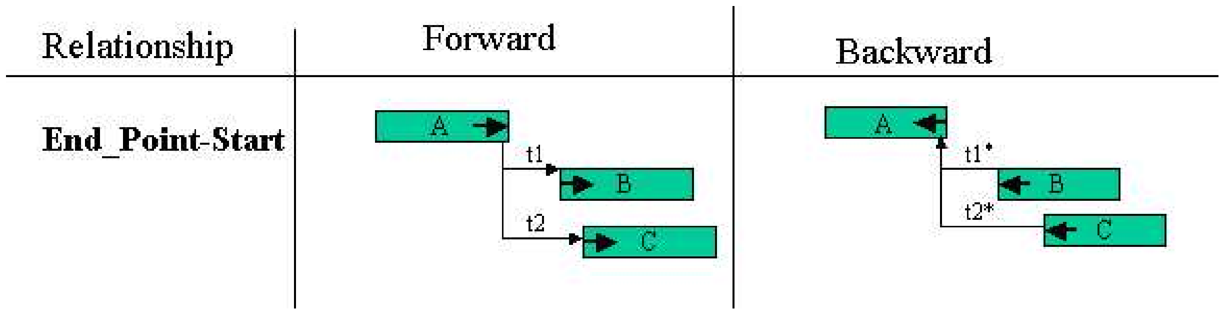


Figure 9.3.5: Ambiguity in relationships with time.

The problem is even clearer when realization events are used with actions having time. We believe that if an action happens based on a time instant referring to a realization point, it does not make much sense. For example, video $V2$ should be started 10 seconds after frame 1800 of $V1$ is played. In the backward presentation, the reference point to start counting 10 seconds before reaching frame 1799 of $V2$ is difficult to estimate. The influencing stream ($V1$) already plays, and the realization point may be moved to a new point instead of using time.

Time should be used if the referring stream has already ended. There is no other choice in this case. EndPoint-Start relationship is an example to this. EndPoint-End relationship may utilize realization event in the backward direction, since they are overlapping.

If there is a case like in Figure 9.3.5 where an event starts many other streams with different starting times, the backwarding of A is not clear since B and C may not finish backwarding at the expected times. We suggest that stream A may be backwarded after $minimum(t1,t2)$ after B and C finish backwarding.

The synchronization requirements need to be specified more accurately. If two streams are overlapping and time-based relationship between them is specified, it has to be distinguished from a realization event. For backward presentation, it is favorable to use realization events for forward presentation if possible instead of time. Whatever synchronization rules are generated for the backward presentation, the author may have different backward presentation. The author should always be allowed to update the rules.

9.3.2 Synchronization Rules for Backward Presentation

Each forward synchronization rule is processed to generate backward synchronization rules. The events in the event expression and the actions in the action expression are extracted to determine the relationships given in the previous subsections.

In our system, InitPoint and EndPoint events have dual actions in the backward presentation. The dual actions for InitPoint and EndPoint is *backend* and *backward*, respectively. In the backward direction, InitPoint and EndPoint events are signaled when *backend* and *backward* actions are performed, respectively. The actions *start* and *end* also have dual events. The dual events for *start* and *end* actions are InitPoint and EndPoint, respectively. Actions have also dual actions. The actions *start* and *end* have dual actions *backend* and *backward*, respectively. Conditions have also dual conditions. The condition (direction=BACKWARD) is the dual condition for (direction=FORWARD).

We will give examples on how backward synchronization rules in Figure 9.3.6 are generated from forward synchronization rules in Figure 8.3.4. The synchronization rule F1 declares what to do when the user starts the presentation. There will be a corresponding rule for the backward presentation. This rule (B1) determines what to do when the user backwards the presentation from the end. For user(START) event in F1, there is user(BACKWARD) event in B1. The action is backward(MAIN) in B1 for start(MAIN) action in F1.

The synchronization rule F2 has an InitPoint event and a start action. This corresponds to an InitPoint-Start relationship. MAIN is the container of PAR1 and backward rule will be generated using the hierarchy rule. The dual event for *start* action is InitPoint. The event expression will be PAR1(InitPoint). The action expression will be backend(MAIN). All the condition expressions will be *direction=BACKWARD*. The corresponding backward rule is B2 for F2. The synchronization rule F4 also contains a similar relationship but with time. Since V1 starts 0.5 seconds after the beginning of SEQ1 in F4, the time is included in the action expression of B4 as backend(SEQ1,0.5s).

F3 has two InitPoint-Start relationships. Since there are two *start* actions (start(A1) and start(SEQ1)), there will be two events in the event expression of the backward rule. The event expression will be $SEQ1(InitPoint)\&\&A1(InitPoint)$. The events are composed using AND composition to ensure that both actions complete their executions. Whenever dual events are generated from action expression, these events are composed using AND composition. Because AND composition is stricter than OR composition, AND composition is preferred. There is no author rule about the composition of dual events. The action expression will only have *backend(PAR1)* due to InitPoint event in F3. The corresponding backward rule is B3. F10 contains an InitPoint-Start relationship, which causes ambiguity in the backward rule generation. A2 and V3 start 2 seconds and 1 second later than PAR2, respectively. The minimum of 2 seconds and 1 second is 1 second. Therefore, the action in B10 will be backend(PAR2,1s).

F5 has an InitPoint-Start relationship with time. This relationship is handled using the master-slave rule, since V1 is not the container of T1. The time will be associated with the action in the backward rule B6. This backward rule is generated using a realization event. Since T1 is the slave in F5, it will also be the slave in the backward rule. The InitPoint event is converted to a realization event. The event expression becomes V1(realization, 0.5s) and the action expression becomes backend(T1).

F6 contains an EndPoint-Start relationship with time. The event expression will be V2(InitPoint) and the action expression will be backward(V1,2s) in backward rule B6. F9 also contains an EndPoint-Start relationship as in F6 but without time. There will be no time in the action expression of B9.

F7 contains an EndPoint-End relationship. The dual event for action end(SEQ1) is SEQ1(EndPoint) and the dual action for event V2(EndPoint) is backward(V2). F8 also contains an EndPoint-End relationship and the backward rule B8 is generated in the same way. F8 is related with the termination of PAR1 and its elements. It has 3 EndPoint-End relationships. In nominal presentation, the elements of PAR1 (A1, SEQ1, and T1) end at different times. The times when PAR1 and its elements end can be detected from the timeline (Figure 8.3.8). A1 ends at 3 seconds and PAR1 ends at 6 seconds. A1 will be backwarded after 3 seconds after beginning backwarding PAR1. F11 contains two EndPoint-End relationships. The dual event for action end(PAR2) is PAR2(EndPoint). The dual actions for V3(EndPoint) and A2(EndPoint) are backward(V3) and backward(A2). However, in the forward presentation V3 ends 4 seconds earlier. The difference is added to the action part as time. Thus the actions will be backward(V3,4s) and backward(A2). F12 is another example of an EndPoint-End relationship.

(B1)	on	user(BACKWARD)	if	direction=BACKWARD	do	backward(MAIN)
(B2)	on	PAR1(InitPoint)	if	direction=BACKWARD	do	backend(MAIN)
(B3)	on	(SEQ1(InitPoint) && A1(InitPoint))	if	direction=BACKWARD	do	backend(PAR1)
(B4)	on	V1(InitPoint)	if	direction=BACKWARD	do	backend(SEQ1,1s)
(B5)	on	V1(Realization,1s)	if	direction=BACKWARD	do	backend(T1)
(B6)	on	V2(InitPoint)	if	direction=BACKWARD	do	backward(V1,4s)
(B7)	on	SEQ1(EndPoint)	if	direction=BACKWARD	do	backward(V2)
(B8)	on	PAR1(EndPoint)	if	direction=BACKWARD	do	backward(A1,6s)
						backward(SEQ1)
						backward(T1)
(B9)	on	PAR2(InitPoint)	if	direction=BACKWARD	do	backward(PAR1)
(B10)	on	(V3(InitPoint) && A2(InitPoint))	if	direction=BACKWARD	do	backend(PAR2,2s)
(B11)	on	PAR2(EndPoint)	if	direction=BACKWARD	do	backward(A2)
						backward(V3,8s)
(B12)	on	MAIN(EndPoint)	if	direction=BACKWARD	do	backward(PAR2)

Figure 9.3.6: Backward synchronization rules.

9.4 Summary

The backward rules are generated automatically based on *author properties* and forward presentation. In this chapter, not only forward temporal relationships are converted to reverse temporal relationships but also the relationships between events and actions are considered for backward presentation.

Chapter 10

Model Checking of the Synchronization Model

The modeling and verification of flexible and interactive multimedia presentations are important for consistent presentations over networks. There has been querying languages proposed whether the specification of a multimedia presentation satisfy inter-stream relationships. Since these tools are based on the interval-based relationships, they cannot guarantee the verification in real-life presentations. Moreover, the user interactions that change the course of the presentation like backward and skip are not considered in the presentation. It is very crucial whether the model satisfies the requirements of the multimedia author. Although there have been conceptual models proposed using Petri-Nets, it is very difficult for an ordinary author design Petri-Nets and verify the requirements. Using PROMELA/SPIN, it is possible to verify the temporal relationships between streams using our model including user interactions that change the course of the presentation. Since the model considers the delay of data, the author is assured that the requirements are really satisfied.

A multimedia presentation is a presentation of multimedia streams in a synchronized fashion. There have been models proposed for the management of multimedia presentations. The synchronization specification languages like SMIL [88] have been introduced to properly specify the synchronization requirements. Multimedia query languages are developed to check the relationships defined in the specification [42]. These tools check the correctness of the specification. However, some synchronization tools have some limitations and do not satisfy all the requirements. The properties that are specified in the specification may not be satisfied by the synchronization tool. Moreover, the specification does not include user interactions. The previous query-based verification techniques cannot verify whether the system is really in a consistent state after a user interaction.

There are also verification tools to check the integrity of multimedia presentations [66]. The user interactions are limited and interactions like backward and skip are ignored. This kind of interactions is hard to model. The Petri-Nets are also used to verify the specification of multimedia presentations [73]. But Petri-Net modeling requires immense Petri-Net modeling for each interaction possible. Authors usually do not have much information about Petri-Nets.

The most common methods for verification of finite-state concurrent systems are simulation, testing, and deductive reasoning. It is not possible to consider all the cases in simulation and testing. If there is a severe problem, it may even be costly to verify by testing and simulation. Deductive reasoning usually requires experts to verify and even sometimes they may not be able to verify. The major advantages of model checking are that it is automatic and usually fast. The counter examples are produced by the model checking tools. We believe that multimedia models should consider model checking first before implementation of the real system. We use PROMELA [45] as the specification language and SPIN [44] as the verification tool. These tools are publicly available and Linear Temporal Logic (LTL) formulas can be verified. The conversion of rules to PROMELA is briefly explained in [10].

PROMELA/SPIN is a powerful tool for modeling and verification of software systems [44]. Since PROMELA/SPIN traces all possible executions among parallel running processes, it provides a way of managing delay in the presentation of streams. In this chapter, we discuss the properties that should be satisfied for a multimedia presentation. We analyze the complexity introduced by user interactions. The interactions that change the course of the presentation like backward and skip are also investigated. The experiments are conducted for parallel, sequential, and synchronized presentations.

Model checking consists of three phases: modeling, specification, and verification. In our system, the user, the user interface, streams, containers, receivers, controllers and actors have to be modeled firstly. In the modeling phase, the model should be kept simple and avoid unnecessary details. The unnecessary details increase the amount of computation for verification. More importantly they may cause false results. Therefore, we make some abstractions to ensure the correctness of the model. The major components of the model are events, conditions, actions, receivers, controllers, actors, the presentation and the streams. The abstraction is performed on the streams and the user interface.

This chapter is organized as follows. The modeling and specification are explained in Section 10.1 and 10.2, respectively. Section 10.3 reports our experiments and analysis on model checking. Section 10.4 summarizes the chapter.

10.1 Modeling

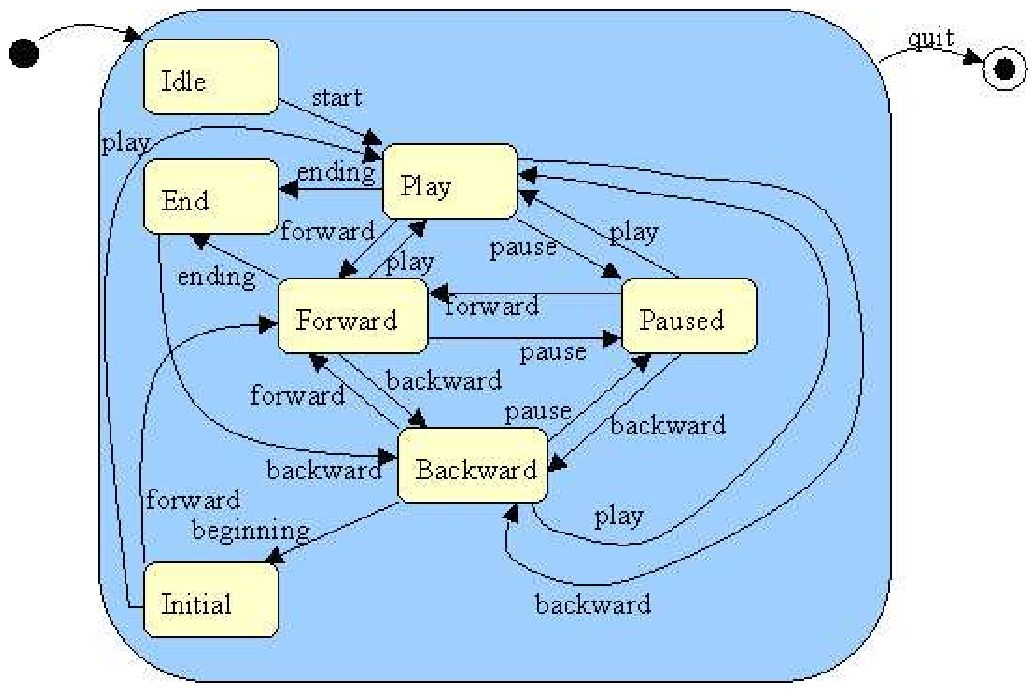
10.1.1 Presentation

The presentation can be in the Idle, Initial, Play, Forward, Backward, Paused, and End states (Fig. 10.1.1 (a)). The presentation is initially in the Idle state. When the user clicks

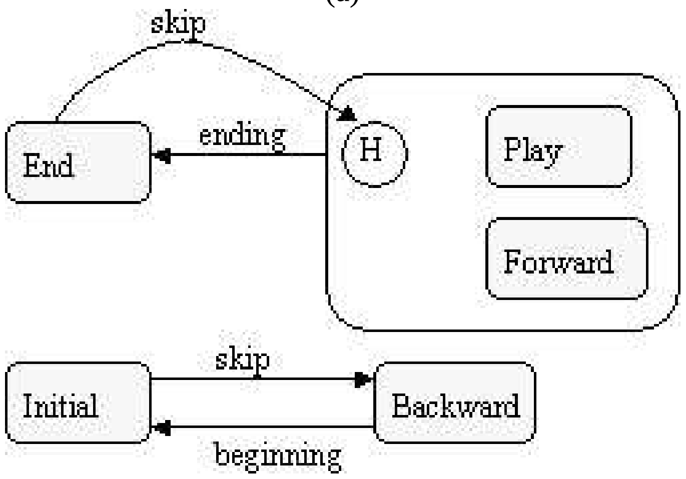
START button, the presentation enters the Play state. The presentation enters the End state, when the presentation ends in the forward presentation. The presentation enters the Initial state when it reaches its beginning in the backward presentation. The user may quit the presentation at any state. Skip can be performed in play, forward, backward, initial, and end states. If the skip is clicked in the Play, Forward, and Backward states, it will return to the same current state after skip unless skip to the Initial or End state is not performed. Therefore, if the state is Play before skip, the state will be after skip. If the presentation state is in the End state or in the Initial state, skip interaction will put into the previous state before reaching these states (Fig. 10.1.1 (b)). The presentation changes states as the user clicks on the button. The most important state variable of the presentation is the direction.

10.1.2 Containers and Streams

A container may enter four states. It is in IdlePoint state initially. Once started, a container is at InitPoint state in which it starts the containers and streams that it contains. After the InitPoint state, a container enters its RunPoint state. In RunPoint state, a container knows that it has some streams that are being played. When all the streams it contains reaches to their end or when the container is notified to end, it stops execution of the streams and signals its end and then enters idle state again. In the backward presentation, the reverse path is followed (Fig. 10.1.2 (a)).

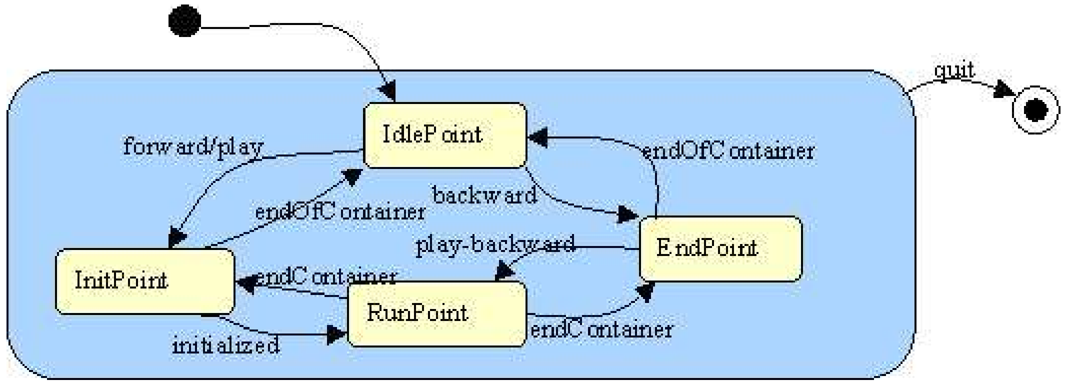


(a)

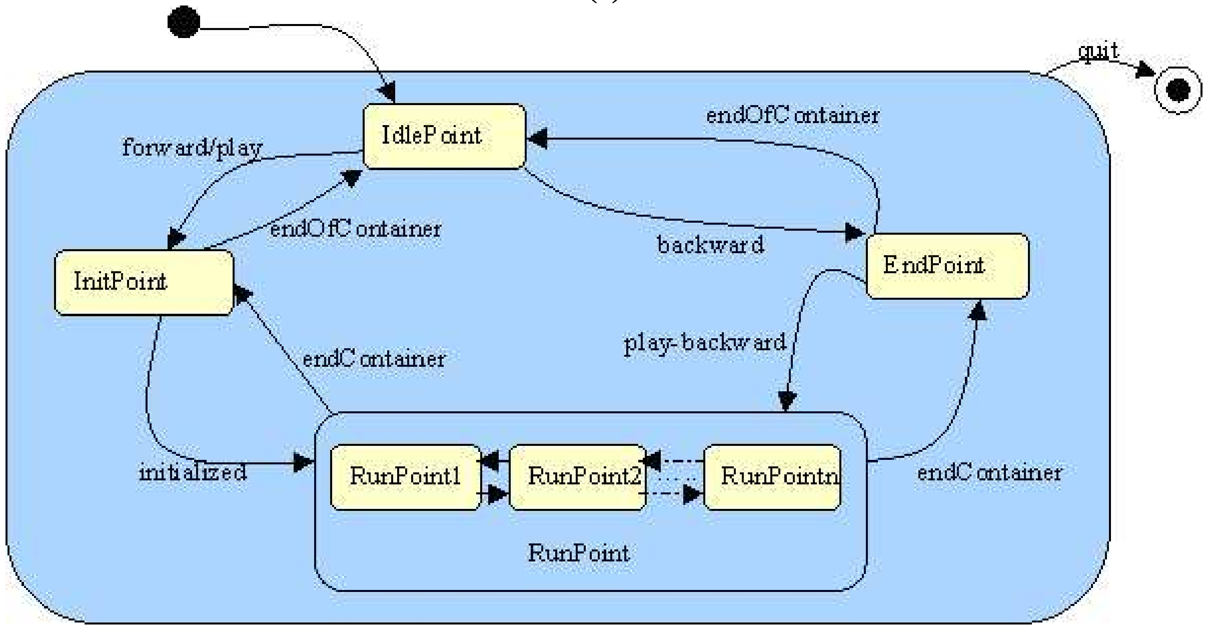


(b)

Figure 10.1.1: The presentation states, (a) general state transitions (b) state transitions for skip.



(a)



(b)

Figure 10.1.2: (a) Container states (b) stream states

A stream is similar to the container. Since the number of states grows exponentially, some abstractions have to be made on modeling a stream. Since we are interested in inter-stream relationships, the points that affect the inter-stream relationships are considered. From the modeling point of view, if the displaying or playing a specific segment of a stream does not affect the playout of the presentation, there is no need to handle each segment of the stream. The streams only affect the model by the events that are triggered by the streams. The (dis)play of a stream element is in the model if it triggers an event. If a video stream has 100 frames without an event for a frame, displaying a frame at an instant is not important if it does not trigger an event. The slow display of frames corresponds to a delay in playing the stream.

If a stream does not signal any event except its start and end, the stream enters the same four states as a container. If a stream has to signal an event, a new state is added to RunPoint state per event. So after the stream signals its event, it is still in the RunPoint state (playing mode) (Fig. 10.1.2 (b)). Since the realization for backward presentation is also considered, there is another event (also state) for the backward presentation. In this sense, each realization event adds two states. One is used for forward presentation and the other is used for the backward presentation. The following is a PROMELA code for playing a stream.

```
proctype playStream (byte stream) {
#if (FC==3 || FC==4 || FC==5 || FC==6)
progressIdleStreams:
```

```
#endif
do
#if FC!=4
   ::  atomic{ (eventHandled && getState() == RUN) &&
        (getStream(stream) == InitPoint) ->
        printf("Starting stream %d",stream);
        setStream(stream, RunPoint);
        if
        ::  (stream==A1)->timeIndex=1;
        ::  else -> skip;
        fi; }
   ::  atomic{ (eventHandled && getState() == RUN) &&
        (getStream(stream) == RunPoint) ->
        printf("Playing stream %d",stream);
        setStream(stream, EndPoint);}
   ::  atomic{ (eventHandled && getState() == RUN) &&
        (getStream(stream) == EndPoint) ->
        to_end:  printf("Ending stream %d",stream);
                setStream(IdlePoint);
                signalEvent(stream,EndPoint) }
#endif
#if (FC!=3 && FC!=5 && FC!=6)
   ::  atomic{ (eventHandled && getState() == BACKWARD) &&
        (getStream(stream) == InitPoint) ->
        to_init:  printf("Ending backwarding stream %d",stream);
                setStream(IdlePoint);
                signalEvent(stream,InitPoint);}
   ::  atomic{ (eventHandled && getState() == BACKWARD) &&
        (getStream(stream) == RunPoint) ->
        printf("Playing stream %d backward",stream);
        setStream(stream, InitPoint);}
   ::  atomic{(eventHandled && getState() == BACKWARD) &&
        (getStream(stream) == EndPoint) ->
        to_backward:  printf("Backwarding stream %d",stream);
                if
                ::  (stream==A1)->timeIndex=1;
                ::  else -> skip;
                fi;
                signalEvent(stream,EndPoint);
                setStream(stream,RunPoint); }
#endif
   ::  atomic{ (eventHandled && getState() == QUIT) ->
        to_playStream_quit:  goto playStream_quit;}
   ::  else -> skip;
   od;
   playStream_quit:  skip;
}
```

The #if directives are used for hard-coded fairness constraints. There are three states for forward and backward presentation. The cases at lines 7, 14 and 15 correspond to forward presentation. The cases at lines 26, 31 and 35 correspond to the backward presentation. The case at line 45 is required to quit the process. The else statement at line 49 corresponds to IdlePoint state. Streams signal events as they reach to the beginning and end (lines 23 and 30). The variable *eventHandled* is used to check whether the system enters a consistent state after a user interaction. The *atomic* command enables execution of statements in a single step thus reduces the complexity and increases safety. The checking and updating the stream state has to be performed in a single step since the stream state may also be updated by the system after an user interaction. Labels like *to_init, to_end, playStream_quit* are added to create LTL formulas related with these points of the presentation. PROMELA may proceed to any state separated with :: in a *do* loop.

10.1.3 Receivers, Controllers and Actors

A receiver is set when it receives its event. As long as there is no user interaction, a receiver will stay at this state for the rest of the presentation. Thus a controller that requires a receipt of this event can be satisfied later. When a controller is satisfied, it activates its actors. And to disable the reactivation of the actors, the controller is reset. An actor is either in idle or running state to start its action after sleeping. Once it wakes up, it starts its action and enters the idle state. The following is a code for receiver definition (lines 51-52), controller satisfaction (lines 54-59) and actor activation (lines 61-64). The expression “receivedReceiver(receiver_Main_INIT)” (line 52) corresponds to the receipt of the event when the main container starts. The expression “setActorState(...,RUN_POINT)” activates the actors (line 58-59). The expression “activateActor(actor_Main_START)” (line 63) elapses the time and the action follows (line 64).

```
51  #define Controller_Main_START_Condition
52     (receivedReceiver(receiver_Main_INIT) && (direction==FORWARD))
53
54  ::  atomic{(eventHandled
55     && !(satisfiedController(controller_Main_START))
56     && Controller_Main_START_Condition) ->
57          setController(controller_Main_START);
58          setActorState(actor_A1_START,RUN_POINT);
59          setActorState(actor_A2_START,RUN_POINT)}
60
61  ::  atomic{(eventHandled
62     && getActorState(actor_Main_START) == RUN_POINT) ->
63          activateActor(actor_Main_START);
64          setContainerState(Main,INIT_POINT);}
```

10.1.4 User and User Interface

The user interface provides seven buttons: START, PLAY, PAUSE, FORWARD, BACKWARD, SKIP, and QUIT. Each button may enter two states in the model. A button is either in enabled or disabled state. As the presentation changes states, the states of the buttons may change. Fig. 10.1.1 shows the possible state transitions with enabled user interactions. The user interface is based on the model presented at [22].

For example, if a skip is requested to the mid (12 seconds) of the presentation that is shown in Fig. 8.3.2, the timeline is followed in Fig. 8.3.8. Receivers R2, R3, R4, R5, R6, R7, R8, R9, and R10 are assumed to receive their events. Receivers R11, R12, and R13 are assumed that they did not receive their events. Controllers C2, C3, C4, C5, C6, C7, C8, C9, and C10 are assumed to be satisfied. C11 and C12 are assumed to be not satisfied. A satisfied controller cannot notify its controllers. It is assumed that it already notified its actors. At the middle point, there is no sleeping actor. The actors A11 and A12 are activated. So, all the actors should be set to their original time. MAIN container should be set to running point. V1, T1, V2, and A1 should be idle. PAR1 should be idle too. PAR2 should be set to its InitPoint so that it can start the streams that it contains. V3 and A2 should also be set to the IdlePoint.

There are infinite number of skips that can be performed by the user. The timeline shown in

Fig. 8.3.8 is divided into pieces where the streams perform similar behavior in each piece. There are 21 pieces that are determined by starting and ending actors and actions. So, it is only possible to perform 21 skips.

On the other hand, the backward modifies the direction of the presentation. The synchronization model needs to synchronize after changing direction since streams may proceed at different speeds. To synchronize, the time at that instant should be known. We define a time index which is initially 0 and can be maximum the number of pieces. Some specific streams are allowed to increase or decrease after the time index and the current time index can be determined (lines 12,19). The necessary actors, actions, receivers, and controllers are set and reset after changing the direction.

10.2 Specification

Specification consists of the properties that a model should satisfy once the model enters some specific states. SPIN automatically checks whether the elements like user, streams, and containers reach their possible states. If not, this is reported by the tool.

There are two basic properties that should be checked: *safety properties* and *liveness properties*. *Safety properties* assert that the system may not enter undesired state. *Liveness properties* on the other hand assure that system executes as expected. Liveness includes the progress, fairness, reachability and termination of the system. Progress corresponds whether the system will *eventually* enter a state. Absence of progress is considered as starvation. Fairness determines whether an action is *eventually* executed. Reachability addresses some specific states are reachable from another state. Termination is related whether the program terminates. Linear Temporal Logic formulas are properties of paths rather than properties of states. Therefore, an LTL formula is interpreted with respect to a fixed path. The operators $\Box$, $\Diamond$, and U correspond to *globally, eventually,* and *until*, respectively. We present the following properties about a multimedia presentation.

Fairness Constraint 1 *The user is only allowed to click START button and clicks START button and no user Interaction is allowed after that. This constraint is expressed as:* $\Box\ (userStart \rightarrow \Box noInteraction)$

Fairness Constraint 2 *The user always clicks enabled button. This is expressed as* $\Box\ (userClickButton \rightarrow buttonEnabled)$

If a property is stated as undesirable, the system should not allow it. We first start with the properties about transitions that are allowed by buttons.

Property 1 *Clicking button for START enables buttons for PAUSE, FORWARD, and BACKWARD, and it changes the presentation state to RUN. (requires fairness constraint 2)*

Property 2 *Clicking button for PAUSE enables buttons for PLAY, FORWARD, and BACKWARD and it changes the presentation's state to PAUSED. (requires fairness constraint 2)*

Property 3 *The buttons for BACKWARD and SKIP are enabled, and the buttons for START, PLAY, PAUSE, and FORWARD are disabled after the presentation reaches its end. (requires fairness constraint 2)*

Property 4 *The user interface should ignore if the user clicks a disabled button. (requires fairness constraint 2)*

Property 5 *The button for PAUSE is enabled only when the presentation is in RUN, FORWARD, or BACKWARD states. (requires fairness constraint 2)*

Property 6 *The button for SKIP is enabled when the presentation is in RUN, FORWARD, BACKWARD, INITIAL, or END states. (requires fairness constraint 2)*

Property 7 *The buttons for PLAY, FORWARD, and BACKWARD are enabled when the presentation is in PAUSED state. (requires fairness constraint 2)*

Property 8 *The button for START is enabled only when the presentation is in IDLE state. (requires fairness constraint 2)*

Property 9 *The button for PAUSE is enabled outside RUN, FORWARD, and BACKWARD. (undesirable, requires fairness constraint 2)*

Property 10 *Buttons for START, PAUSE, PLAY, FORWARD, and BACKWARD are in enabled condition at any particular time. (undesirable, requires fairness constraint 2)*

Some refinements are needed to convert the properties to LTL formulas. In the following formulas, *actionButtonClicked* corresponds to successful clicking *Button* when the button is enabled. *actionToState* corresponds to state transition to *State*. ButtonEnabled corresponds to *Button* is enabled. *UserButton* corresponds to clicking of *Button* by the user. Some of the specification patterns are presented in [29, 71]. These specification patterns can be used in the verification. For each property, the following LTL formulas are generated.

LTL 1 $\Box\,(actionStartClicked \rightarrow\ \Diamond\ actionToRun)$

LTL 2 $\Box\,(actionPauseClicked \rightarrow\ \Diamond\ actionToPaused)$.

LTL 3 $\Box\,(backwardEnabled \wedge skipEnabled \wedge !startEnabled \wedge !playEnabled\ \wedge !pauseEnabled$ $\wedge\ !forwardEnabled \rightarrow stateEnd)$

LTL 4 1. $\Box\,((userStart \wedge\ !startEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

2. $\Box\,((userPause \wedge\,!pauseEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

3. $\Box\,((userPlay \wedge\,!playEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

4. $\Box\,((userForward \wedge\,!forwardEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

5. $\Box\,((userBackward \wedge\,!backwardEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

6. $\Box\,((userSkip \wedge\,!skipEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

7. $\Box\,((userQuit \wedge\,!quitEnabled) \rightarrow !eventHandled\ U\ userInterfaceIgnore)$

LTL 5 $\Box\,((stateInitial \vee stateEnd \vee statePaused \vee stateIdle) \rightarrow !pauseEnabled)$

LTL 6 $\Box\,((stateRun \vee stateForward \vee stateBackward \vee stateEnd) \rightarrow skipEnabled)$

LTL 7 $\Box\,(statePaused \rightarrow (playEnabled \wedge forwardEnabled \wedge backwardEnabled)$

LTL 8 $\Box\,((stateRun \vee stateEnd \vee statePaused \vee stateForward \vee stateBackward \vee stateInitial) \rightarrow !startEnabled)$

LTL 9 $\Box\,((stateInitial \vee stateEnd) \rightarrow !pauseEnabled)$

LTL 10 $\Box\,(startEnabled \wedge pauseEnabled \wedge playEnabled \wedge forwardEnabled \wedge backwardEnabled)$

A liveness property that should be checked whether the presentation reaches to its end once it starts.

Property 11 *The presentation will eventually end. (requires fairness constraints 1 and 2)*

LTL 11 $\Box\,(stateRun \rightarrow\ \Diamond\, stateEnd)$

There are also some properties that should be satisfied for streams. If a stream is in RunPoint state, the stream cannot be started by other streams. This is assumed to be a wrong attempt. So, a warning should be signaled to the author. In the same way, a stream cannot be terminated if it is already idle. The properties are as follows:

Property 12 *A stream can be started if it is already active. (undesirable, requires fairness constraints 1 and 2)*

Property 13 *A stream can be terminated if it is already idle. (undesirable, requires fairness constraints 1 and 2)*

The LTL formulas are:

LTL 12 $\Diamond\,(streamRunPoint\ U\ streamInitPoint)$

LTL 13 $\Diamond\,(streamIdlePoint\ U\ streamEndPoint)$

It needs to be verified that a stream will actually be played or not. Once it is ensured that streams will be played, further checks can be performed based on the relationships among streams. Based on Allen's temporal relationships, the following properties may be checked:

Property 14 *Stream A is before stream B. (requires fairness constraints 1 and 2)*

Property 15 *Stream A starts with stream B. (requires fairness constraints 1 and 2)*

Property 16 *Stream A ends with stream B. (requires fairness constraints 1 and 2)*

Property 17 *Stream A is equal to stream B. (requires fairness constraints 1 and 2)*

Property 18 *Stream B is during stream A. (requires fairness constraints 1 and 2)*

Property 19 *Stream B overlaps stream A. (requires fairness constraints 1 and 2)*

Let $P = streamA\_InitState$, $Q = streamA\_EndState$, $R = streamA\_IdleState$, $K = streamB\_InitState$, $L = streamB\_EndState$, $M = streamB\_IdleState$. The LTL formulas are as follows:

LTL 14 $(Q\,U\,((R \wedge M)\,U\,K))$

LTL 15 $\Diamond(P \wedge K)$

LTL 16 $\Diamond(Q \wedge L)$

LTL 17 $\Diamond(P \wedge K \wedge \Diamond (Q \wedge L))$

LTL 18 $\Diamond(P \wedge \Diamond(K \wedge \Diamond (Q \wedge \Diamond L))$

LTL 19 $!(\Diamond(Q \wedge \Diamond K) \vee \Diamond (L \wedge \Diamond Q))$

In [66], some properties between two consecutive user interactions based on time are verified. In a distributed system, these constraints cannot be satisfied due to delay of data. For example, pause operation for a stream may be performed within t seconds after the start of the presentation where $0 < t < d$ and d is the duration of the stream. In our model, user cannot change the state of a stream directly but he/she can change the state of the presentation thus changing the state of a stream indirectly. Since there are relationships among streams and containers, these can start and end each other. In our case, time is associated with actors. Since there is no delay in passing of time, the actor elapses its time right away once it is activated.

Since SPIN does not support time explicitly, time proceeds in the presentation as the streams proceed. In other words, if none of the streams can be played, it is assumed that time does not proceed. The streams update the time index as they start, end and are actually played. There are 21 (from 0 to 20) time indices for Figure 8.3.8. For example, stream V2 starts at time index 10, which corresponds to 9 seconds. A stream only updates the time index, if the current index at that instant is lower than its time index. Otherwise, it means that there is a delay in the play of the stream. For example, we would like to check V2 starts 3 seconds after V1 ends. The time index when V1 ends and V2 starts are t_1 and t_2. A $time$ array is kept to map time indices to real times. We need to check $(time[t_2] - time[t_1])$ when V2 starts. This is added as $timeCondition$ in the formulas.

Property 20 *Stream A is t seconds before stream B (requires fairness constraints 1 and 2).*

Property 21 *Stream B starts t seconds after stream A starts (requires fairness constraints 1 and 2).*

Property 22 *Stream B ends t seconds after stream A ends (requires fairness constraints 1 and 2).*

The corresponding LTL formulas are as follows:

LTL 20 $(Q\, U\, (R \wedge M)\, U\, (K \wedge timeCondition))$

LTL 21 $\Diamond(P \wedge (K \wedge timeCondition))$

LTL 22 $\Diamond(Q \wedge (L \wedge timeCondition))$

Note that delay may always occur. So it is meaningless to check that A2 starts 10 seconds after A1 ends. However, it is more meaningful to check that A2 starts 1 second after V3 starts since there is no delay in time.

One of the basic queries is whether all streams are played or not. If one of the streams is not played, this may be considered as an undesirable behavior and the author may correct it.

Property 23 *Stream A is played. (requires fairness constraints 1 and 2)*

LTL 23 $\Diamond(P \wedge \Diamond Q)$

For a multimedia presentation, the states of streams that are possible to visit in the backward presentation should also be reachable in the forward presentation. We call this property as *backward consistency* of a presentation and term such a presentation as *backward consistent* presentation. If we show the existence of a state that is not reachable in forward presentation while it is reachable in backward presentation, it is not backward consistent.

There are a couple of ways writing the LTL formula to check the backward consistency of a presentation. The major problem is the state that is reachable in the forward presentation may also be given (if exists) as a contradictory example. This complicates the verification since we also need to distinguish the states that are reachable in the forward presentation. Another problem is that the presentation may enter in an inconsistent state after backward operation and the desired state may be reachable in the forward presentation from that inconsistent state. So, the property is stated as two fold.

Property 24

1. *The state is not reachable in forward presentation (requires fairness constraints 1 and 2)*
2. *It is possible to reach the state in the backward presentation. (requires fairness constraint 2)*

Notice that first part requires the existence check. The corresponding LTL formulas are as follows:

LTL 24

1. $!\Diamond state$
2. $\Diamond state$

If the first part is right, then the second part is verified. The number of states that need to be checked is $\lfloor m^n \rfloor$ where m is the number of states that a stream may enter and n is the number of streams. Eventually, we need to convert the following property into the following one:

Property 25

1. *The state is not reachable in forward presentation (undesirable, requires fairness constraints 1 and 2)*
2. *It is possible to reach the state after user interactions. (requires fairness constraint 2)*

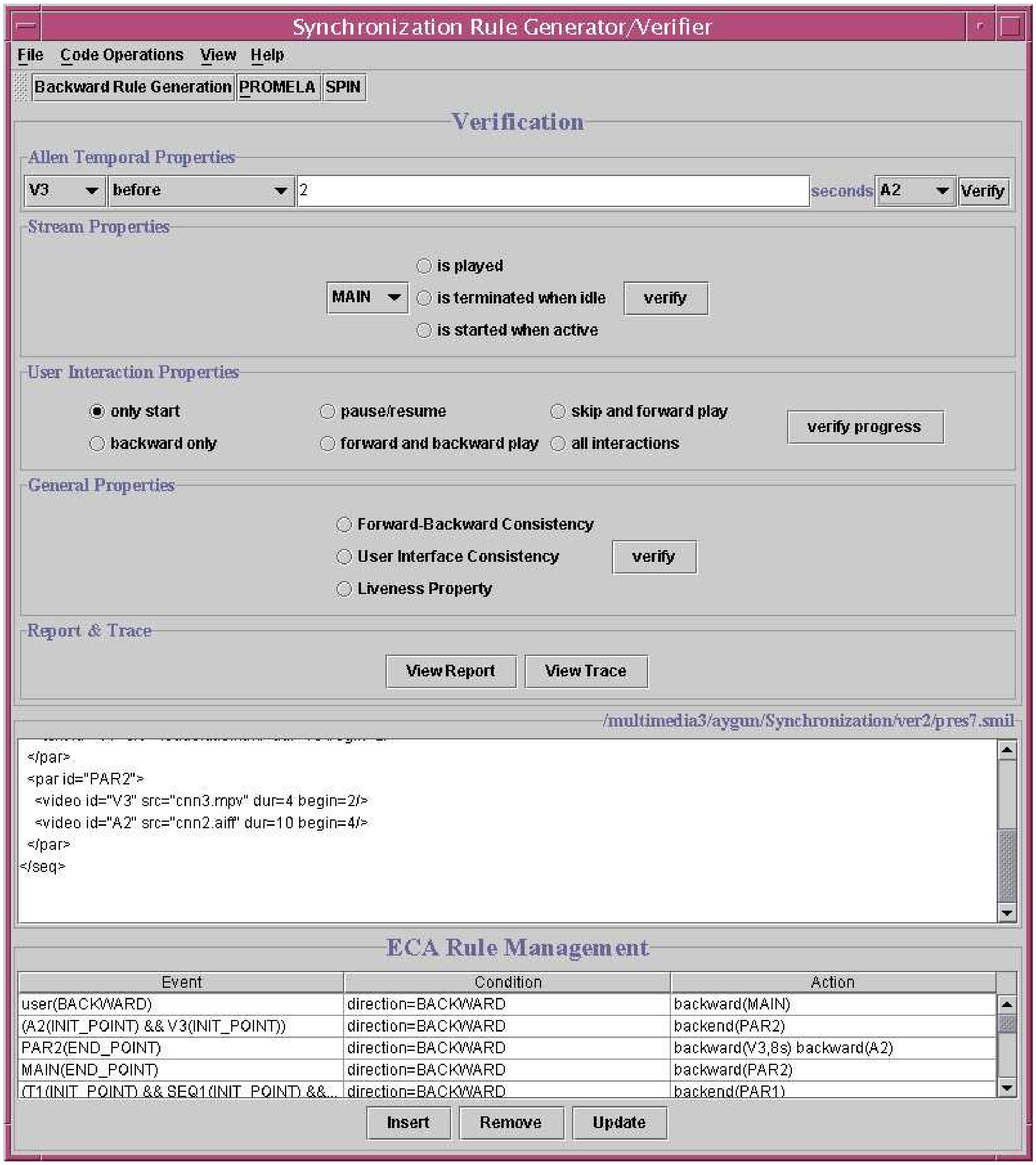
The previous LTL formula, in fact, corresponds to this property. Fig. 10.2.1 shows the user interface for verification.

Model checking is performed by generation of never claims from LTL properties. The system is checked at each state whether the undesired state occurs or not.

10.3 Experiments and Analysis

We have firstly developed a complex model to handle the user interactions. Since this user interface increases the number of initial states significantly, we removed the user interface during verification. Only buttons change their states as part of the user interface. The forward (fast) interaction is not allowed to reduce the complexity of the model since we are not interested in the speed of the presentation. We are rather interested in the direction change. The backward and play interactions are enough to verify the model.

Different kinds of presentations have been used to check the correctness of the presentation



Allen Temporal Properties: V3 before 2 seconds A2

Stream Properties: MAIN

User Interaction Properties: [✓ only start]

/multimedia3/aygun/Synchronization/ver2/pres7.smil

```
 </par>
 <par id="PAR2">
  <video id="V3" src="cnn3.mpv" dur=4 begin=2/>
  <video id="A2" src="cnn2.aiff" dur=10 begin=4/>
 </par>
</seq>
```

Event	Condition	Action
user(BACKWARD)	direction=BACKWARD	backward(MAIN)
(A2(INIT_POINT) && V3(INIT_POINT))	direction=BACKWARD	backend(PAR2)
PAR2(END_POINT)	direction=BACKWARD	backward(V3,8s) backward(A2)
MAIN(END_POINT)	direction=BACKWARD	backward(PAR2)
(T1(INIT_POINT) && SEQ1(INIT_POINT) &&...	direction=BACKWARD	backend(PAR1)

Figure 10.2.1: The user interface for verification.

Presentation	# Streams	Depth	States	Transitions	Memory (Mbyte)	Time (in seconds)
single	1	67	177	306	1.5	0.03
sequential	2	99	432	865	1.5	0.04
sequential	3	143	1021	2321	1.6	0.08
sequential	4	209	2347	5868	2.0	0.25
parallel	2	101	488	1021	1.5	0.05
parallel	3	139	1699	4642	1.8	0.13
parallel	4	173	6678	26132	2.8	0.76
synchronized	2	73	185	334	1.5	0.03
synchronized	3	78	201	398	1.5	0.05
synchronized	4	83	233	542	1.5	0.04

Table 10.3.1: Experiments without interaction.

model. We considered the number of streams and their organization. The streams are presented in a sequential order or in parallel. If the streams are presented in parallel, they may also be presented in a synchronized fashion.

The fairness constraints are hard-coded in the presentation. For each interaction, there is a fairness constraint and these are hard-coded in the model (lines 2,6,25). FC==3, FC==4, FC==5, FC==6 and FC==7 correspond to interactions where only start, only backward, pause-resume, skip, and backward-play are allowed, respectively.

We first investigated the complexity of the number of streams and the organization when no interaction (except to start the presentation) is allowed. The results are given in Table 10.3.1.

When a new stream is added into the sequential presentation, there are phases where the new stream starts, plays, and ends. The ending of stream does not add any complexity since they will all be idle at the end of the presentation. Since each stream adds 3 more phases, the number of states is nearly tripled after each addition of a stream in a sequential presentation. The complexity of number of states is $O(m^n)$ where n is the number streams in the sequential presentation and *m* is one less than the number of states that a stream may enter (to exclude idle state). In our experiments, *m* is 3. The running time and the depth also increase in the same way.

For a parallel presentation, there are more combinations of playing streams. In the parallel presentations, the streams may be interleaved. The number of possible interleavings for n streams that have m states is

$$I(n,m) = \frac{(2n)!}{(m!)^n}. \tag{10.3.1}$$

This explains the steep increase in running time, memory, states, transitions, and depth. Nevertheless, the running time is still within a second for 4 streams. The verification can be performed for a presentation having a fair number of parallel presentations. On the other hand, a synchronized parallel presentation's complexity is $O(n)$ for depth, transitions, and running time but $O(2^n)$ for the states.

To evaluate the effect of user interactions, we tested user interactions separately. The experiments with pause-resume interactions are given in Table 10.3.2. The pause-resume

Presentation	# Streams	Depth	States	Transitions	Memory (Mbyte)	Time (in seconds)
single	1	94	279	487	1.5	0.04
sequential	2	168	718	1435	1.6	0.06
sequential	3	304	1814	4060	1.8	0.12
sequential	4	559	4564	11341	2.5	0.36
parallel	2	178	829	1810	1.6	0.07
parallel	3	258	3519	11174	2.1	0.32
parallel	4	423	22913	101443	6.1	2.76
synchronized	2	106	287	517	1.5	0.03
synchronized	3	111	303	591	1.5	0.04
synchronized	4	122	335	749	1.5	0.06

Table 10.3.2: Experiments with Pause-Resume

interactions increase the complexity in linear time. Therefore, the presentations having pause and resume interactions do not add more complexity and this is an expected result. But these interactions increased the initial number of states, depth, and complexity.

The experiments with skip interaction are given in Table 10.3.3. The skip interaction increases the time complexity more than pause-resume due to possible number of skips at each interaction. The time complexity of synchronized presentations is $O(2^n)$. On the other hand, the complexity of states for parallel presentations increased from $O(4^n)$ to (10^n). The complexity of states for sequential presentations increased from $O(3^n)$ to $O(4^n)$.

The experiments with backward interaction are given in Table 10.3.4. The play interaction is allowed along with the backward interaction. The time complexity of synchronized presentations is $O(2^n)$. On the other hand, the complexity of states for parallel presentations increased from $O(4^n)$ to (10^n). The complexity of states for sequential presentations

Presentation	# Streams	Depth	States	Transitions	Memory	Time
sequential	1	156	4476	7219	2.0	0:00.17
sequential	2	380	20622	34946	4.8	0:00.81
sequential	3	1085	102309	179545	21.7	0:04.80
sequential	4	2436	438514	784559	100	0:24.01
parallel	2	347	26914	44746	5.6	0:00.98
parallel	3	677	233890	402438	43.8	0:09.96
parallel	4	1356	2.34 K	4.15 K	473	2:00.59
synchronized	2	166	5020	8179	2.2	0:00.20
synchronized	3	176	6108	10171	2.5	0:00.27
synchronized	4	186	8284	14299	3.0	0:00.38

Table 10.3.3: Experiments with Skip

Presentation	# Streams	Depth	States	Transitions	Memory	Time
sequential	1	216	8358	14898	2.6	0:00.31
sequential	2	561	34201	62691	7.1	0:01.45
sequential	3	1437	140408	260996	29	0:07.11
sequential	4	3157	596432	1.12 K	136	0:34.40
parallel	2	359	55780	101419	10	0:02.24
parallel	3	948	528264	978800	97	0:24.63
parallel	4	2031				
synchronized	2	228	9548	17254	2.9	0:00.39
synchronized	3	239	11912	22098	3.5	0:00.55
synchronized	4	250	16640	32154	4.7	0:00.81

Table 10.3.4: Experiments with Backward

increased from $O(3^n)$ to $O(4^n)$. These results show that the complexity of backward is similar to the skip. Since the direction of the presentation may change in the backward presentation, the number of initial states doubled and this caused severe exponential increase in the running time. To realize the effects of interactions, experiments where all interactions are allowed are conducted. The complexity of sequential, parallel, and synchronized presentations is similar to the backward and the skip.

Presentation	# Streams	Depth	States	Transitions	Memory	Time
sequential	1	745	24586	46364	4.8	0:00.92
sequential	2	1954	103197	200756	18.5	0:04.71
sequential	3	5717	424039	846276	85	0:23.13
sequential	4	18676	2.21 K	4.50 K	500	2:27.68
parallel	2	1509	184092	351747	31	0:07.87
parallel	3	3571	1.75 K	3.42 K	318	1:30.09
synchronized	2	823	30299	57461	6.1	0:01.30
synchronized	3	869	38179	74420	8.3	0:01.86
synchronized	4	871	53939	109319	12	0:02.75

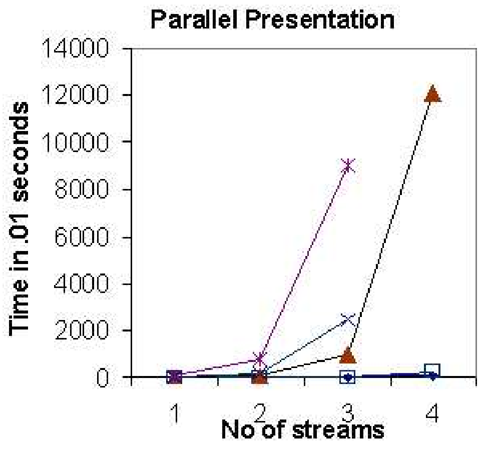
Table 10.3.5: Experiments with all interactions allowed

To realize the effects of interactions, experiments where all interactions are allowed are conducted. The complexity of sequential, parallel, and synchronized presentations is similar to the backward and the skip (Table 10.3.5).

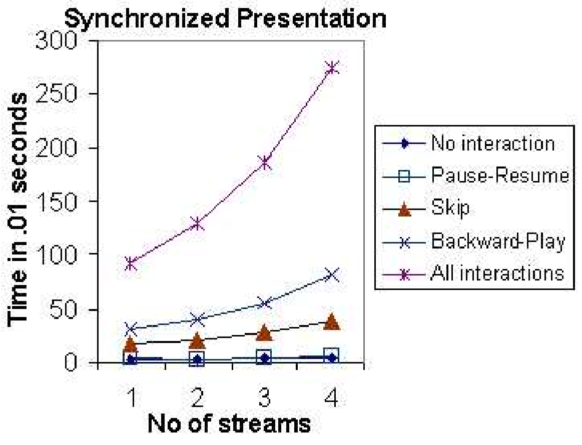
The effects of interactions on types of presentations are depicted in Figure 10.3.1.

10.3.1 Improving the Specifi cation of Multimedia Presentations

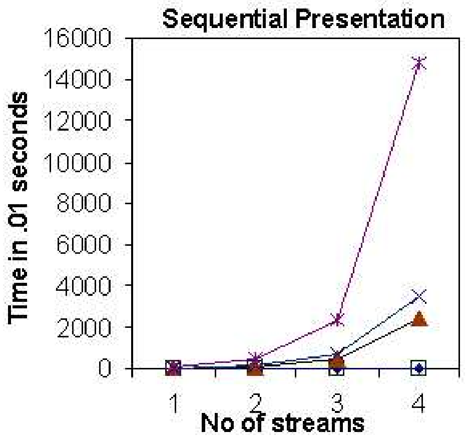
Different authors may provide different specification for the same presentation (in nominal presentation). Since there may be problems in the network, the satisfaction of synchronization requirements in real time is an important issue. The author is usually considered as the expert for the best specification. For example, the author requires that stream B should start after stream A. During the verification phase, the author realizes that stream B may start before end of A due to delay in the presentation of stream A. This case is presented by the



(a)



(b)



(c)

Figure 10.3.1: Elapsed time for verification of properties on (a) parallel, (b) synchronized, and (c) sequential presentations.

tool. The author realizes that there should be a dependency between stream A and stream B. If the system does not provide such information, the author will be unaware of the loss of synchronization. This information is not visible in the specification since the specification usually considers a perfect presentation. If in the specification their corresponding times are equivalent, it is assumed that the system will present at the corresponding times that may not be true. This model checking enforces the verification at the model and the author can be assured that the presentation will be presented correctly.

These verification results give an upper bound on the verification of properties. These results consider the non-progress cycles and other possible errors. Verification of properties takes less since only a specific condition is checked in the model. For the presentation given in Fig. 8.3.2, the verification for Allen's temporal properties takes less than 3 seconds.

10.3.2 Evaluation

The previous work on checking the integrity of multimedia presentations deals with presentations that are presented in nominal conditions (i.e., no delay). SPIN verifier takes into account each possible state that the processes and elements of a presentation may enter. Since the processes may iterate at different states as long as they are enabled, this introduces processes proceeding at different speeds. From the perspective of a multimedia presentation, this may correspond to delay of data in the network. The SPIN verifier checks the properties of a presentation also at the worst case. The unexpected false presentations are reported by contradictory examples.

SPIN enables verification of LTL formulas. LTL formulas require tracing all the execution paths. For example, it may be possible that two streams may start at the same time. What we are really interested is whether these two streams will eventually start at the same time in all occasions. The never-claims expressions provide the contradictory examples.

The detection of non-progress cycles when all the user interactions are allowed yields a general status of the presentation model. In reality, it is not possible to perform all the interactions at all possible occasions. During the initial modeling phases of our model, SPIN verifier detected a case that naturally is less likely to occur. In this case, the user starts the presentation and then clicks the BACKWARD button just before the presentation proceeds. This leads to an unexpected state where the presentation enters an infinite loop.

After the user starts a presentation and just before the presentation proceeds if the user attempts to backward the presentation, the presentation then enters an unexpected state and stays in this state forever.

10.4 Summary

The synchronization model is incorporated into the NetMedia [108] system, a middleware design strategy for streaming multimedia presentations in distributed environments. It is necessary whether the system will present a consistent presentation after the user interactions. In this chapter, we have showed a way of verifying multimedia presentations that also include backward and skip. Firstly, the synchronization model is developed to respond these functionalities. Then the user interactions are allowed and the specification is verified. SPIN's tracing of all possible states provides a way of modeling of delay for multimedia presentations.

This technique is better than testing and simulation since all the states that a model may enter is considered. The system may be verified whether it conforms to the specification. If a property is not satisfied, a counter example is provided by the SPIN tool. There have been methods proposed for temporal querying of presentations. But it is not tested whether the model really satisfies the author specification. The PROMELA code also supports delayed presentation. Thus it is possible whether the system satisfies the specification. In the chapter, we also considered satisfaction of Allen's temporal relationships.

Chapter 11

Conclusion and Future Work

In this dissertation, we have proposed solutions for effective spatio-temporal browsing of multimedia presentations. In the first part of the dissertation, methods for generating sprites in compressed domain and increasing resolution of sprites are covered. In the second half, the incorporation of model checking for multimedia presentations have been explained for a robust and flexible synchronization model that can handle user interactions that can also change the course of a presentation. As a result of these methods, we have developed two systems, *VideoCruise* (a spatial browser) and *RuleSync* (a robust and flexible synchronization model). These two systems are integrated into NetMedia system [108].

11.1 Conclusion

In this dissertation, we have started with methods that are necessary for effective spatial browsing. We have realized that most of the previous video is in compressed form using DCT blocks. We have proposed methods to generate effective features to be used in compressed domain for stationary background generation and video object segmentation. By only using DC coefficients of DCT blocks, we are able to generate the stationary background in compressed domain. We also have extracted boundary features from DCT compressed blocks. We have showed that DC coefficient, smoothness, boundary visibility, boundary type, and darkness are good features to determine significant blocks. The boundary features are used to eliminate the insignificant blocks for video processing.

Sprite is considered as a big picture of the environment. Since one goal is video compression to decrease the bandwidth, static sprite has to be generated from the sequence of frames in a video shot. The static sprite generation requires motion detection, image alignment, and residual estimation. Multiresolution sprite generation is significant if the camera has zoomed specific regions of the environment. We have presented a method for high resolution sprite generation from video. Motion estimation is performed between each consecutive frame not to miss visible areas in the sequence for sprite generation. Temporal integration and warping introduces blurring in sprite generation. The problem caused by temporal integration is reduced by histemporal filter. The blurring caused by warping is reduced by increasing the resolution (detail) of the sprite and warping at intervals. Although high resolution sprite warping increases elapsed time, this is compensated by warping at intervals. We introduced *conservative sprite* to reduce the blurring in the sprite.

We have also presented the sprite pyramid for videos and images having finite-depth scenes. In applications like distance learning, zoom-in and zoom-out are common camera operations. The original sprite is only appropriate for applications having no zooming. Traditional mosaicing techniques usually ignore these basic operations and cause blurred or very large mosaics. This problem is resolved by mapping the frames on a pyramid where layers show different resolution. More importantly, this sprite pyramid model allows the regeneration of the video frames and objects at the resolution they were captured.

Depending on the improvements and methods that are specified in the previous chapters, we have developed a spatial browser system, *VideoCruise*. The VideoCruise requires high quality sprite generation. High quality sprite generation can only be achieved by accurate global motion estimator. Our experiments show that average PSNR is not always a good indicator of quality by itself. PSNR does not consider blurring in the sprite. Sharpness measure is an indicator of blurring in the sprite. We obtained different sharpness measures for the same average PSNR. We demonstrated examples from standard MPEG test sequences. Once the motion vectors and the sprite are generated, VideoCruise provides interactive spatial browsing. VideoCruise provides panning, tilting and zooming interactions. VideoCruise allows the use to gain and release the camera control at any frame display. When the camera control is gained, all frames are mapped according to the frame which camera control is gained. In addition to browsing, it enables camera stabilization.

After discussing the contents of a spatial browser, we have introduced a robust and flexible synchronization model, *RuleSync*. The RuleSync synchronization model is developed to support the NetMedia [108] system, a middleware design strategy for streaming multimedia presentations in distributed environments. The synchronization is handled by synchronization rules based on event-condition-action (ECA) rules. The backward rules are generated automatically based on *author properties* and forward presentation. Not only forward temporal relationships are converted to reverse temporal relationships but also the relationships between events and actions are considered for backward presentation.

The model checking technique is used for verification of the model. This technique is better than testing and simulation since all the states that a model may enter is considered. The system may be verified whether it conforms to the specification. If a property is not satisfied, a counter example is provided by the SPIN tool. There have been methods proposed for temporal querying of presentations. But it is not tested whether the model really satisfies the author specification. We have also considered satisfaction of Allen's temporal relationships. During the real time presentation those constraints that are checked at the specification level may not be satisfied in a real time environment. The specification may in fact lead to false presentations due to delay and the synchronization model. This model enables the author to check whether the system really satisfies the requirements in the real life. Hence, the author may better specify the presentation using this model checking since extra-ordinary conditions are considered.

11.2 Future Work

Although we have achieved satisfactory results for spatio-temporal browsing, there are improvements that are necessary for further applications both in the spatial and temporal domain. In the spatial domain, below is a list of improvements that will be helpful.

- We have proposed a method to improve the results to generate the stationary background using motion vectors. The motion vectors need to be extracted from the compressed data and motion vectors depend on the performance of the MPEG encoder. Instead of extracting motion vectors, existence of motion at a macroblock can be utilized and evaluated within a group of pictures (GOP).
- If there is a temporal texture like wavy sea, fire and ocean in the scene, the generated sprite will only show an instance of the general picture. It is hard to use the sprite repeatedly. The temporal texture in the sprite can be modeled yielding a more robust

sprite. The object segmentation can increase content-based functionalities for further research.

- The boundary types that are covered are vertical, horizontal, and diagonal. Other types of boundaries like curved boundaries can also be detected using other AC coefficients but this is left as a further research. In those cases, the boundary type is represented with the index of the highest AC coefficient.

- If sprite pyramid representation is included in MPEG-4, it allows the regeneration of video objects at higher resolutions. There are two ways to incorporate this into MPEG-4: to consider each layer of the sprite pyramid as a separate sprite or to introduce sprite pyramid into MPEG-4. We will work on efficient incorporation of sprite pyramid into MPEG-4. In addition, more experiments will be conducted from other image and video resources like news and movies.

- One of the applications of VideoCruise is distance education application. The students will be able to follow the lecture as if they are in the classroom. The student can follow the instructor and the board. An index will be created for important objects in the environment. The user can follow the desired objects by just clicking on the indexed objects.

- The features of the VideoCruise will be enlarged to improve video editing. Sometimes, the camera is unstable as in ’foreman’ sequence and this can distract the audience. The camera can be stabilized by just clicking the camera gain control. The user will be able to save the document as he/she browses it or stabilize the camera. In sports games, sometimes the cameraman cannot track the fast moving objects properly.

We have developed a robust and flexible synchronization model and verified the correctness of the properties using model checking. Especially, features provided by model checking can be improved.

- There are also limitations that are put forward by the model checking. For example, it is not possible to check the *meet* relationship using SPIN. There is more than one state between ending a stream and starting a new stream. This is also true in real applications.

- The system should provide a better synchronization specification to satisfy the violated constraints.

- The PROMELA language does not provide time in the modeling. Thus it is not possible to incorporate time directly in the model. RT-SPIN enables the declaration of time constraints and checks acceptance cycles, non-progress cycles and some liveness properties. The first problem is some guards may be skipped due to lazy behavior of RT-SPIN. In our case, most of the time constraints are equality constraints. Also the interactions like pause, resume, skip, and backward require the guard condition to be updated after these interactions even when waiting for the guard condition to be satisfied. Other model checking tools need to be used and evaluated for multimedia synchronization.

Bibliography

[1] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of ACM*, 26(11):823–843, November 1983.

[2] Steele Arbeeny and Deborah Silver. Spatial navigation of video streams. In *ACM Multimedia 2001 Conference Proceedings*, pages 467–470, Ottawa, Canada, October 2001.

[3] F. Arman and *et al.* Image processing on compressed data for large video databases. In *Proc. ACM Int. Conf. Multimedia*, June 1993.

[4] R. S. Aygun and A. Zhang. Interactive multimedia presentation management in distributed multimedia systems. In *Proc. of Int.Conf. on Information Technology: Coding and Computing*, pages 275–279, Las Vegas, Nevada, April 2001.

[5] R. S. Aygun and A. Zhang. Middle-tier for multimedia synchronization. In *2001 ACM Multimedia Conference*, pages 471–474, Ottawa, Canada, October 2001.

[6] R. S. Aygun and A. Zhang. Stationary Background Generation in MPEG compressed Video Sequences. In *International Conference on Multimedia Expo*, pages 908–911, Tokyo, Japan, August 2001.

[7] R. S. Aygun and A. Zhang. Extracting Coarse Boundary Features for Video Processing. In *International Conference on Multimedia Expo*, Lausanne, Switzerland, August 2002.

[8] R. S. Aygun and A. Zhang. Global Motion Estimation from Semi-Dynamic Video using Motion Sensors. In *International Conference on Image Processing*, pages 273–276, Rochester, New York, September 2002.

[9] R. S. Aygun and A. Zhang. Management of backward-skip interactions using synchronization rules. In *The 6th World Conference on Integrated Design & Process Technology*, Pasadena, California, June 2002.

[10] R. S. Aygun and A. Zhang. Modeling and verification of interactive flexible multimedia presentations using promela/spin. In *The 9th International SPIN Workshop, LNCS 2318*, pages 205–212, Grenoble, France, April 2002.

[11] R. S. Aygun and A. Zhang. Reducing Blurring-Effect in High Resolution Mosaic Generation. In *International Conference on Multimedia Expo*, Lausanne, Switzerland, August 2002.

[12] Brian Bailey, Joseph Konstan, Robert Cooley, and Moses Dejong. Nsync - a toolkit for building interactive multimedia presentations. In *Proceedings of ACM Multimedia*, pages 257–266, September 1998.

[13] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *European Conference on Computer Vision*, pages 237–252, May 1992.

[14] Kiran Bhat, Mahesh Saptharishi, and Pradeep Khosla. Motion detection and segmentation using image mosaics. In *IEEE International Conference on Multimedia and Expo*, volume 3, pages 1577–1580, July 2000.

[15] T. Blaszka and R. Deriche. Recovering and characterizing image features using an efficint model based approach, November 1991. Technical Report 2422, INRIA.

[16] L. Bonnaud and C. Labit. Multiple occluding objects tracking using a non-redundant boundary-based representation for image sequence interpolation after decoding. In *IEEE International Conference on Image Processing, Volume I*, pages 426–429, October 1997.

[17] A. Bors and I. Pitas. Optical flow estimation and moving object segmentation based on median radial basis function network. *IEEE Transactions on Image Processing*, 7(5):693–702, May 1998.

[18] M. C. Buchanan and P. T. Zellweger. Scheduling multimedia documents using temporal constraints. In *Proceedings Third International Workshop on Network and Operating Systems Support for Digital audio and Video*, pages 237–249. IEEE Computer Society, November 1992.

[19] P. J. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.

[20] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):679–698, November 1986.

[21] R. Castagno, T. Ebrahimi, and M. Kunt. Video segmentation based on multiple features for interactive multimedia applications. *IEEE Transactions on Circuit and Systems for Video Technology*, 8(5):562–571, September 1998.

[22] CMIS. http://www.cis.ksu.edu/ robby/classes/spring1999/842/index.html.

[23] S. Coorg and S. Teller. Spherical mosaics with quaternions and dense correlation. *International Journal of Computer Vision*, 37(3):259–273, June 2000.

[24] J. P. Courtiat and R. C. De Oliveira. Proving temporal consistency in a new multimedia synchronization model. In *Proceedings of ACM Multimedia*, pages 141–152, November 1996.

[25] F. Dellaert and R. Collins. Fast image-based tracking by selective pixel integration. In *ICCV'99 Workshop on Frame-Rate Vision, September, 1999*, volume 1, September 1999.

[26] Y. Deng and B. S. Manjunath. Netra-v: Toward an object-based video representation. *IEEE Transactions on Circuit and Systems for Video Technology*, 8(5):616–627, September 1998.

[27] R. Deriche and T. Blaszka. Recovering and characterizing image features using an efficint model based approach. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 530–535, June 1993.

[28] F. Dufaux and J. Konrad. Efficient, robust, and fast global motion estimation for video coding. *IEEE Transactions on Image Processing*, 9(3):497–501, March 2000.

[29] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of 21st International Conference on Software Engineering*, May 1999.

[30] C. Eveland, K. Konolige, and R.C. Bolles. Background modeling for segmentation of video-rate stereo sequences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1998.

[31] R. Fablet, P. Bouthemy, and M. Gelgon. Moving object detection in color image sequences using region-level graph labeling. In *IEEE International Conference on Image Processing, ICIP'96*, pages 673–676, September 1996.

[32] R. Fablet, P. Bouthemy, and M. Gelgon. Moving object detection in color image sequences using region-level graph labeling. In *IEEE International Conference on Image Processing, ICIP'99*, October 1999.

[33] L. Garrido, P. Salembier, and J.R. Casas. Representing and retrieveing regions using binary partition trees. In *IEEE International Conference on Image Processing, ICIP'99*, October 1999.

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

[35] M. Gelgon and P. Bouthemy. Determining a structured spatio-temporal representation of video content for efficient visualization and indexing. In *European Conference on Computer Vision*, volume 1, pages 595–609, June 1998.

[36] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Data Modeling of Time-Based Media. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 91–102, Minneapolis, Minnesota, May 1994.

[37] R. Gonzales and R. Woods. *Digital Image Processing*. Addison-Wesley, 1993.

[38] G. Gordon, T. Darrell, M. Harville, and J. Woodfill. Background estimation and removal based on range and color. In *Proceedings of IEEE CVPR*, June 1999.

[39] J. Guo, J. Kim, and C.-C. J. Kuo. Fast and adaptive semantic object extraction from video. In *Proceedings of SPIE Image and Visual Communications Processing*, January 2000.

[40] R. Hamakawa and J. Rekimoto. Object composition and playback models for handling multimedia data. *Multimedia systems*, 2:26–35, 1994.

[41] Ivan Herman, Nuno Correira, David A. Duce, David J. Duke, Graham J. Reynolds, and James Van Loo. A standard model for multimedia synchronization: Premo synchronization objects. *Multimedia systems*, 6(2):88–101, 1998.

[42] S. Hibino and E. A. Rundensteiner. User interface evaluation of a direct manipulation temporal visual query language. In *ACM Multimedia'97 Conference Proceedings*, pages 99–107, Seattle, USA, November 1997.

[43] N. Hirzalla, B. Falchuk, and A. Karmouch. A Temporal Model for Interactive Multimedia Scenario. *IEEE Multimedia*, 2(3):24–31, 1995.

[44] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[45] G.J. Holzmann. *Design and Validation of Computer Protocols*. Englewood Cliffs, N.J.: Prentice Hall, 1991.

[46] W. Hurst and R. Muller. A synchronization model for recorded presentations and its relevance for information retrieval. In *Proceedings of ACM Multimedia*, pages 333–342, October 1999.

[47] M. Irani and P. Anandan. Video indexing based on mosaic representations. In *Proceedings of IEEE*, pages 905–921, May 1998.

[48] M. Irani, P. Anandan, J. Bergen, R. Kumar, and S. Hsu. Efficient representations of video sequences and their applications. *Signal Processing: Image Communication*, 8(4), 1996.

[49] M. Irani, S. Hsu, and P. Anandan. Video compression using mosaic representations. *Signal Processing: Image Communication*, 7(4-6), 1995.

[50] Murial Jourdan, Nabil Layaida, Cecile Roisin, Loay Sabry-Ismail, and Laurent Tardif. Madeus, an authoring environment for interactive multimedia documents. In *Proceedings of ACM Multimedia*, pages 267–272, September 1998.

[51] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Proc. of First International Conf. on Computer Vision*, pages 259–268, 1987.

[52] C. Kim and J-N. Hwang. An integrated scheme for object-based video abstraction. In *ACM*, pages 303–311, October 2000.

[53] Don Kimber, Jonathan Foote, and Surapong Lertsithicai. Flyabout: Spatially indexed panoramic video. In *ACM Multimedia 2001 Conference Proceedings*, pages 339–347, Ottawa, Canada, October 2001.

[54] S. Kruse, A. Graffunder, and S. Askar. A new tracking scheme for semiautomatic video object segmentation. In *Workshop on Image Analysis for Multimedia Application Services (WIAMIS'99)*, volume 2419, pages 93–96, June 1999.

[55] T. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Arears in Communications*, 8(3):413–427, April 1990.

[56] T. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):551–563, 1993.

[57] Yan Lu, Wen Gao, and Feng Wu. Fast and robust sprite generation for mpeg-4 video coding. In *The second IEEE Pacific-Rim conference on Multimedia (PCM)*, pages 118–125, October 2001.

[58] B. Marcotegui, F. Zanoguera, P. Correia, R. Rosa, F. Marques, R. Mech, and M. Wollborn. A video object generation tool allowing friendly user interaction. In *IEEE International Conference on Image Processing, ICIP'99*, October 1999.

[59] S. Marcus and V.S. Subrahmanian. Foundations of multimedia database systems. *Journal of the ACM*, 43(3):474–523, 1996.

[60] K. E. Matthews and N. M. Namazi. A bayes decision test for detecting uncovered background and moving pixels in image sequences. *IEEE Transactions on Image Processing*, 7(5):720–728, May 1998.

[61] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *Proceedings ACM SIGMOD Conference on Management of Data*, pages 215–224, 1989.

[62] R. Mech and M. Wollborn. A noise robust method for segmentation of moving objects in video sequences. In *ICASSP97*, pages 2657–2660, April 1997.

[63] R. Mech and M. Wollborn. A noise robust method for 2-d shape estimation of moving objects in video sequences considereing a moving camera. *EURASIP, Signal Processing*, 66(2):203–218, 1998.

[64] T. Meier and K.N. Ngan. Automatic video sequence segmentation using object tracking. 1998.

[65] T. Meier and K.N. Ngan. Video object plane segmentation using a morphological filter and hausdorff object tracking. In *ICIP*, October 1998.

[66] I. Mirbel, B. Pernici, T. Sellis, S. Tserkezoglou, and M. Vazirgiannis. Checking temporal integrity of interactive multimedia documents. *VLDB Journal*, 9(2):111–130, 2000.

[67] D.M. Monro, H. Li, and J.A. Nicholls. Object based video with progressive foreground. In *IEEE International Conference on Image Processing, ICIP'97*, 1997.

[68] Jongho Nang and Sujin Kang. A new multimedia synchronization specification method for temporal and spatial events. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 236–243. IEEE Computer Society, June 1997.

[69] A. Neri, S. Colonnese, G. Russo, and P. Talone. Automatic moving object and background separation. *Signal Processing*, 66:119–232, 1998.

[70] C.W. Ngo, T. C. Pong, and R. T. Chin. Exploiting image indexing techniques in dct domain. *Pattern Recognition*, 2000.

[71] Dimitrie O. Paun and Marsha Chechik. Events in linear-time properties. In *Proceedings of 4th International symposium on Requirements Engineering*, June 1999.

[72] Shmuel Peleg, Benny Rousso, Alex Rav-Acha, and Assaf Zomet. Mosaicing on adaptive methods. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(10):1144–1154, October 2000.

[73] B. Prabhakaran and S. Raghavan. Synchronization Models for Multimedia Presentation with User Participation. *Multimedia Systems*, 2(2), 1994.

[74] A. Puri and T. Chen. *Multimedia Systems, Standards, and Networks*. Signal Prpcessing and Communications Series, 2000.

[75] Paul Pzandak and Jaideep Srivastava. Interactive multi-user multimedia environments on the internet: An overview of damsel and its implementation. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 287–290. IEEE Computer Society, June 1996.

[76] H. Richter, A. Smolic, B. Stabernack, and E. Mller. Real time global motion estimation for an mpeg-4 video encoder. In *Proc. PCS'2001, Picture Coding Symposium*, April 2001.

[77] C. Ridder, O. Munkelt, and H. Kirchner. Adaptive background estimation and foreground detection using kalman-filtering. In *Proceedings of Interenational Conference on Recent Advances in Mechatronics*, pages 193–199, June 1995.

[78] P. L. Rosin and T. Ellis. Image difference threshold strategies and shadow detection. In *British Machine Vision Conference*, pages 347–356, 1995.

[79] Yong Rui, Liwei He, Anoop Gupta, and Qiong Liu. Building an intelligent camera management system. In *ACM Multimedia 2001 Conference Proceedings*, pages 2–11, Ottawa, Canada, October 2001.

[80] P. Salembier and F. Marques. Region-based representations of image and video: Segmentation tools for multimedia services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1147–1167, December 1999.

[81] P. Salembier, O. Pujol, and L.Garrido. Connected operators for sprite creation and layered representation of image sequences. In *IV European Signal Processing Conference*, pages 2105–2108, September 1998.

[82] H.S. Sawhney and Rakesh Kumar. True multi-image alignment and its application to mosaicking and lens distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3), March 1999.

[83] J. Schnepf, J. Konstan, and D. Du. FLIPS: Flexible Interactice Presentation Synchronization. *IEEE Selected Areas of Communication*, 14(1):114–125, 1996.

[84] B. Shahraray. Scene change detection and content-based sampling of video. In *Proceedings IST/SPIE*, volume 2419, February 1995.

[85] Bo Shen and Ishwar K. Sethi. Direct feature extraction from compressed images. In *Proc. SPIE Storage, Retrieval for Image and and Video Databases IV*, volume 2670, 1996.

[86] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*. Seattle, June 1994.

[87] T. Sikora. The mpeg-4 video standard verification model. *IEEE Trans. Circuits Syst. Video Technology*, 7:19–31, February 1997.

[88] SMIL. http://www.w3.org/AudioVideo.

[89] A. Smolic and J.-R. Ohm. Robust global motion estimation using a simplified m-estimator approach. In *Proc. ICIP2000, IEEE International Conference on Image Processing*, September 2000.

[90] A. Smolic, T. Sikora, and J.-R. Ohm. Long-term global motion estimation and its application for sprite coding, content description and segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1227–1242, December 1999.

[91] A. Smolic and T. Wiegand. High-resolution video mosaicing. In *Proc. ICIP2001, IEEE International Conference on Image Processing*, October 2001.

[92] Y. Song, M. Mielke, and A. Zhang. NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments. In *IEEE International Conference on Multimedia Computing and Systems*, pages 585–590, Italy, Florence, June 1999.

[93] J. Stauder, R. Mech, and J. Ostermann. Pfinder: Real-time tracking of the human body. *IEEE Multimedia*, 1(1):65–76, March 1999.

[94] E. Steinbach, P. Eisert, and B. Girod. Motion-based analysis and segmentation of image sequences using 3-d scene models. *Signal Processing: Special Issue Video Sequence Segmentation for Content-based Processing and Manipulation*, 66(2):233–248, 1998.

[95] O. Sukmarg and K.R. Rao. Fast object detection and segmentation in mpeg compressed domain. In *International Conference on Signal Processing and Applications and Technology*, 2000.

[96] Xinding Sun, Jonathan Foote, Don Kimber, and B.S. Manjunath. Panoramic video capturing and compressed domain virtual camera control. In *ACM Multimedia 2001 Conference Proceedings*, pages 329–338, Ottawa, Canada, October 2001.

[97] R. Szeliski and H-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Computer Graphics Proceedings, Annual Conference Series*, pages 251–258, 1997.

[98] R. Szewcyk, A. Ferencz, H. Andrews, and B.C. Smith. Motion and feature-based video metamorphosis. In *Proceedings of ACM Multimedia*, 1997.

[99] Michael Vazirgiannis, Yannis Theodoridis, and Timos Sellis. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems*, 6(4):284–298, 1998.

[100] D. Wang. Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Transactions on Circuit and Systems for Video Technology*, 8(5):539–646, September 1998.

[101] J. Wang and Z. Li. Kernel-based multiple cue algorithm for object segmentation. In *SPIE Symposium on Electronic Imaging 2000, Image and Video Communications and Processing, SPIE Proceedings*, volume 3974, pages 462–473, 2000.

[102] J. Weber and J. Malik. Rigid body segmentation and shape description from dense optical flow under weak perspective. In *Proceedings of the 5th International Conference on Computer Vision*, 1995.

[103] Andrew Wilson, Ming C. Lin, Dinesh Manocha, Boon-Lock Yeo, and Minerva Yeung. A video-based rendering acceleration algorithm for interactive walkthroughs. In *ACM Multimedia 2000 Conference Proceedings*, pages 75–84, Los Angeles, USA, November 2000.

[104] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1998.

[105] B.-L. Yeo and S.-F. Chang. Rapid scene analysis on compressed video. *IEEE Trans. Circuits Syst. Video Technology*, 5(6):533–544, December 1995.

[106] K-J. Yoon and I-S. Kweon. Moving object segmentation algorithm for human-like vision system. In *1st International Workshop on Human-friendly Welfare Robotic Systems*, 2000.

[107] Kyoungro Yoon and P. Bruce Berra. Topcn: Interactive temporal model for interactive multimedia documents. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 136–144. IEEE Computer Society, June 1998.

[108] A. Zhang, Y. Song, and M. Mielke. *NetMedia*: Streaming Multimedia Presentations in Distributed Environments. *IEEE Multimedia*, 9(1):56–73, 2002.

[109] H. Zhang, A. Kankahalli, and S. Smoliar. Automatic partitioning of full-motion video. *ACM/Springer Multimedia Systems*, 1(1):10–28, 1993.

[110] I. Zoghlami, O. Faugeras, and R. Deriche. Using geometric corners to build a 2d mosaic from a set of images. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 420–425, 1997.