UNITED STATES PATENT AND TRADEMARK OFFICE

---

BEFORE THE PATENT TRIAL AND APPEAL BOARD

---

Canon Inc., Canon U.S.A., Inc., and Axis Communications AB,

Petitioner

v.

Avigilon Fortress Corporation,

Patent Owner

---

Case: Unassigned

U.S. Patent No. 7,932,923
Issue Date: April 26, 2011

Title: Video Surveillance System Employing Video Primitives

---

# DECLARATION OF EMILY R. FLORIO

I, Emily R. Florio, state and declare as follows:

1. I have prepared this Declaration in connection with the Petitions of Axis Communications AB, Canon Inc., and Canon U.S.A., Inc. (collectively "Petitioner") for two *inter partes* reviews of U.S. Patent No. 7,932,923 ("the '923 patent"), which I understand will be filed concurrently with this Declaration. I also understand that October 1999 is a date that is relevant for determining what is prior art to the '923 patent.

2. I am currently the Director of Research & Information Services at Finnegan, Henderson, Farabow, Garrett & Dunner LLP, 901 New York Avenue NW, Washington, DC 20001-4413.

3. I am over 18 years of age and am competent to make this Declaration. I make this Declaration based on my own personal knowledge, based on my knowledge of library science practices, as well as my knowledge of the practices at the Massachusetts Institute of Technology ("MIT") Libraries.

4. I earned a Master's of Library Science ("MLS") from Simmons College in 2006, and I have worked as a librarian for over a decade. I have been employed in the Research & Information Services (formerly Library) Department of Finnegan since 2013, and from 2005-2013, I worked in the Library Department of Fish & Richardson P.C.

5. I am currently the Vice-President Elect of the American Association of Law Libraries and the President of the Law Librarians' Society of Washington, DC, and a member of the International Legal Technology Association.

**Attachments**

6. Attached as Exhibit A (Exhibit 1003 to the Petition) is a true and correct copy of "Visual Memory," May 1993, pp. 1-92, by Christopher James Kellogg ("*Kellogg*"), obtained from the MIT Libraries.

7. Attached as Exhibit B is a true and correct copy of the "Standard" record from the MIT Libraries' catalog system (known as the Barton Catalog) for its copy of *Kellogg*.

8. Attached as Exhibit C is a true and correct copy of the MARC record of the MIT Libraries for its copy of *Kellogg*.

9. Attached as Exhibit D (Exhibit 1004 to the Petition) is a true and correct copy of F. Brill et al., "Event Recognition and Reliability Improvements for the Autonomous Video Surveillance System," Proceedings of the Image Understanding Workshop, Monterey, CA, Nov. 20-23, 1998, Vol. 1, pp. 267-283 ("*Brill*"), obtained from the Duderstadt Center, formerly known as the University of Michigan Media Union (UMMU).

10. Attached as Exhibit E is a true and correct copy of the MARC record of the University of Virginia Library for its copy of *Brill*.

11. Attached as Exhibit F is a true and correct copy of the MARC record of the North Carolina State University library for its copy of *Brill*.

12. Attached as Exhibit G (Exhibit 1006 to the Petition) is a true and correct copy of N. Dimitrova et al., "Motion Recovery for Video Content Classification," ACM Transactions on Information Systems, October 1995, Vol. 13, No. 4, pp. 408-439 ("*Dimitrova*"), obtained from University of California Los Angeles Science & Engineering Library.

13. Attached as Exhibit H is a true and correct copy of *Dimitrova*, obtained from the Library of Congress.

14. Attached as Exhibit I is a true and correct copy of the MARC record of the MIT Libraries for its copy of the ACM Transactions on Information Systems journal, in which *Dimitrova* was published.

15. Attached as Exhibit J is a true and accurate copy of B. Flinchbaugh et al., "Autonomous Video Surveillance," SPIE Proceedings, 25th AIPR Workshop: Emerging Applications of Computer Vision, Feb. 26, 1997, Vol. 2962, p. 144-151 ("*Flinchbaugh*"), obtained from the MIT Libraries.

16. Attached as Exhibit K is a true and correct copy of the MARC record of the Library of Congress for its copy of the SPIE Proceedings publication that includes *Flinchbaugh*.

17. Attached as Exhibit L is a true and correct copy of the MARC record of the MIT Libraries for its copy of the SPIE Proceedings publication that includes *Flinchbaugh*.

**The MARC Cataloging System**

18. The MAchine-Readable Cataloging ("MARC") system is used by libraries to catalog materials. The MARC system was developed in the 1960s to standardize bibliographic records so they could be read by computers and shared among libraries. By the mid-1970's, MARC had become the international standard for bibliographic data, and it is still used today.

19. Each field in a MARC record provides information about the cataloged item. MARC uses a simple three-digit numeric code (from 001-999) to identify each field in the record.

20. For example, field 245 lists the title of the work and field 260 lists publisher information. In addition, field 008 provides the date the item was cataloged. The first six characters of the field 008 are always in the "YYMMDD" format.

21. It is standard library practice that once an item is cataloged using the MARC system, it is shelved. This process may take a relatively nominal amount of time (i.e., a few days or weeks). During the time between the cataloging and

shelving of an item, the public may still find the item by searching the catalog and requesting the item from the library.

***Kellogg***

22. As indicated in Exhibit A (Exhibit 1003 to the Petition), *Kellogg* has an MIT Libraries date stamp of "JUL 09 1993" on page 1, indicating that the MIT Libraries received *Kellogg* on July 9, 1993. Further, as indicated in Exhibit B, the Standard record of the Barton Catalog confirms that *Kellogg* is shelved at the MIT Libraries and was published in 1993. In view of the above and the following, *Kellogg* was published and accessible to the public in 1993, years before October 1999.

23. As indicated in Exhibit C, *Kellogg* has a cataloging date of September 28, 1993 (shown as "930928" in field 008). This confirms that *Kellogg* was entered into the OCLC database, in which MIT does its cataloging, on September 28, 1993. This is also consistent with its noted year of publication in the MARC record (shown as "1993" in field 260). The OCLC database (also referred to as "WorldCat") is the largest online public access catalog (OPAC) in the world.

24. Soon after *Kellogg* received a cataloging date, a record of its existence would have appeared in and been keyword-searchable through the Barton Catalog of the MIT Libraries. The Barton Catalog is currently available online to any user of the World Wide Web. Before it was accessible by Web (i.e., at the time the

*Kellogg* thesis was received by the MIT Libraries in July 1993), it would have been accessible to anyone on the MIT campus *and* anyone who had access to the OCLC database.

25. During the time period from September 1993 through October 1999, the Barton Catalog allowed keyword searching for words in the thesis title, and *Kellogg* would have appeared in a relevant Barton Catalog search conducted on or shortly after September 28, 1993.

26. After being cataloged, a document such as *Kellogg* will undergo a process of being labeled and then shelved at the MIT Libraries. Based on my knowledge of MIT Libraries' current and prior practices, *Kellogg* would have been shelved in a relatively nominal amount of time (i.e., a few days or weeks). Thus, *Kellogg* was cataloged and shelved at the MIT Libraries at least before the end of 1993.

27. Once shelved, *Kellogg* can be borrowed by any member of the MIT community. Furthermore, a copy of *Kellogg* can be purchased from MIT by any member of the public. Indeed, the first page of *Kellogg* confirms that there were no restrictions placed on its publication, as it states that "[t]he author hereby grants to MIT permission *to reproduce and to distribute copies of this thesis document* in whole or in part, and to grant others the right to do so."

28. Further evidence of the public availability of *Kellogg* before October 1999 is provided in Exhibit J, which is a copy of *Flinchbaugh.* In its Bibliography, *Flinchbaugh* cites to *Kellogg* (reference [4] on p. 151). As addressed below, *Flinchbaugh* was published in SPIE Volume 2962, which corresponds to the Proceedings from the 25th Annual AIPR Workshop on Emerging Applications of Computer Vision. The Workshop was held October 16-18, 1996, and the Proceedings were published by at least 1997. Thus, *Kellogg* was at least available to members of the public in 1997, as shown by its citation in *Flinchbaugh.*

29. For the avoidance of any doubt, I note that on June 23, 2001, *Kellogg* was also cataloged in the MIT Archive Noncirculating Collection 1, Noncirculating Collection 3, and in microfiche form in the Barker Library, as indicated in the three entries for PST8 and in the second, third, and fourth instances of field 008 on page 1 of Exhibit C. However, none of this alters the fact that *Kellogg* was published and accessible to the public in 1993, as indicated above.

***Brill***

30. As indicated in Exhibit D, *Brill* is part of the published Proceedings of the 1998 Image Understanding Workshop. The Workshop was held in Monterey, California during November 20-23, 1998, and the Proceedings were "APPROVED FOR PUBLIC RELEASE" with "DISTRIBUTION UNLIMITED." Ex. D at 1. In

view of the above and the following, the Proceedings, including *Brill,* was published and accessible to the public before October 1999.

31. Evidence of *Brill's* publication and availability to the public includes the hand-written receipt date of "8-13-99" at the top of page 3 of Exhibit D. This indicates it was received by the UMMU (the University of Michigan Media Union, now known as the Duderstadt Center) on August 13, 1999. In my experience as a librarian and knowledge of standard library practices, the hand-written information at the top of p. 2 of Exhibit D appears to be the catalog record information for *Brill.* Based on standard library practices, this reference would have been shelved shortly after being received and cataloged by UMMU.

32. Further evidence of the publication and accessibility of *Brill* to the public can be found in Exhibit E, which is the MARC record for the Proceedings, including *Brill*, that was obtained from the University of Virginia Library. As shown in field 008 near the top of page 2 of Exhibit E, *Brill* was cataloged by the library on December 15, 1998. Based on standard library practices, this reference would have been shelved shortly after (i.e., within a few days or weeks) and been accessible to the public prior to October 1999.

33. Further evidence of the publication and public availability of *Brill* can be found in Exhibit F, which is the MARC record for the Proceedings, including *Brill*, that was obtained from North Carolina State University. As shown in field

008 on page 1 of Exhibit F, *Brill* was cataloged by the library on December 15, 1998. Based on standard library practices, this reference would have been shelved shortly after (i.e., within a few days or weeks) and been accessible to the public prior to October 1999.

***Dimitrova***

34. As indicated in Exhibit G, *Dimitrova* was published in a special issue of the ACM Transactions on Information Systems Journal. Ex. G at 1. In view of the above and the following, the ACM Journal, including *Dimitrova*, was published and accessible to the public before October 1999.

35. Evidence of *Dimitrova's* publication and availability to the public includes the "November 17, 1995" date stamp on page 1 of Exhibit G. This confirms that *Dimitrova* was received by the University of California Los Angeles Science & Engineering Library on November 17, 1995. Based on standard library practices, this reference would have been shelved shortly after (i.e., within a few days or weeks) and accessible to the public before October 1999.

36. Further evidence of the publication and accessibility of *Dimitrova* to the public is found in Exhibit H, which is a copy of *Dimitrova* obtained from the Library of Congress. Page 3 of Exhibit H bears the date stamp of "November 21, 1995" from the Library of Congress. This confirms that *Dimitrova* was received by Library of Congress on November 21, 1995. Based on standard library

practices, this reference would have been shelved shortly after (i.e., within a few days or weeks) and accessible to the public before October 1999.

37. Further evidence of the publication and accessibility of *Dimitrova* to the public can be found in Exhibit I, which is the MARC record of the ACM Journal, which includes Dimitrova, obtained from the MIT Libraries. As shown in field 008 near the top of page 1 of Exhibit I, the MIT Libraries began receiving the ACM Journal in September 1989.

38. Field 362 (shown as "3620") on page 1 of Exhibit I indicates that the MIT Libraries has issues dating back to Volume 7 of the journal, which as noted above was published in September 1989. There is no end-date in field 362, indicating that the MIT Libraries has an ongoing subscription, which would have included receipt of the issue that contained *Dimitrova* in 1995. Based on standard library practices, the issue containing *Dimitrova* would have been shelved shortly after cataloging (i.e., within a few days or weeks) and accessible to the public before October 1999.

39. For the avoidance of any doubt, I note that it appears that on June 23, 2001, online access to *Dimitrova* was provided to certain MIT-associated individuals, as indicated by the field 008 on the last line of page 1, field 8528, and the URL entry at the top of page 2 of Exhibit I. Also on June 23, 2001, the ACM Journal, including *Dimitrova*, was archived at the MIT Library Storage Annex

("LSA"), as indicated by the 008 field and subsequent 85271 entry on page 2 of Exhibit I. However, none of this alters the fact that *Dimitrova* was published and accessible to the public years before October 1999, as indicated above.

***<u>Flinchbaugh</u>***

40. As indicated in Exhibit J, *Flinchbaugh* was published in the Proceedings of the 25th AIPR Workshop: Emerging Applications of Computer Vision, SPIE Vol. 2962. The Workshop was held in Washington, D.C. during October 16-18, 1996, and the Proceedings was published by SPIE (The International Society for Optical Engineering). Ex. J at 1. In view of the above and the following, *Flinchbaugh* was published and accessible to the public before October 1999.

41. Page 2 of Exhibit J shows a copyright date of 1997. The edition of the SPIE Proceedings that was published with *Flinchbaugh* is Volume 2962, and it was "Printed in the United States of America." Ex. J at 2.

42. Although the copyright date of *Flinchbaugh* is listed as 1997, it appears that *Flinchbaugh* was actually published before that, in 1996. First, as noted above, the Workshop was held in Washington, D.C. during October 16-18, 1996. Second, a copy of *Flinchbaugh* was received and cataloged by the Library of Congress in November 1996. *See* Ex. K at 1. Exhibit K is the MARC record for the SPIE Proceedings, including *Flinchbaugh*, that was obtained from the

Library of Congress. As shown in field 008 near the top of page 2 of Exhibit K, *Flinchbaugh* was cataloged by the library on November 21, 1996. Based on standard library practices, this reference would have been shelved shortly after it was cataloged (i.e., within a few days or weeks). Collectively, Exhibits J and K show that *Flinchbaugh* was published and accessible to the public years before October 1999.

43. Further evidence of the publication and public availability of *Flinchbaugh* can be found in Exhibit L, which is the MARC record for the SPIE Proceedings, including *Flinchbaugh*, that was obtained from the MIT Libraries. As shown in field 008 on page 1 of Exhibit L, *Flinchbaugh* was cataloged by the library on March 10, 1997. Based on standard library practices and my understanding of the practices of the MIT Libraries, this reference would have been shelved shortly after it was cataloged (i.e., within a few days or weeks) and accessible to the public before October 1999.

44. For the avoidance of any doubt, I note that on April 8, 2011, online access to *Flinchbaugh* was provided to certain MIT-associated individuals, as indicated by the fields 008 and 8528 and the URL entry at the top of page 2 of Exhibit L. Also, on June 23, 2001, the SPIE Proceedings, including *Flinchbaugh*, was archived at the MIT Library Storage Annex ("LSA"), as indicated by the second 008 field and subsequent 8520 entry on page 2 of Exhibit L. However,

none of this alters the fact that *Flinchbaugh* was published and accessible to the public years before October 1999, as indicated above.

I have been warned and understand that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. § 1001). I declare that all statements made in this declaration of my own knowledge are true and all statements made on information and belief are believed to be true.

I declare under penalty of perjury that the foregoing is true and correct. Executed on July 9, 2019 in Washington, D.C.

[illegible]

Emily R. Florio

# Visual Memory

by

Christopher James Kellogg

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science
and
Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1993

Signature redacted

Author ..........................................................................

Department of Electrical Engineering and Computer Science
April 21, 1993

Signature redacted

Certified by ..........................................................................

Alex P. Pentland
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Signature redacted

Certified by ..........................................................................

Bruce E. Flinchbaugh
Manager of Image Understanding Branch at Texas Instruments
Thesis Supervisor

Signature redacted

Accepted by ..........................................................................

Campbell L. Searle
Chairperson, Departmental Committee on Graduate Students

# Visual Memory

by

Christopher James Kellogg

Submitted to the Department of Electrical Engineering and Computer Science
on April 21, 1993, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Science in Computer Science

## Abstract

Visual memory supports computer vision applications by efficiently storing and retrieving spatiotemporal information. It is a unique combination of databases, spatial representation and indexing, and temporal representation and indexing. This thesis designs a visual memory architecture that meets the requirements of a number of computer vision applications. It also presents an implementation of part of this design in support of a scene monitoring prototype.

Thesis Supervisor: Alex P. Pentland
Title: Associate Professor of Media Arts and Sciences

Thesis Supervisor: Bruce E. Flinchbaugh
Title: Manager of Image Understanding Branch at Texas Instruments

## Acknowledgements

My primary thanks goes to my two thesis supervisors, Bruce Flinchbaugh at Texas Instruments and Sandy Pentland at MIT. Bruce pointed me to the visual memory project that he was starting and guided my research at Texas Instruments. Sandy provided useful feedback throughout the research stage. They were both very helpful in critiquing the thesis document.

I'd also like to thank the other people at Texas Instruments who helped me with this project. Steve Ford and Tom Bannon were especially helpful in developing the visual memory design. In addition, I don't think I would have survived the bugs in PC++ without Steve's expertise. Tom Bannon and Tom O'Donnell provided a nice tracking system with which to test the visual memory prototype.

Finally, I'd like to thank my family, Fred, Jeannette, and Mark Kellogg, my fiancée Christine Bailey, and my brothers at Phi Kappa Sigma for their support throughout my MIT career.

# Contents

# List of Figures

# Chapter 1

# Introduction

Visual memory supports computer vision applications by efficiently storing and retrieving spatiotemporal information. It is a unique combination of databases, spatial representation and indexing, and temporal representation and indexing. Visual memory provides representational flexibility and high-performance information access to meet the requirements of a variety of computer vision applications.

## 1.1 Needs for Visual Memory

Applications use spatiotemporal data in many different ways and place many different demands on a visual memory. Studying possible uses helps to clarify the concept of a visual memory and to identify the functionality it provides.

Visual memory could serve as the repository for static information, such as object descriptions, maps, and environment models, that applications reference during execution. For example, a vehicle navigator could store maps and images to help it later recognize its location. A large amount of such information could be established prior to application execution, and the visual memory would subsequently provide an application with efficient access to desired pieces of information.

An application could store dynamic information in the visual memory. For example, a vehicle navigator's input systems could maintain in the visual memory a description of the vehicle's local environment, updating it as the vehicle moved. The visual memory could provide the navigator's planning processes with information about the vehicle's latest state and could analyze its progress to help determine a course of action. The high performance of the visual memory allows it to handle the frequent updates and queries needed by such dynamic, real-time systems.

Visual memory could manipulate spatiotemporal information about objects and collections of objects too large to fit into volatile memory. For example, a computer-aided design and modeling system could use the visual memory in building up a large design layout and simulating its execution over time; a photo interpretation system could similarly construct in the visual memory a complex representation of a scene. The visual memory would retrieve into main memory only a manageable part of a large representation at a time.

Visual memory could act as the interface between inputs and applications in a computer vision system. For example, computer vision algorithms for a security system could analyze data provided by various cameras and store information in the visual memory. Applications could then retrieve this data to track objects, watch for suspicious events, and respond to user queries. The visual memory would coordinate the information from its inputs and eliminate the need for full connectivity between inputs and applications.

Finally, visual memory could serve as a means for data transfer. A computer vision application could store spatiotemporal information in the visual memory for other applications to retrieve at any time in the future. To run comparative studies, different algorithms could use common data stored in the visual memory.

## 1.2 Goals

This thesis explores visual memory design and implementation. The primary goal of the thesis is to design a visual memory architecture that meets the requirements of various computer vision applications. A secondary goal is to implement a visual memory prototype to support a real-time scene monitoring prototype.

# Chapter 2

# Background

Visual memory builds on research in database design, spatial representation and indexing, and temporal representation and indexing. While there has been significant research in each of these areas, no previous project has combined them in this manner. The visual memory design uses knowledge gained from research projects in all these areas. This chapter summarizes and discusses some especially relevant projects.

## 2.1 Database Research

Visual memory must address concerns that a great deal of database research has already investigated. It must provide everything from information storage techniques to concurrency control for multiple inputs and outputs. Visual memory should build on the results of research into these topics. Presented here are two databases that address a number of the issues important to visual memory and that could be the basis for a visual memory system.

### 2.1.1 DARPA Open OODB

The DARPA Open Object-Oriented Database (Open OODB) project at Texas Instruments outlines an extensible architecture that allows "...tailoring database functionality for particular applications in the framework of an incrementally improvable system ...." [25] The architecture meets functional requirements such as an object data model and concurrent access, along with "meta requirements" including openness and reusability. The open architecture lets separate modules handle extensions to the basic storage mechanism. These extensions cover standard database issues such as transactions, versions, and queries.

The Open OODB architecture is very suitable for visual memory. The object-oriented model can flexibly and intuitively represent the information used by computer vision applications. Following the Open OODB architecture, visual memory could avoid confronting standard database issues by letting other modules support those features. Instead, visual memory would consist only of those extensions necessary to support efficient manipulation of spatiotemporal information. If new features were needed, extra modules could easily be added to the architecture.

### 2.1.2 POSTGRES

The POSTGRES database [23] expands the relational database model to meet the needs of complex applications. Because it builds on traditional relational databases, it provides a number of standard features, such as transactions, a query language, and recovery processing. In addition, it allows applications to specify new data types, operators, and access methods. POSTGRES supports active databases and rules, letting applications set up daemons in the database that react to changes in the data. A versioning mechanism keeps track of old data and works with the query language to let applications retrieve this information. Finally, the POSTGRES storage server can "vacuum" old data onto archival media.

POSTGRES supplies many features useful to a visual memory, such as transactions, queries, and application-defined access methods. However, the relational model might not be sufficiently expressive to meet the representational needs of complex computer vision applications. In addition, the POSTGRES design does not support application-specific extensions to the database, so it would be hard for the visual memory to expand to meet future requirements.

## 2.2 Spatial Research

There are many ways to describe spatial objects and to handle their storage and retrieval. Visual memory must consider how well different spatial models meet the representational needs of computer vision applications and how efficiently information in these models can be stored and retrieved.

### 2.2.1 CODGER

Researchers at Carnegie Mellon University developed the CODGER (COmmunications Database with GEometric Reasoning) "whiteboard" database and communication system to support the autonomous NAVLAB vehicle [20]. CODGER stores data to be communicated among the various modules that control vehicle navigation. It represents this information as tokens consisting of attributes and values.

CODGER uses a fairly simple spatial model. Token attributes represent basic spatial information such as position and object extent. The tokens support some standard geometric operations like area calculation. A query mechanism can answer some spatial queries like the proximity query "Return the tokens with location within 5 units of (45,32)." CODGER does not provide an indexing mechanism, and spatial operations and queries are performed in memory.

### 2.2.2 Core Knowledge System

The Core Knowledge System (CKS) [24], developed at SRI International, stores information for a robot. Like CODGER, it encodes this information as attribute-value tokens. CKS introduces special support for the uncertainty that results from inconsistent or incomplete information provided to the database. Its query mechanism includes keywords such as *apparently* and *possibly* to discern multiple opinions. Since spatial information is often imprecise, this support for uncertainty would be very useful in a visual memory context. However, CKS does not provide any special spatial operations or query constructs.

### 2.2.3 ISR

The ISR project at the University of Massachusetts at Amherst [3] defines a spatial representation (the Intermediate Symbolic Representation) and a management system for accessing data represented this way. The intermediate symbolic representation includes tokens for basic spatial objects such as lines, regions, and sets of parallel lines, but not for higher-level spatial objects such as people and vehicles. The data management system manipulates these tokens in an efficient manner. Applications built with ISR perform classification and in-memory spatial indexing.

### 2.2.4 Image Understanding Environments

The Image Understanding Environments (IUE) program [16] specifies a spatial representation to meet the needs of a wide variety of computer vision applications. An IUE spatial object is defined by a set of points; this point set can be concrete (a list of all the points) or abstract (an equation defining the points in the object). IUE spatial objects are manipulated through set operations – complex objects can be constructed through conjunction and disjunction of point sets. In addition to its point set, each spatial object also defines a bounding box, a centroid, and other attributes for different, and perhaps more efficient, methods of spatial manipulation. The IUE specification only briefly discusses data transfer and does not provide database support for storage and retrieval of spatial information.

### 2.2.5 PROBE

The PROBE database [15], developed at the Computer Corporation of America, extends an object-oriented database management system to meet the requirements of a variety of computer vision applications. It implements a number of spatial data types and supports operations on sets of points. It outlines a query language with some support for spatial queries. To provide more efficient spatial access, it also provides what the authors call *approximate geometry*, a limited form of spatial indexing.

### 2.2.6 Spatial Indices

A large number of spatial data structures can provide efficient access to spatial information. Samet [18] describes a number of these, including quadtrees, hash tables, grid files, range trees, and R trees. Each index is specialized for specific storage and retrieval characteristics; visual memory would benefit from including a number of different indices to efficiently manipulate data for different applications.

## 2.3 Temporal Research

Databases manipulate two different types of time: *transaction time*, specifying when updates for events are stored in the database, and *valid time*, specifying when events actually happen. *Rollback databases* implement transaction time, *historical databases* implement valid time, and *temporal databases* implement both. Sometimes historical and rollback databases are informally called temporal databases to indicate their concern with time. Since the computer vision applications discussed in the Introduction are concerned with the times at which events happen, visual memory should be a historical database.

A number of different historical and temporal databases represent and store temporal information. Each addresses a different set of concerns, and some designs suit visual memory better than others. The following research projects address many of the issues that visual memory must consider.

### 2.3.1 TQuel

The temporal database TQuel [21] is a temporal extension to a relational database. TQuel associates with each database record the slots *valid-from* and *valid-to*, defining an interval during which the record is valid. For example, the Employees relation might have three records for Frank, one valid from 0 to 1/1/93, another valid from 1/1/93 to 5/7/93, and a third valid from 5/7/93 to $\infty$. If Frank were changed on 8/7/93, then the third record's *valid-to* slot would be changed to 8/7/93, and a new record valid from 8/7/93 to $\infty$ would be added.

TQuel extends the query language Quel [12] to support temporal access of records. A temporal query specifies an interval of interest; the database retrieves any record whose valid interval overlaps that interval. A query can also ask for records before, after, or as of a given moment. TQuel provides operators such as *overlaps* and *extend* to form complex query intervals.

### 2.3.2 Temporal Sequences

The temporal database outlined in [19] models object state changes with temporal sequences. A temporal sequence can be discrete or continuous; for example, sales per month could be modeled as a discrete temporal sequence, while the voltage in alternating current could be modeled as a continuous temporal sequence. A temporal sequence is always represented by a set of state snapshots; interpolating functions estimate continuous sequences. Characteristics such as granularity and regularity of state snapshots define each temporal sequence. Functions including selection, aggregation, and accumulation operate on sets of time sequences. The database also includes a powerful SQL-like [1] query language for retrieving temporal sequences.

### 2.3.3 Temporal Sets

Researchers at the University of Houston proposed some temporal additions [8] to the Extended Entity-Relationship Model. The basic temporal representation in this temporal model is a finite union of time intervals; for example, a particular state could be valid during the set of time intervals { [50,60), [90,230), [231,239) }. The database stores with each object a temporal element denoting its valid time. Basing temporal representation on sets of intervals preserves closure under set operations and provides a standard means for manipulating temporal information and querying the database.

This model was later augmented to better represent temporal uncertainty [13]. The extended model preserves the definition of a temporal element but modifies the definition of a temporal interval. Each endpoint in an interval specifies a valid time method that returns an ordered set of time points. The endpoint belongs to this set, but in order to allow for uncertainty, it is not explicitly specified. The model also modifies the standard set operations to manipulate uncertain temporal elements.

### 2.3.4 Relative Time

Some applications, such as computer-aided design systems, know how events are ordered but not the actual times of the events. Chaudhuri [5] proposes a temporal model to handle these cases. This model represents time as a graph rather than as a time line. Events are ordered with binary relations like *before* and *simultaneously*. These relations must obey properties such as transitivity and antisymmetry so that the database can navigate through a graph and infer additional relationships. The model supports temporal queries about event relations; for example, a query could ask for a lower time bound on an event or for common ancestors of two events. This capability could be useful in a visual memory to support efficient handling of temporal information for some applications.

### 2.3.5 Temporal Indices

Much of the spatial indexing research also applies to temporal indexing. For example, interval trees can store intervals in space or in time. To handle more complex, specialized temporal representations, however, requires additional research. Some of the databases described above provide their own temporal indices; [22] references many other systems with temporal indices.

# Chapter 3

# Design

This chapter presents a design for a visual memory system. It examines requirements and considerations that the design must take into account. It discusses key visual memory topics such as representation and indexing of spatial, temporal and spatiotemporal information. This chapter outlines a concrete, implementable system; the next chapter presents the prototype implementation of this design.

## 3.1 Requirements and Considerations

The design of a visual memory must address a number of concerns. Some of these come from anticipated uses of the visual memory, while others are common themes in spatial, temporal, and database research. This section covers a number of these requirements and considerations.

### 3.1.1 Database Considerations

One database issue relevant to visual memory is how to represent and store information. There are several standard models, including the relational model, the entity-relationship model, and the object-oriented model. The visual memory should use an object-oriented model to meet the broad representational requirements of a variety of applications. An object-oriented approach is intuitive and highly extensible, allowing applications to define new, complex objects at any time.

Another important consideration is concurrency control. The visual memory must be able to handle multiple, dynamic inputs and outputs. For example, in a scene-monitoring system, many different cameras could update the visual memory simultaneously. The visual memory must ensure data consistency.

Much database research involves well-defined program interfaces, including explicit storage mechanisms and query algebras. Applications using the visual memory do not need to know how it achieves its results, but they should know what results to expect. For example, performance-enhancing measures such as indexing and caching do not affect the objects returned by a query and can be added without affecting the query algebra.

Recoverability is another database issue important to some visual memory applications. The visual memory must work to guarantee that, even in the case of a system crash, it does not lose stored information. In addition, it must be able to remove inconsistencies resulting from system failure during information storage.

### 3.1.2 Spatial and Temporal Considerations

The purpose of visual memory is to store information about the history of a visual environment. Visual memory is not just a generic database — it must have spatiotemporal concerns at the heart of its design.

A visual memory must provide representational flexibility. Rather than forcing one spatiotemporal representation on all applications, the visual memory should be tailorable to an application's needs. Applications can trade off between representational power and performance.

A visual memory must handle dynamic objects. Some computer vision applications need to update spatial information in response to changes in the environment. The visual memory must define spatiotemporal representations to effectively handle such changes. It must provide a versioning mechanism to store and retrieve different state snapshots of objects.

A visual memory must provide a flexible, expressive query mechanism with extensive spatiotemporal support. This query mechanism should support a wide variety of spatiotemporal queries. For example, a security system might ask the visual memory to retrace a person's path over the past five minutes, a vehicle navigator might ask it to watch for objects entering the field of view, and a CAD system might ask for simulation results for everything electrically connected to a specific chip. The visual memory should let applications conveniently express such queries.

### 3.1.3 Performance Considerations

High performance is one of the key requirements for a visual memory. Some visual memory applications, such as a vehicle navigator, need to store and retrieve information very quickly. Many spatial and temporal models in the literature are very expressive but do not provide the necessary information throughput. A visual memory must be both expressive and fast enough to meet the demands of its applications.

Indexing can help a visual memory achieve high performance by quickly identifying objects satisfying given constraints. Visual memory indices should be *conservative*, never mistakenly omitting objects that satisfy a query. In this manner, indices can improve query performance but are guaranteed to not affect the results.

A visual memory must provide a variety of indices to meet the needs of different applications. For example, a real-time scene monitoring system could set up an index to track the centroids of moving objects, while a photo interpretation system could index the areas covered by objects. A visual memory indexing mechanism should be extensible, handling additional application-defined indices.

A visual memory must let applications control which objects are indexed. For example, an application could establish one index on all objects, another index on everything in the current session, and yet another index only on certain objects of interest. This would prevent the visual memory from wasting time and space updating unimportant indexing information.

Caching and look-ahead techniques can increase the performance of a visual memory. Caching improves storage performance by not requiring the visual memory to wait for information to be written to disk. Both caching and look-ahead improve retrieval performance by reducing the number of disk accesses.

Visual memory performance can be increased by letting applications tailor the visual memory to their specific requirements. For example, some applications can afford to lose a small amount of data, so they could eliminate recoverability information. Other applications could optimize specific storage and retrieval cases; for example, a vehicle navigator could optimize its real-time performance by sacrificing some historical performance.

## 3.2 Design Overview

The visual memory design consists of a set of extensions to an open database architecture like DARPA Open OODB [25]. An open architecture allows the visual memory to add spatiotemporal customizations to the database. The visual memory can take full advantage of other modules implementing features such as concurrency control, caching, and versioning, without having to handle these capabilities directly.

The visual memory design follows the object-oriented model discussed in the previous section. A class hierarchy defines representations for spatiotemporal information. Abstract superclasses define the interfaces for manipulating spatiotemporal information, and their subclasses extend the definitions to represent more specific types of objects. This document denotes classes in italics; for example, *SpatialObject* is the class representing spatial objects. A concrete member of this class is referred to as "a *SpatialObject* instance" or informally just as "a spatial object."

The visual memory design specifies a number of classes for representing spatiotemporal information. These classes provide methods through which computer vision applications and the visual memory can manipulate them. For example, the spatial class *Square* could include a method to return its area, the temporal class *TemporalInterval* could have a method to determine its duration, and the spatiotemporal class *Person* could implement a method plotting its space-time trajectory. Applications can design their own classes inheriting from these classes and extending them to meet additional needs.

The visual memory design extends the database's storage mechanism. It provides a mechanism for object identity and maintains a history for each object. Each version of an object specifies when it was valid, and the visual memory can manipulate versions based on valid time. The design lets applications customize the database storage server based on characteristics of the data they typically store.

The visual memory design extends the database's query mechanism to provide spatiotemporal support. The additional spatiotemporal constructs allow computer vision applications to flexibly and expressively specify objects of interest.

To achieve suitable query performance, the visual memory provides spatiotemporal indices that can efficiently identify objects satisfying query conditions. A visual memory index is an object that maintains information about other objects, allowing it to efficiently indicate those objects that meet certain constraints. For example, a visual memory spatial index might store object centroids so that it can quickly identify all the objects within a specified area. The visual memory provides a powerful and flexible indexing mechanism.

## 3.3 Spatial Representations

The visual memory spatial class hierarchy provides a powerful framework that allows applications flexibility in designing spatial representations while ensuring that the visual memory can access the information it requires. The class hierarchy draws on the research outlined in the Background chapter. It provides the basic framework for any visual memory application, and it allows applications to extend it to meet additional needs.

Spatial operations are often complex and require much computation. Spatial indices, described in Section 3.8, can increase the performance of these operations by maintaining information about sets of spatial objects. This chapter presents a number of spatial operations; Section 3.8 describes related performance issues.

### 3.3.1 Core Spatial Classes

#### SpatialObject

The *SpatialObject* class is the basis for all high-level spatial representations. Possible subclasses derived from *SpatialObject* include *Cube*, *QuestionMark*, and *Person*, depicted in Figure 3-1. *SpatialObject* captures the common representational requirements of a variety of such spatial objects. It provides a standard set of slots and methods to yield a consistent spatial interface. Applications can design additional spatial representations as long as they provide the same functionality.

A spatial object is defined by a set of points and a local coordinate system. This information is sufficient to fully represent a spatial object. The point set specifies what area of space the object fills. The coordinate system relates these points to the points in other spatial objects. Additional information, such as centroid, orientation, and bounding box, is derivable from this information.

*SpatialObject* provides a wide variety of methods to manipulate its data. These methods can translate and rotate an object, operate on its point set, and find the object's bounding box, among other things. Most of these are actually point set and coordinate system functions and will be discussed further below. Concrete spatial

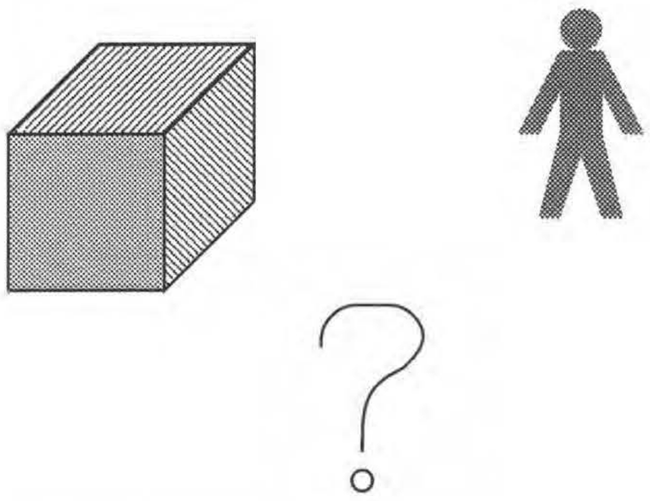


Figure 3-1: Spatial objects

objects can provide additional relevant information; for example, a cube could have functions returning the length of its side, its surface area, and its volume. Using the methods of *SpatialObject* and its subclasses, an application can manipulate spatial data in many different ways.

Several lower-level classes manipulate information for the high-level *SpatialObject* class. The following sections present these classes.

### Point

The most elementary unit of spatial representation is the point. The visual memory provides the abstract class *Point* and subclasses *TwoDPoint*, *ThreeDPoint*, etc., to represent this elementary unit. *Point* is a fairly simple class, only storing and manipulating a coordinate in some space. As will be shown below, however, it is an important building block.

### PointSet

Complex spatial information can be represented as a collection of points, or point set. The visual memory provides the class *PointSet*, derived from a generic *Set* class,

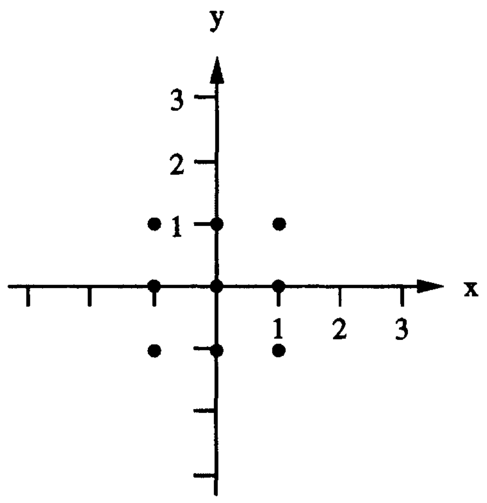


Figure 3-2: Discrete point set

to store and manipulate sets of points. Since *PointSet* is a kind of *Set*, it provides standard set operations, such as union, disjunction, member, and difference. This allows a powerful means for constructing complex objects. It also furnishes a well-defined and sound mathematical basis for spatial representation and manipulation.

The class *DiscretePointSet* represents a set of points simply as an exhaustive list of all desired points. This representation is feasible only for small point sets. For example, consider the task of representing the square area of the points plotted in Figure 3-2. A system could, by convention, represent a square area by such a discrete set of points. Standard set operations can easily manipulate this information. Unfortunately, the space required for this representation grows too quickly to be broadly applicable.

The class *AbstractPointSet* is a far more efficient means for representing large or even infinite point sets. It abandons an exhaustive list of all points in favor of a functional definition of the points in the set. An abstract point set specifies a function that returns TRUE for points in the set and FALSE for points not in the set. For example, the function for the continuous square in Figure 3-3 would check for $-1 <= x, y <= 1$. This fully represents the square area. A point set's representation function grows complex as the set is modified by operations such as conjunction and

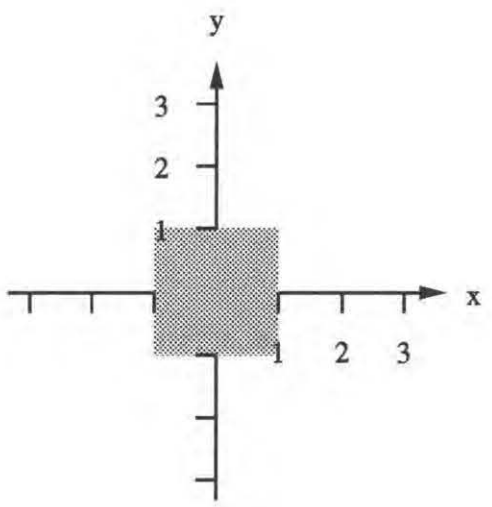


Figure 3-3: Abstract point set

disjunction, but a suitably complex function can represent any desired set of points.

Some visual memory applications start with discrete approximations to the continuous world but want to interpolate to a continuous description. For example, an application might recognize only the list of discrete points above that make up just a small part of an actual continuous square. For these applications, a subclass of *AbstractPointSet*, *InterpolatingAbstractPointSet*, can apply a specified interpolation function to that list of points to derive a continuous function. For example, an interpolation of the point set in Figure 3-2 could yield the point set in Figure 3-3.

An instance of *PointSet* is more than just a set of points; it also includes a number of methods deriving spatial information from this set. Important methods find a point set's centroid, boundary, bounding box, and surface normal, among other things. These methods extend the power of the point set and enhance the visual memory spatial support.

### CoordinateSystem

A point in space is useful only in relation to other points. The *CoordinateSystem* class establishes relationships between points in the visual memory. Figure 3-4 shows a couple of possible coordinate systems. Each *CoordinateSystem* subclass must define

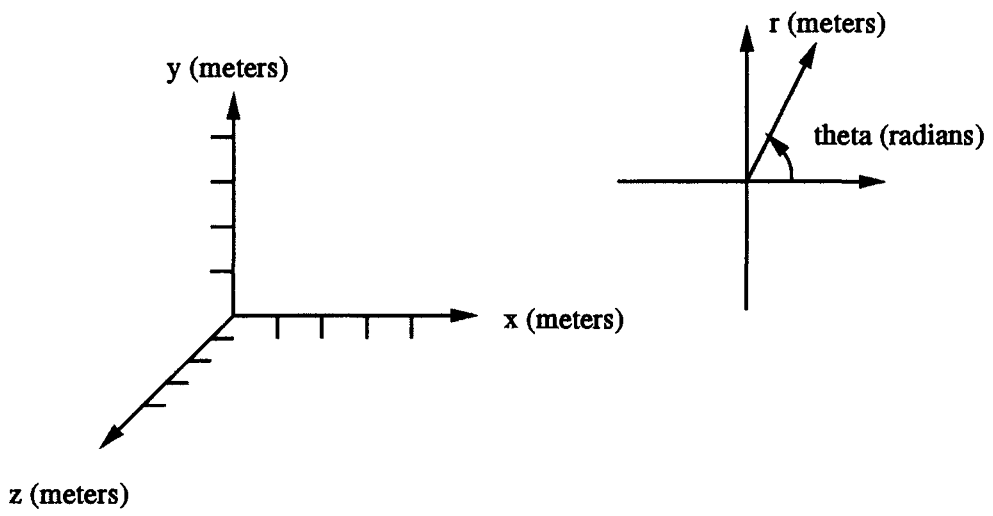


Figure 3-4: Coordinate systems

dimensions, axes, and other features of the space. These specifications give meaning to points and provide the basis for relating points. The visual memory defines a coordinate system for a set of points; the *SpatialObject* class associates a local coordinate system with each point set.

The main job of a coordinate system is to relate points. To do this, it maintains a list of coordinate transforms between it and other coordinate systems. To achieve high run-time speed efficiency, a coordinate system can maintain transforms between it and several other coordinate systems. Alternatively, it can trade off speed of operation for lower space requirements by storing only a few transforms and letting the visual memory follow a chain of transforms among related coordinate systems.

To reduce the cost of multiple transforms, an application can adopt a unified coordinate system to relate a number of nearby local coordinate systems. This unified coordinate system would maintain transforms to and from each local coordinate system. In this manner each coordinate system would not need to keep a large list of transforms, and only two transforms would be needed to relate points in one coordinate system to those in any other. A limitation of this approach is that it does not scale well for large distances, because the error induced by each transform could compound significantly.

The *CoordinateSystem* class provides methods to transform a coordinate system's relationship with other coordinate systems. For example, one coordinate system might translate and rotate with respect to others. Transforming a coordinate system modifies its list of coordinate transforms, and all coordinate transforms between it and other systems must be updated. This is automatically provided by the visual memory as part of the transformation method.

Transforms like translation and rotation are *CoordinateSystem* methods rather than *PointSet* methods for a number of reasons. The coordinate system relates the point set to other coordinate systems, and it is probably more efficient to store a transform than a transformed point set for each other coordinate system. It is also more efficient to accumulate a set of transforms into one transform than to repeatedly apply transforms to a whole set of points. If the points are represented by a function, it could be hard to determine how the transform should modify that function. The transform could be applied only when needed; if it were used repeatedly, the results could be cached.

Coordinate system transforms permit the construction of multiple-object scenes. Each spatial object is developed in its local coordinate system, and then coordinate system transforms construct relations between local coordinate systems. The opposite effect occurs when multiple sets of points in one coordinate system are split into separate spatial objects with local coordinate systems. In this case, the transformation from each local coordinate system to the original unified coordinate system is already defined. Standard computer graphics texts, such as [11], discuss coordinate system transforms and the construction of multiple-object scenes in further detail.

### 3.3.2 Relative Spatial Specification

In many cases, a coordinate system has explicitly-defined relationships to other coordinate systems. For example, one coordinate system might have an origin 3 units to the east of the origin of another coordinate system. In other instances, however, this information is not so clear. For example, an application might only need to know

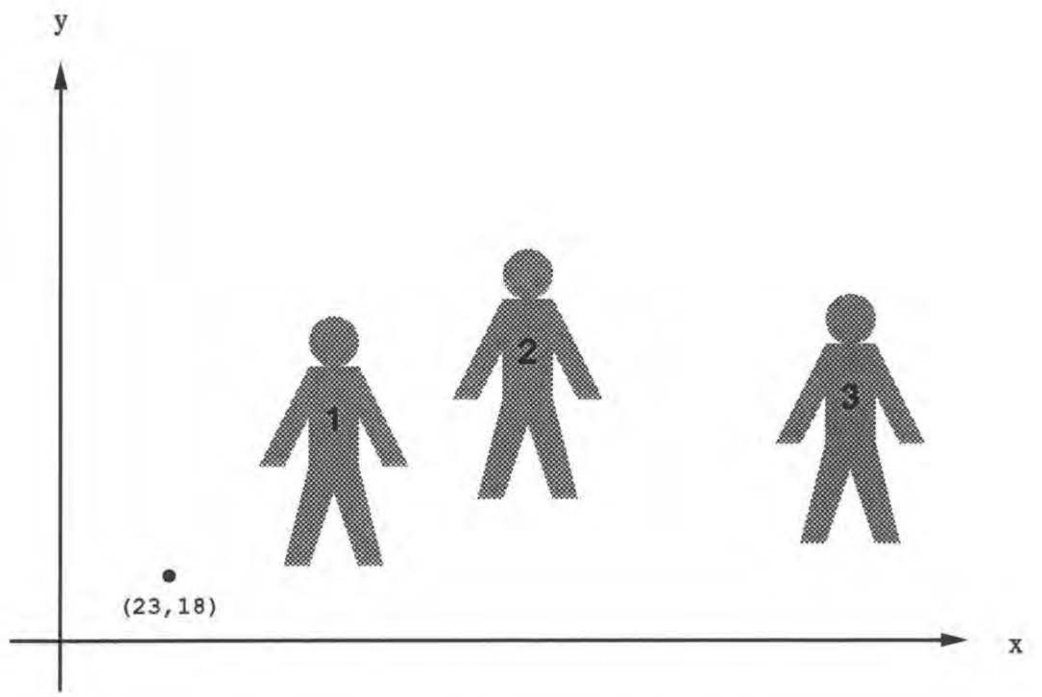


Figure 3-5: Relative spatial objects

that one block was to the east of another block, without knowing an explicit distance. In these cases a relative spatial specification is required.

There are actually two kinds of relative spatial specification: specification relative to a concrete position or object with concrete position, and specification relative to another relative spatial specification. For example, in Figure 3-5 the visual memory knows that object 1 is to the east of the point (23,18), object 2 is to the east of object 1, and object 3 is to the east of object 2. This description does not precisely specify the scene; for example, object 2 could be further to the north and east and still meet the specification.

The visual memory provides the class *RelativeSpatialObject*, a subclass of *SpatialObject*, to handle relative spatial specification. A relative spatial object simply keeps lists of objects relative to it in various ways. For example, one subclass of *RelativeSpatialObject* might provide lists for objects west, east, north, and south of it, while another subclass might provide a list for nearby objects, where "near" is defined by a method of the class. *RelativeSpatialObject* can represent both kinds of relative spatial specification mentioned above, since an instance can be defined relative to any spatial object, including a fixed position or another relative spatial object.

An application can construct arbitrary graphs of relative spatial objects. For example, in Figure 3-5, object 1 is to the west of object 2, which is to the west of object 3, and so forth. *RelativeSpatialObject* provides methods to trace through the transitive closure of a graph operation. In the above example, since object 1 is to the west of object 2 and object 2 is to the west of object 3, it follows that object 1 is to the west of object 3. Both objects must keep track of the relationship so that the connection can go in either direction; in the above example, it also follows that object 3 is to the east of object 1. If a large number of links separate two related objects, an application might want to establish a direct connection. Alternatively, the visual memory could cache this information.

The design of *RelativeSpatialObject* must determine how to handle transformation of an object in a relative object graph. In Figure 3-5, object 2 was to the east of object 1. If object 2 moved west, it could be either to the east or to the west of object 1, as shown in Figure 3-6 and Figure 3-7 respectively. When an object is transformed, the visual memory must eliminate all of its relative dependencies. If objects maintain their relationship after transformation, that relationship must be reasserted. If objects are somehow connected so that the relationship is always maintained, they should be established as subobjects of a larger object that maintains the relationship.

### 3.3.3 Uncertain Spatial Specification

Some computer vision applications do not know exactly where objects are located and exactly which points are in the point sets. They deal with approximate information and conflicting evidence from multiple sources. These applications require uncertain spatial specifications.

The visual memory class *ProbabilisticPointSet*, a subclass of *PointSet*, represents uncertain spatial information. *ProbabilisticPointSet* associates with each point the probability that it belongs to the point set. Thus instead of just knowing that point (3,4,5) was in a point set, a probabilistic point set would know that point (3,4,5) was

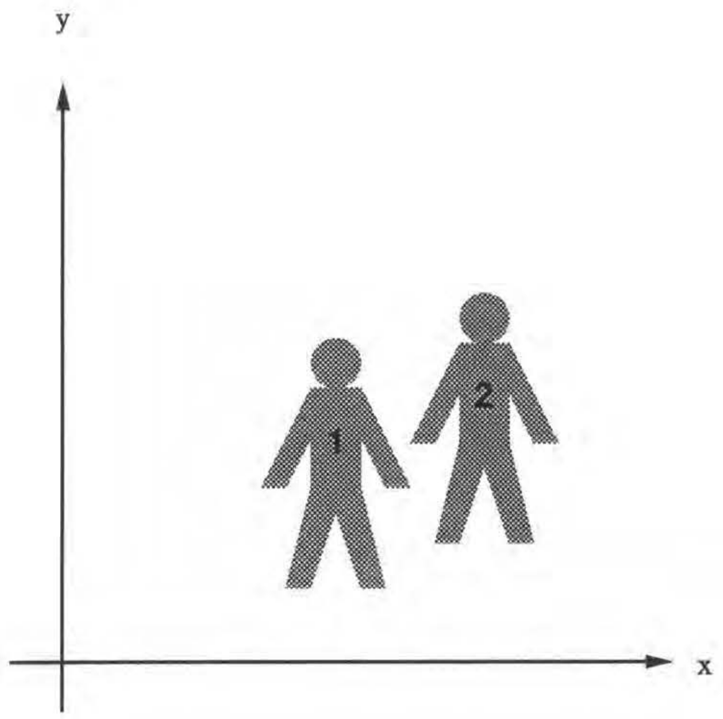


Figure 3-6: Breaking a relative spatial specification, part 1

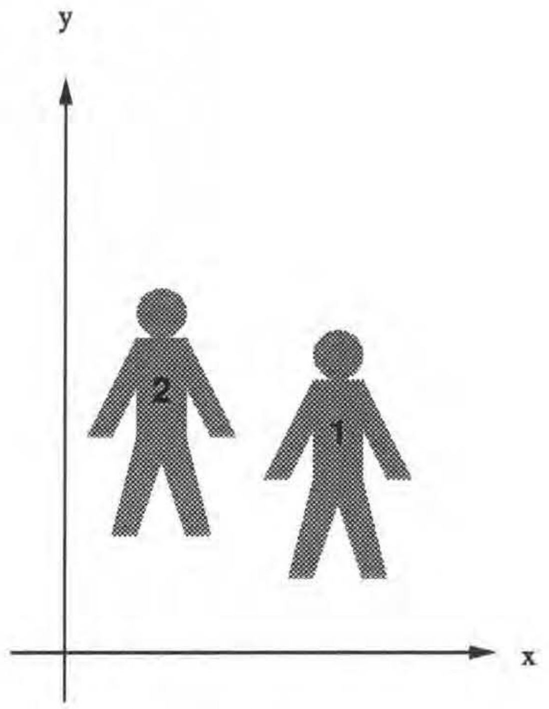


Figure 3-7: Breaking a relative spatial specification, part 2

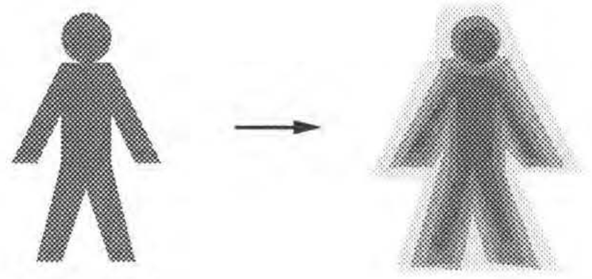


Figure 3-8: Uncertain edges

in the point set with probability 0.7. Point probabilities allow applications to specify empirical certainty factors for point sets. As will be shown below, this provides great flexibility in representing uncertain spatial information.

The standard *PointSet* methods must be modified to take point probabilities into account. The methods can reflect uncertainty in various ways. For example, there is no real boundary to a point set with point probabilities asymptotically approaching 0. However, an application can define the boundary as the set of points with probability 0.1. Subclasses of *ProbabilisticPointSet* support such variations.

Like *PointSet*, *ProbabilisticPointSet* has both discrete and abstract subclasses. The discrete subclass maintains a list of <point, probability> pairs, while the abstract subclass defines a function that returns a probability for a given point. Set operations can combine the point probabilities from different point sets by maximizing, minimizing, or averaging, among other operations.

Probabilistic point sets can handle a number of different types of uncertain spatial specification. The following sections examine a few of these.

### Uncertain Edges

*ProbabilisticPointSet* lets applications be fuzzy about the region of space occupied by an object. For example, the probability function can decrease at the edges of an object, where a segmentation algorithm might be unsure of exactly how to separate regions. Figure 3-8 shows examples of a standard person and a person with uncertain edges, where darker regions have higher probability.

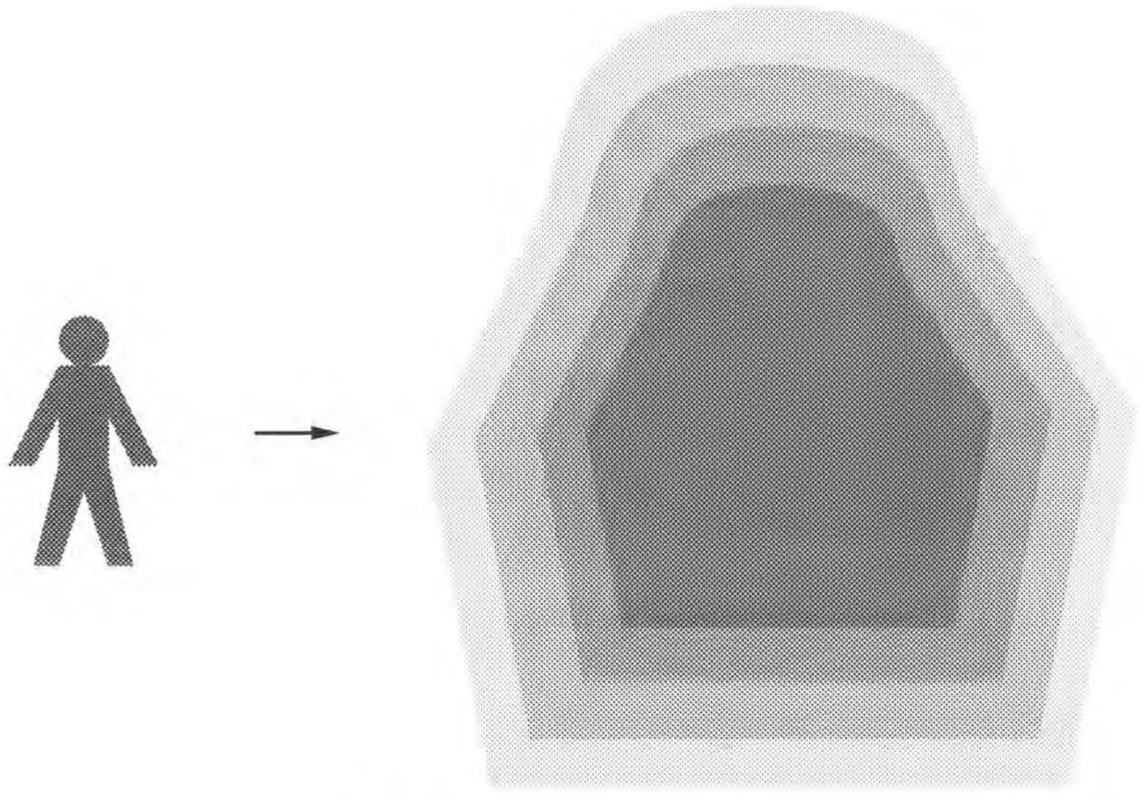


Figure 3-9: Uncertain location

### Uncertain Location

An application might know the points in the point set but might not be sure of its exact location. For example, a tracking algorithm might identify a group of points as a person and decide to use the default point set to represent it. However, it might know the person's location only to within a meter. The point set for this particular person can be "spread out" to cover the range of possibilities. An application can indicate the areas most likely to contain the object by giving them the highest probability. Figure 3-9 shows a person and a "spread out" probabilistic point set, with darker regions for points with higher probability.

### Conflicting Information

An application might have separate processes providing inconsistent information to the visual memory. For example, one process in a vehicle navigator might identify a

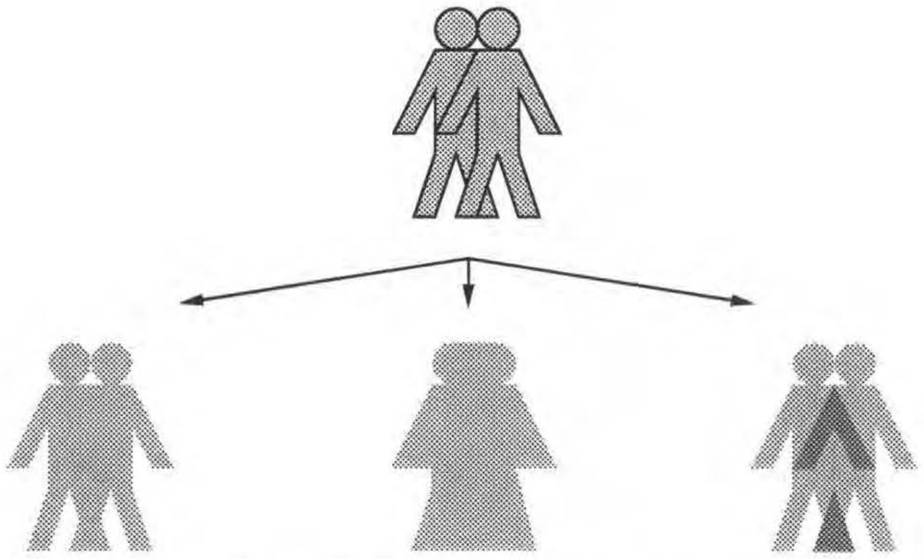


Figure 3-10: Conflicting information

car at one location, while another process might identify the same car at a slightly different location. *ProbabilisticPointSet* handles this situation similarly to uncertain location, but it must combine only a discrete set of point sets rather than spreading out one point set over a continuous area.

*ProbabilisticPointSet* provides several different ways to combine point sets. For example, it can combine point probabilities by maximizing, minimizing, summing, or differencing (with probabilities staying between 0 and 1), or it can interpolate between the point sets. These approaches to combining probabilities are similar to those taken by expert systems [7] and multi-valued logics [14].

Figure 3-10 shows two point sets to be combined and the results of three different types of combination. The points in the original point sets have the same probabilities. The leftmost combination is formed by maximization; since the probabilities are all the same, this is equivalent to a point set union operation. The middle combination interpolates horizontally between the two point sets. The rightmost combination adds probabilities, assigning higher probabilities to the areas where the point sets overlap.

## 3.4 Temporal Representations

Temporal representations fit into another branch of the visual memory class hierarchy. There are some parallels between the spatial branch and the temporal branch, but the temporal branch has many of its own requirements and features. This section presents the visual memory temporal representations. Like spatial operations, many temporal operations are complex and require the indexing mechanism of Section 3.8 to achieve high performance.

### 3.4.1 Core Temporal Classes

#### TemporalObject

The class *TemporalObject* is the basis for high-level representation of temporal information in the visual memory. Visual memory is concerned with *valid time*, the time at which events happen. *TemporalObject* provides slots and methods defining a standard interface for visual memory temporal support. Its subclasses extend the definition to handle additional temporal information. Any class that needs to keep track of its valid time should inherit from *TemporalObject*.

*TemporalObject* represents valid time as a set of time intervals and a local clock. It provides methods to manipulate this information, setting and retrieving the valid time, relating the clock to other clocks, and so forth. Most of these methods are furnished by the lower-level classes that make up *TemporalObject*, discussed further in the following sections.

#### VMTime

The most elementary temporal representation is the class *VMTime*, an abbreviation for *Visual Memory Time*. An instance of this class represents exactly one point in time. Like its spatial counterpart *Point*, *VMTime* is a fairly simple but essential building block in the visual memory class hierarchy.

### TemporalInterval

Most objects are valid not for just one point in time but rather for some duration of time. The visual memory provides the class *TemporalInterval* to represent temporal extents. *TemporalInterval* is defined as an open interval $[t_1, t_2)$ to denote valid time from time $t_1$ to time $t_2$. The interval is open because applications generally recognize when an object is first valid ($t_1$) and when it is first invalid ($t_2$); its valid interval then extends from $t_1$ up to but not including $t_2$.

*TemporalInterval* provides a variety of methods for manipulating temporal information. Standard methods set and retrieve the starting and ending times of the interval. Additional methods check interval overlap, combine overlapping intervals, and check the equality of intervals.

### TemporalElement

While some temporal databases use the temporal interval as the main temporal representation, that is insufficient for all visual memory applications. One problem is that the difference of two temporal intervals might not be a temporal interval: if interval 1 covered [10, 30) and interval 2 covered [15, 25), the difference would be [10,15) $\cup$ [25,30). The same problem occurs with disjunction, when an object is valid for multiple distinct intervals. The visual memory follows Elmasri [10] and goes one step further than *TemporalInterval* to provide a more powerful temporal representation.

The class *TemporalElement* maintains a temporal object's valid time in the visual memory. A temporal element consists of a set of temporal intervals. Thus it is closed under set operations and can represent complex temporal specifications. Each of the less expressive temporal representations is a subcase of *TemporalElement*: *TemporalInterval* is a singleton *TemporalElement* and *VMTime* is a singleton *TemporalElement* with the same starting and ending point. Figure 3-11 depicts an example temporal element.

*TemporalElement* furnishes many methods for manipulating its temporal information. It is a subclass of the generic *Set* class, so it provides standard set operations such as member, conjoin, disjoin, and difference. In addition, by using *TemporalEle-*

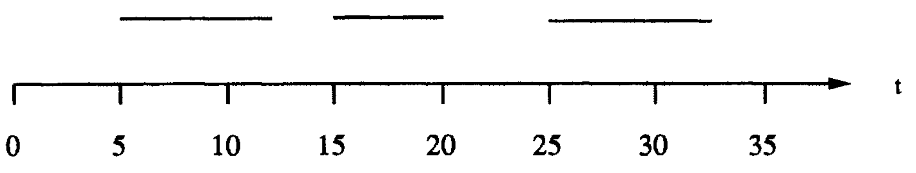


Figure 3-11: Temporal element

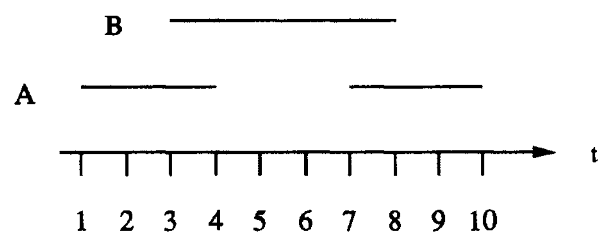


Figure 3-12: Overlapping temporal elements

*ment* methods, an application can set and retrieve valid times, compare valid times, combine overlapping intervals in a temporal element, and resolve two temporal elements, eliminating overlapping times from one in favor of the other.

Resolution of conflicting temporal elements is an important concept in the visual memory. An application can specify what to do in case of conflict between valid times: it can resolve in favor of the original valid time, it can resolve in favor of the new valid time, or it can leave them in an inconsistent state. Figure 3-12 shows two overlapping temporal elements, version A and version B; Figure 3-13 and Figure 3-14 show the two ways in which they can be resolved. Temporal resolution is especially useful for an application that is initially unsure of the full extent of an object's valid time. The application could assume that the object was valid from a given point until told otherwise and then later resolve that when it learned more information.

Like its spatial counterpart *PointSet*, *TemporalElement* has both discrete and abstract subclasses. The class *DiscreteTemporalElement* lists all the temporal intervals in the set, while the class *AbstractTemporalElement* uses a function to determine whether or not a given temporal interval is in the set. Since time is one-dimensional

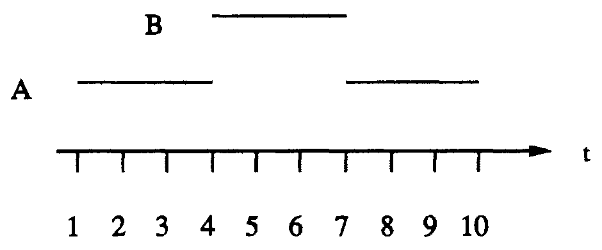


Figure 3-13: Temporal resolution in favor of version A

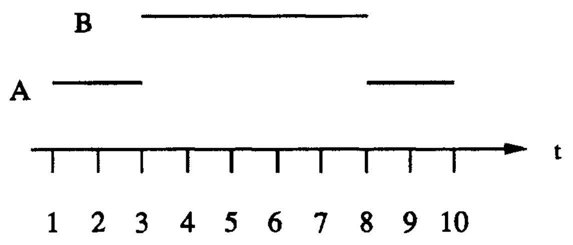


Figure 3-14: Temporal resolution in favor of version B

and most valid times are in just a few continuous blocks, the discrete class is probably more useful for most applications. The abstract class is available for applications that need to represent a large number of disjoint intervals.

### Clock

A time point makes sense only with specification of the clock on which it was measured. The visual memory provides the class *Clock*, the temporal analog of the spatial class *CoordinateSystem*, to represent this information. Each clock can assign a different meaning to time points: one clock might use milliseconds since January 1, 1900, while another might use seconds since March 8, 1970. In addition, a *Clock* instance can specify the machine on which the clock is located so that applications can try to account for inaccuracies and differences between system clocks.

The *TemporalObject* class associates a clock with a temporal element. Clocks are associated at this level of granularity because *TemporalElement* is the main visual memory temporal representation. Using a finer granularity would hurt performance

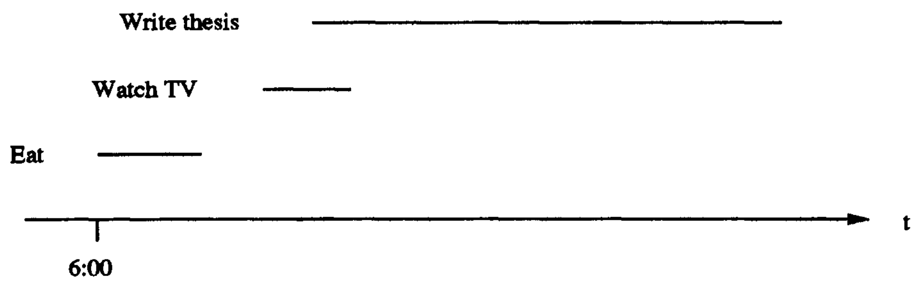


Figure 3-15: Relative temporal specification

for complex temporal specifications and would not greatly improve performance for simple temporal specifications that can be represented as trivial temporal elements.

Like each coordinate system, each clock provides a set of transforms between it and other clocks. This establishes meaning behind the time points associated with a clock and allows the visual memory to convert times among clocks. To increase performance, applications can use the same or compatible clocks.

### 3.4.2 Relative Temporal Specification

Some applications, such as planners and schedulers, do not know explicit temporal information but can specify some relative ordering of events. For example, a planner might know that it must move to the other side of the room, which will take 5 seconds, before it can pick up a block. To support these applications the visual memory provides classes representing relative temporal specifications.

There are two kinds of relative temporal specification: specification relative to a definite time or object with a definite time, and specification relative to another relative temporal specification. For example, Figure 3-15 illustrates that I plan to eat dinner after 6:00, watch TV after that, and start writing my thesis while I watch TV. This description does not precisely specify the times of these events; if I took a longer break between eating and watching TV the relative specification would be the same.

The visual memory class *RelativeTemporalObject*, a subclass of *TemporalObject*, supports relative temporal specification. A relative temporal object maintains a list of other objects to which it is temporally related. For example, one subclass of *RelativeTemporalObject* might keep track of events before and after it, while another might maintain a list of events happening at approximately the same time.

*RelativeTemporalObject* allows applications to build arbitrary graphs of temporal relations. For example, the specification in Figure 3-15 directly relates a time and three temporal objects. *RelativeTemporalObject* also provides methods to trace through the transitive closure of a graph. In this example, it could report that I will study after 6:00. Both related objects must keep track of the relationship so that the link can be traversed in either direction. In this manner, the visual memory could also report that 6:00 is before the time when I will study.

### 3.4.3 Uncertain Temporal Specification

In many cases an application might be unsure about the valid time of an object's state. This could happen, for instance, if the application did not notice an abrupt change of state or could not pinpoint the time of the state change. The visual memory provides two classes, *ProbabilisticTemporalInterval* and *ProbabilisticTemporalElement*, to support uncertain temporal information. Like their spatial counterpart *ProbabilisticPointSet*, these classes follow in the tradition of multi-valued logics and expert system certainty factors.

#### ProbabilisticTemporalInterval

*ProbabilisticTemporalInterval* extends the definition of an interval to include a function that, given a time, returns the probability that the interval includes that time. Thus, as shown in Figure 3-16, a probabilistic temporal interval can specify that the valid time most likely includes [10,25), is increasingly less likely to include times on the other sides of 10 and 25, and definitely does not include times outside of [5,30). This probability drop-off could indicate where the application was trying to determine state-change boundaries. The deterministic temporal interval is merely a special case where the probability is 1 during a specific interval and 0 elsewhere.

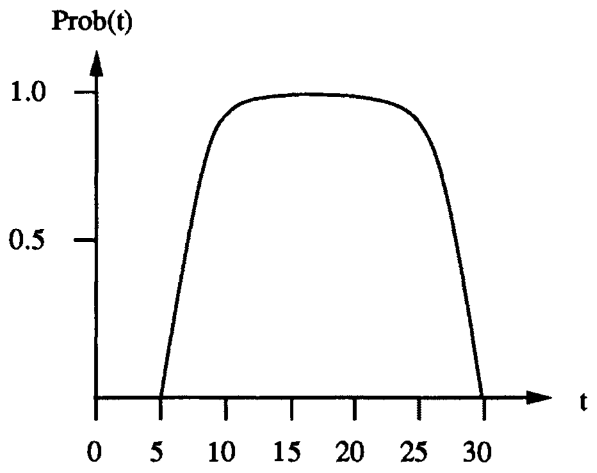


Figure 3-16: Probabilistic temporal interval

*ProbabilisticTemporalInterval* modifies standard *TemporalInterval* methods to use the temporal probability function. For example, a probabilistic temporal interval does not have clearly-defined endpoints; the method to find endpoints uses a threshold supplied by the application to separate points in the interval from those outside it.

## ProbabilisticTemporalElement

*ProbabilisticTemporalElement*, a subclass of *TemporalElement*, contains a set of probabilistic temporal intervals rather than a set of temporal intervals. This allows a temporal object to represent the probability that it is valid during a time in a set of disjoint intervals.

The methods of *ProbabilisticTemporalElement* are specialized to handle temporal probability. For example, multi-valued logic systems often define probabilistic conjunction as a minimization operation and probabilistic disjunction as a maximization operation [14]. Figure 3-17 shows two overlapping temporal elements; Figure 3-18 demonstrates conjunction by minimization and Figure 3-19 shows disjunction by maximization.

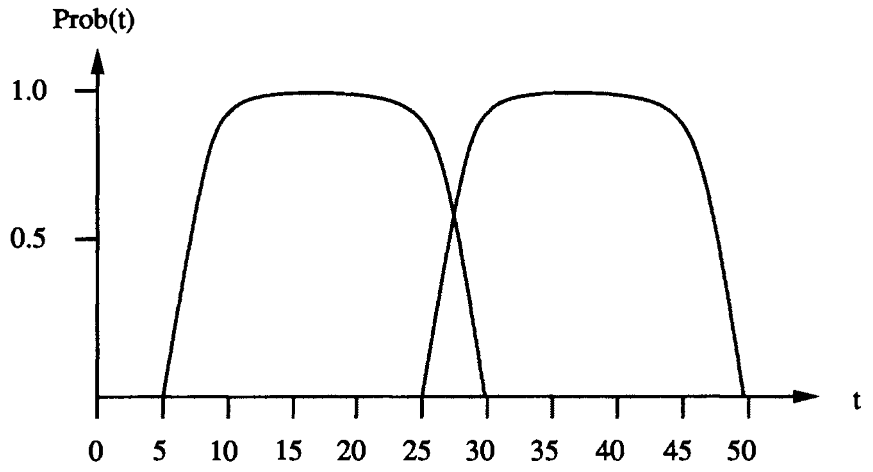


Figure 3-17: Overlapping probabilistic temporal intervals

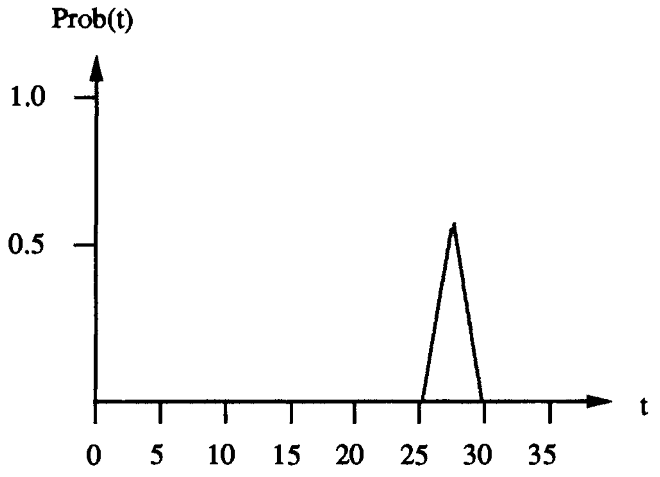


Figure 3-18: Probabilistic conjunction by minimization

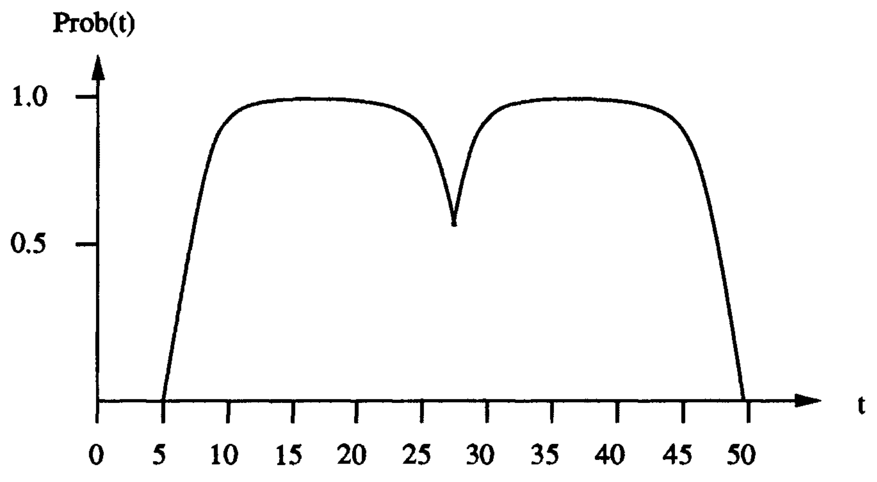


Figure 3-19: Probabilistic disjunction by maximization

## 3.5 Spatiotemporal Representations

Many objects stored in the visual memory have both spatial and temporal components. For example, a vehicle navigator might watch other vehicles driving nearby and a security system might track people walking in a hall. In both of these cases, objects are moving in space over an extent of time. The meshing of spatial and temporal information in these cases suggests that, in addition to spatial and temporal support, the visual memory should provide spatiotemporal support.

The class *SpatiotemporalObject*, a subclass of both *SpatialObject* and *TemporalObject*, represents spatiotemporal information in the visual memory. Because it is a subclass of both *SpatialObject* and *TemporalObject*, it contains the same information, including a point set, a coordinate system, a set of valid times, and a clock. It also supports all the *SpatialObject* and *TemporalObject* methods for manipulating this information.

The class *DiscreteSpatiotemporalObject*, a subclass of *SpatiotemporalObject*, stores state snapshots of objects. For example, a vehicle navigator could use an instance of this class to periodically store information indicating the spatial extent of the vehicle over some interval of time. In this way it could build up a whole history of the vehicle's motion.

*DiscreteSpatiotemporalObject* provides interpolation methods to estimate additional spatiotemporal information from existing information. For example, from the information in Figure 3-20, the visual memory could interpolate the snapshot of Figure 3-21. *DiscreteSpatiotemporalObject* subclasses implement a variety of interpolation procedures; for example, the circle in Figure 3-21 could be interpolated by radius or by area, and acceleration over several snapshots could be taken into account. Interpolation allows applications to store spatiotemporal information more sparsely and still closely approximate necessary information.

Like *SpatialObject* and *TemporalObject*, *SpatiotemporalObject* also provides an abstract subclass to represent information by means of a function. *AbstractSpatiotemporalObject* uses a trajectory method to determine which points are in its point set

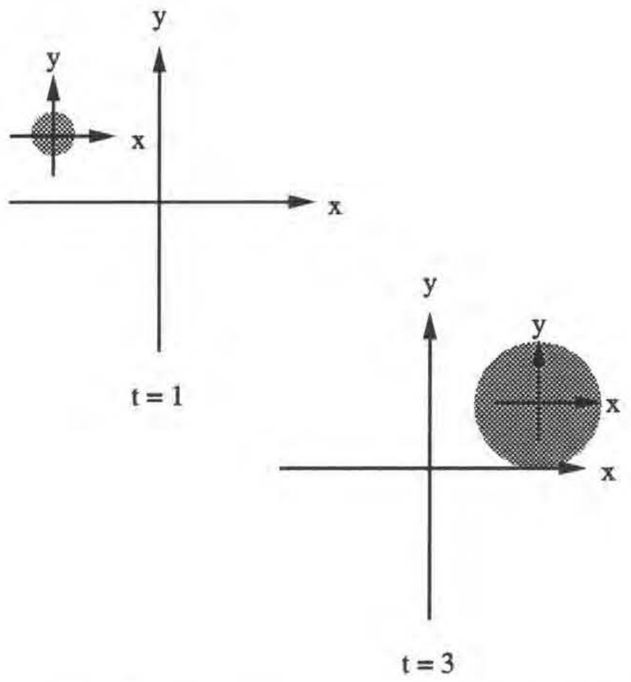


Figure 3-20: Discrete spatiotemporal information

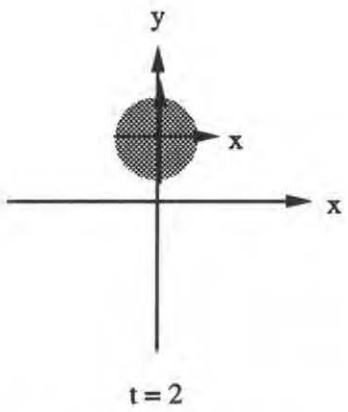


Figure 3-21: Interpolated spatiotemporal state

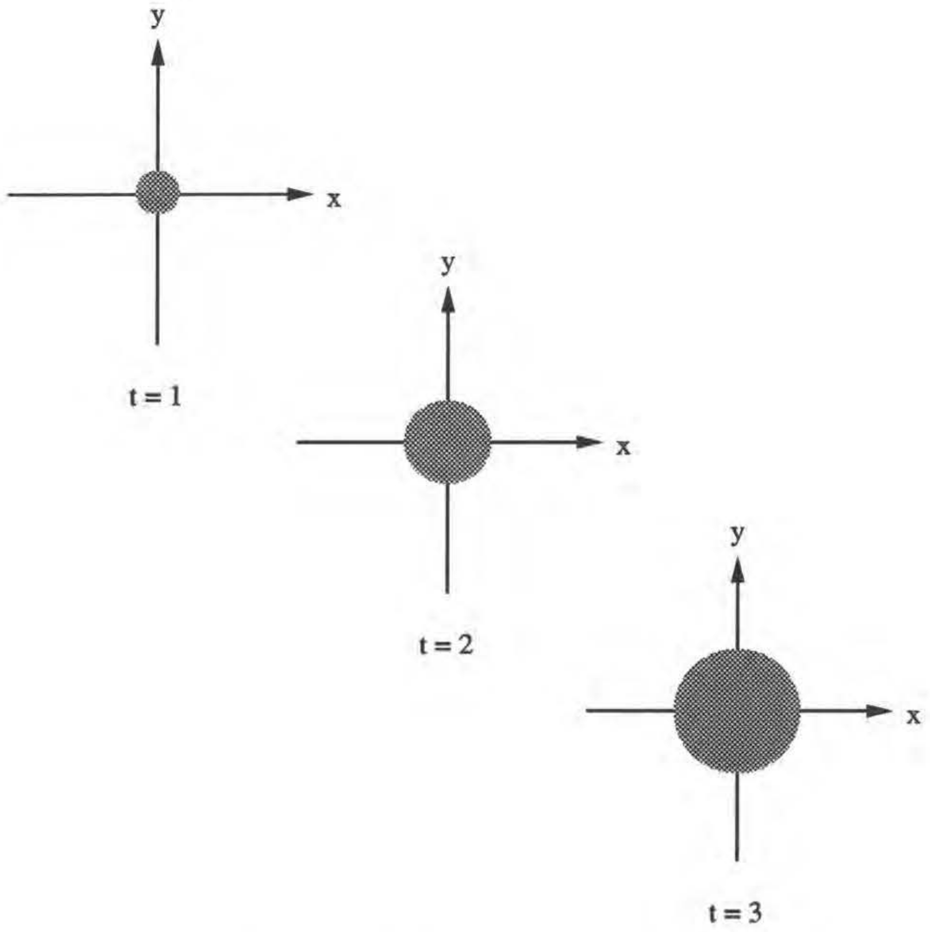


Figure 3-22: Point set trajectory

at specified times. The trajectory specifies how the object's point set and coordinate system change with time. Thus an abstract spatiotemporal object is equivalent to a set of discrete spatiotemporal state snapshots.

The spatial description of an object can change over time in two different ways: the point set itself can change, or the point set's relation with other points can change. The circle with an expanding radius shown in Figure 3-22 is an example of a changing point set, while the translating circle shown in Figure 3-23 demonstrates changing relationships. A trajectory for an abstract spatiotemporal object can handle either or both types of change.

The trajectory can modify a point set over time by supplying a time point as an additional argument to the point set function. For example, the trajectory for the

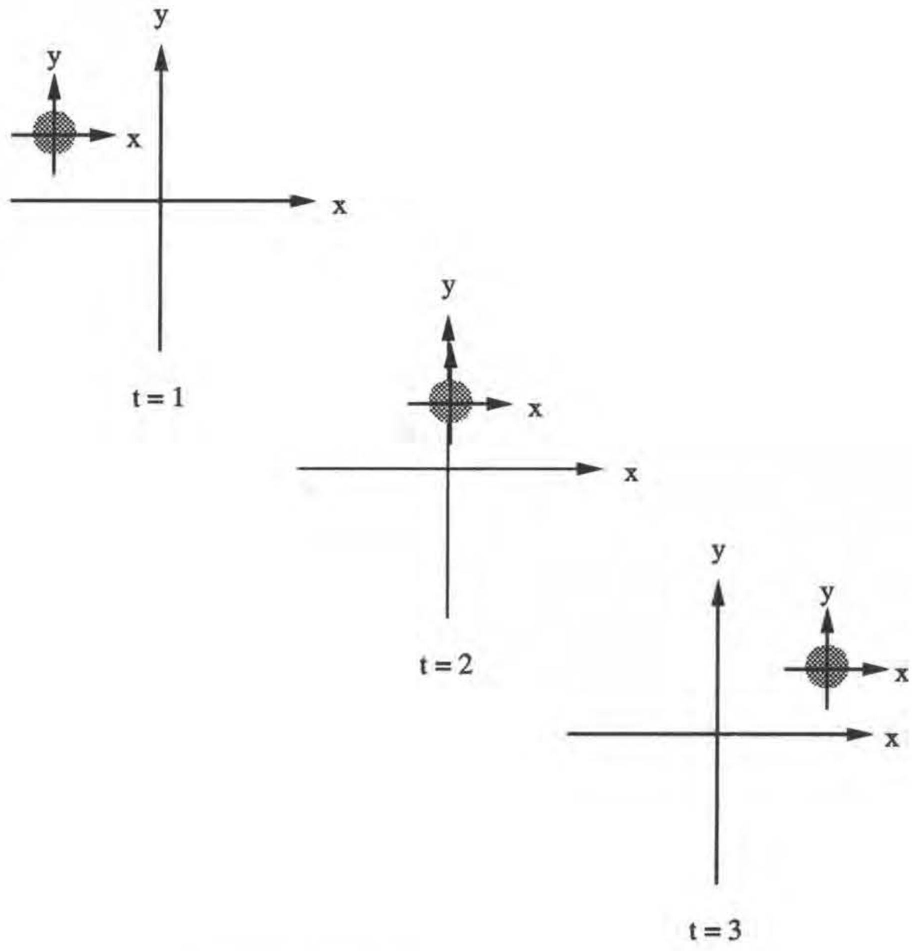


Figure 3-23: Coordinate system trajectory

circle with expanding radius in Figure 3-22 could check $x^2 + y^2 <= t$ to determine all points $(x, y)$ in the point set at time $t$. If the changes in the point set follow some pattern, the spatiotemporal point set function can capture that pattern; otherwise, the discrete approach is probably more suitable.

The trajectory can change relationships between point sets over time by establishing a function to specify coordinate system transforms as a function of time. In Figure 3-23, the trajectory would translate the coordinate system one unit along the x-axis every second. This can be implemented by establishing an initial coordinate system and its relationships to other coordinate systems and then identifying differences between the coordinate system at a given time and the initial coordinate system. This way the trajectory does not have to establish all the coordinate system's relations at each time; instead, it can transform from a given coordinate system to the established coordinate system and from it to any other related coordinate system.

Visual memory provides the class *RelativeSpatiotemporalObject* to express spatiotemporal relationships. For example, an application could describe a relative spatiotemporal object as being to the right of another object sometime after 6:00. *RelativeSpatiotemporalObject* and its subclasses simply combine the relative spatial and temporal classes detailed in earlier sections.

Spatiotemporal representations can benefit from probabilistic methods. The visual memory class *ProbabilisticSpatiotemporalObject* combines the spatial and temporal probabilistic methods previously described. It allows applications to express uncertainty about both the spatial and temporal extents of spatiotemporal objects. Probabilistic functions are especially useful with spatiotemporal interpolation, allowing a measure of uncertainty to accompany an interpolated object description. Abstract spatiotemporal objects can establish probabilistic trajectories to be imprecise about the changes in an object's spatial description over time.

## 3.6 Object Storage

An important part of the visual memory design addresses how to store and retrieve spatiotemporal information. The object-oriented database on which the visual memory builds provides basic support for object storage and retrieval. This section discusses the concepts and issues most relevant to the visual memory design.

### 3.6.1 Identity

Each object in the visual memory has a unique identity. This identity does not necessarily correspond to physical identity; for example, an application might not recognize a person appearing in its view as the same person who disappeared moments ago, causing it to create a new object for the person. To preserve identity, the visual memory assigns each object an object identitifier (OID), a number that distinguishes that object from all others. The object maintains the same OID through all of its state changes.

Each object can have multiple versions. For example, a security system could track a person walking down a hall and store a new version describing that person's location every tenth of a second. The versions of an object maintain the same OID, but each has a different version number. Thus an <OID, version number> pair uniquely distinguishes a particular state snapshot of a particular object. By maintaining all of an object's versions, the visual memory can answer questions about the object's history.

Some visual memory applications might need to combine the histories of different objects to form the history of one object. This could happen, for example, if a tracking system lost sight of a person, found a new person and created a new object, and later realized that the two people were actually the same. The visual memory can consolidate object histories to create versions of one object from versions of other objects.

An application can report to the visual memory that an object has disappeared. Making an object disappear is quite different from deleting that object, which actually removes old versions of the object from the visual memory. Disappearing does not affect an object's history but instead removes it from the current state of the visual memory. Visual memory queries after an object's time of disappearance do not retrieve that object.

### 3.6.2 Storage Mechanism

The database underlying the visual memory decides how to store objects. To implement an appropriate storage policy, the database should consider the visual memory's storage needs and the characteristics of the objects that it stores. This section discusses how object storage should be tailored for the visual memory.

Many visual memory objects change very little from one version to the next. For example, a rigid object moving across the room changes only its coordinate system and valid time; the point set, clock, and other information remains the same. In cases like this, the database should store one base version of the object and then indicate differences for each new version.

The visual memory obeys a nondeletion policy: it creates a new version each time an object changes, and it never deletes old versions. Deleting a version would cause problems for other object versions containing references to it. The visual memory is not an append-only database since it actually modifies old versions, as discussed below in section 3.6.3. Only the visual memory can modify old versions, since uncontrolled modification could lead to inconsistencies. These considerations allow the database to implement a simpler storage policy.

Some visual memory applications store a great amount of data. Since old information might never be deleted, the available space can quickly fill. Once old information has settled down and will not be accessed or modified often, the database can move it onto long-term, high-capacity storage devices. This keeps the most useful information readily available while increasing the amount of information that can be stored.

### 3.6.3 Time

As a historical database, the visual memory keeps track of when events happened. It stores with each version of a temporal object information about that version's valid time. Since one version's valid time might conflict with the valid times of other versions, the visual memory attempts to ensure consistency by resolving these valid times. Section 3.4.1 discusses temporal resolution strategies.

The valid time of a new version could conflict with the valid times of many old versions. The indexing strategies discussed below in Section 3.8 allow the visual memory to quickly identify which old versions must be changed. The necessity of resolving old temporal information encourages the use of caching techniques to reduce the number of disk accesses.

Applications can improve the performance of temporal resolution by operating in "real-time mode." In real-time mode, the valid time of the latest version of an object is an infinite interval starting from the current time. Thus each new version must be resolved only with the previous version. For example, if the first version were valid $[0, \infty)$, then a second version valid $[5, \infty)$ would change the first version's valid time to $[0, 5)$, a third version valid $[10, \infty)$ would change the second version's valid time to $[5, 10)$, and so forth. Performing only one temporal resolution per object update can greatly improve storage performance.

## 3.7 Queries

The visual memory provides a powerful and expressive mechanism for retrieving information. This query mechanism is tailored to the spatial and temporal representations presented in earlier sections. It is also designed to meet a wide variety of retrieval needs, providing flexibility in specifying objects of interest. This section describes the query mechanism and the types of queries supported by the visual memory.

### 3.7.1 Query Mechanism

The visual memory query mechanism extends a standard SQL-based [1] object query language, such as OQL [2]. The queries below demonstrate the basic form and functionality of such a query language.

Find everyone with the same age as the object stored in program variable "me":

```
Select p from Person
  where p.age() == %me.age()
```

Find everyone named Larry who used to play professional basketball:

```
Select p from Person
  where p.firstname() == "Larry" and
    p.occupation().title() == "pro basketball player" and
    p.occupation().status() == "retired"
```

Find the children of the above people:

```
Select p from Person
  where p.father() in
    (Select p from Person
        where p.firstname() == "Larry" and
            p.occupation().title() == "pro basketball player" and
            p.occupation().status() == "retired")
```

The database literature contains many examples demonstrating the power of query languages. The visual memory query language extensions allow applications to construct complex spatiotemporal queries.

A query language provides flexibility and expressiveness but can be hard to use. For applications that do not need the full power of a query language, graphical query specification might be more suitable. A graphical query could be specified by outlining regions of space and intervals of time; objects satisfying the specification could also be displayed graphically. A graphical query language could be built over the visual memory query language by transforming graphical specifications into visual memory queries. Chapter 4 discusses an implementation of such a graphical query language.

A query mechanism works on two levels, on disk and in memory. The visual memory indices, discussed further in Section 3.8, provide information to help the query mechanism eliminate objects that do not satisfy a query before bringing them into memory. The query mechanism then further filters these objects to determine which objects satisfy the specification. A number of the query constructs outlined below could easily be performed in memory but are implemented as part of the query language to allow the query language to optimize object retrieval.

Rather than adding a large number of special spatial and temporal constructs to the query language, the visual memory bases its query support on instances of the spatial and temporal classes discussed in previous sections. Each query includes spatial or temporal keywords and a spatial or temporal object; the keyword describes how instances satisfying the query must interact with the given object. The specified spatial or temporal object could be a program variable, allowing the application to form a complex specification before posing the query. Alternatively, it could be the result of another query, allowing an application to compose queries. These mechanisms provide great flexibility in spatial and temporal query specification.

### 3.7.2 Spatial Queries

Instances of the class *SpatialObject* form the basis for all spatial queries. A query specifies a spatial object of interest and how objects satisfying the query must interact with that spatial object. Described below are the ways that applications can use a

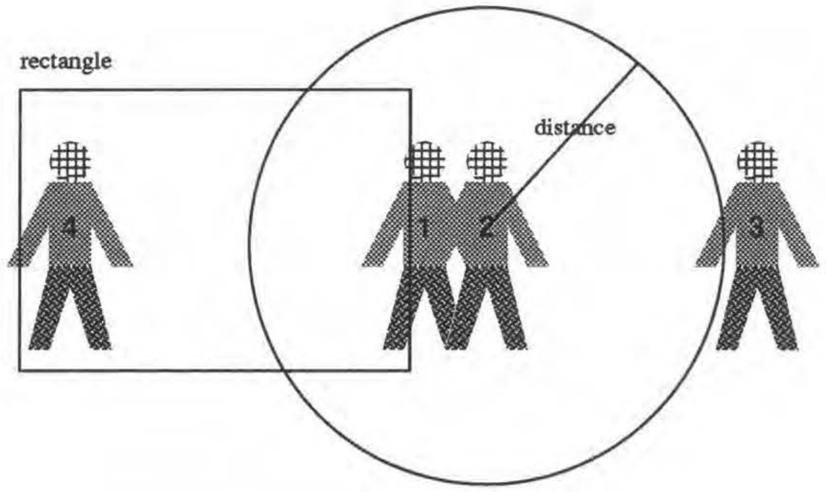


Figure 3-24: Spatial queries

specified spatial object to retrieve objects of interest. The accompanying examples demonstrate the query language syntax and reference objects of Figure 3-24.

**Intersects Query**

The *intersects* query looks for the intersection of spatial objects. For example, the following query returns the set { person-1, person-4 }:

```
Select p from Person
  where p intersects %rectangle
```

This is one of the most broadly useful spatial query constructs. The specified spatial object can be a point, line, pentagon, pyramid, or just about any other spatial specification imaginable. This construct is also useful when negated. For example, the set { person-2, person-3 } satisfies the following query:

```
Select p from Person
  where not p intersects %rectangle
```

A security system could use intersection to find all the objects within a room, and a vehicle navigator could use negated intersection to make sure that nothing was on the road in front of the vehicle.

### Borders Query

The *borders* query checks for bordering objects. The set { person-4.torso() } satisfies the following query:

```
Select p from Person
  where p borders %person-4.head()
```

A VLSI system could use this query construct to look for electrical contact, and a photo interpretation system could use it in constructing a high-level representation of connected regions. Applications can use probabilistic point sets to specify imprecise borders for this query.

### Centroid-Within Query

The *centroid-within* query ignores the spatial extent of objects and checks distances between centroids. For example, the following query returns the set { person-1 person-2 }:

```
Select p from Person
  where p centroid within %distance of %person-2
```

This distance parameter specifies within how many units, using the specified spatial object's coordinate system, an object must be to satisfy the query. With this query, applications can quickly gather objects roughly within a given distance from a specified object. The estimation is fairly accurate if the point sets are much smaller than the distance between them.

### Point Set-Within Query

To select nearby objects with greater accuracy than the *centroid-within* query provides, applications can use the *point set-within* query. The following example selects the set { person-1, person-2, person-3 }:

```
Select p from Person
  where p point set within %distance of %person-2
```

This query is similar to the *centroid-within* query, but it retrieves all objects that have at least one point within the given distance of any point of the specified spatial object. To select objects meeting some specialized definition of nearness, an application can construct any spatial object and perform an *intersects* query; this is just a specialized, optimized version of that process.

**Transitive-Closure Query**

The *transitive-closure* query compounds any of the above specifications, applying a query to its results until there are no new results. It returns all objects identified in the process. For example, the transitive closure of a *borders* query shown below returns the set { person-4.torso(), person-4.legs() }:

```
Select p from Person
  where p borders by transitive closure %person-4.head()
```

This query retrieves any objects bordering the given object, any object bordering those objects, and so forth. A photo interpretation system could use it to find connected regions.

### 3.7.3 Temporal Queries

The visual memory temporal query mechanism retrieves all the versions of objects that satisfy some set of constraints. A temporal query specifies a *TemporalObject* instance to describe the times of interest and a keyword to describe how the valid time of a satisfying version must interact with those times. Described below are the visual memory temporal query specifications. Accompanying examples demonstrate the query language syntax using versions shown in Figure 3-25.

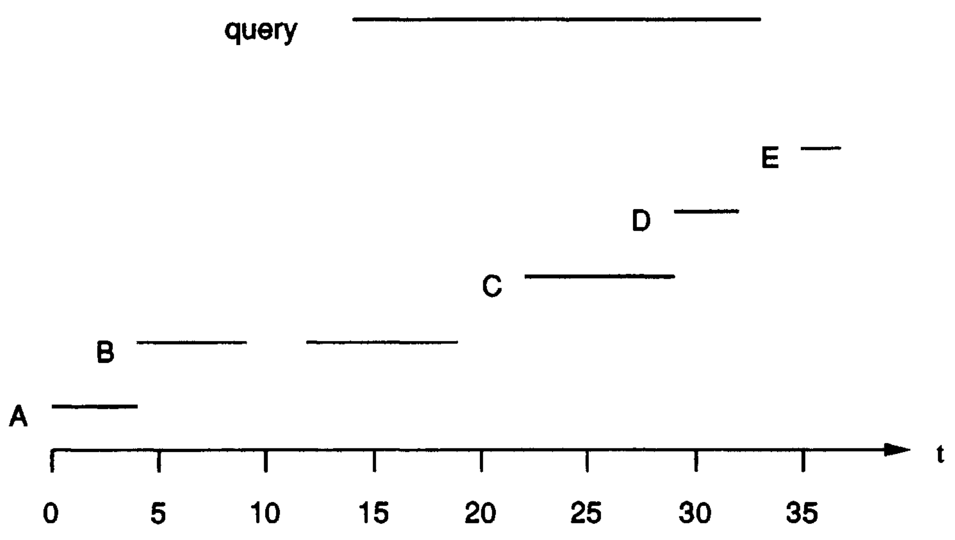


Figure 3-25: Temporal queries

### During Query

The *during* query checks for versions whose valid times intersect the given valid time. For example, the following query returns the set { Version-B, Version-C, Version-D }:

```
Select p from Person during %query
```

This is a very powerful query, allowing applications to retrieve versions during any specified set of times. It is also useful in its negated form, where it returns versions whose valid times do not intersect the given valid time. The negated query below selects the set { Version-A, Version-E }:

```
Select p from Person
  not during %query
```

### Latest-During Query

The *latest-during* query retrieves only the latest version of an object during some specified temporal element. For example, the set { Version-D } satisfies the following query:

```
Select p from Person
  latest during %query
```

An application could use this query to update a memory-resident model with the latest information in the visual memory. For example, a vehicle navigator could establish a model of static objects at the beginning of its execution and then use this query to update that model with the latest dynamic information stored by image processing software.

### 3.7.4 Spatiotemporal Queries

In addition to spatial and temporal queries, the visual memory supports spatiotemporal queries. Some of this support comes from the query language's natural ability to handle combined specifications. For example an application could pose the following query:

```
Select p from Person
  where p intersects %square
  during %times
```

This query retrieves all versions of all objects valid during the specified times and intersecting the specified square. Figure 3-26 depicts five states of a spatial object, at time $t = 1$ through $t = 5$. Figure 3-27 depicts a square valid over [1,5) and shows that the above query would return the third state of the object.

The joint spatial and temporal query checks a static spatial object over time, so it does not handle interactions between spatial and temporal information. Some applications want to track a moving object and retrieve versions near it at various times. To handle cases like this, the visual memory provides spatiotemporal queries.

A spatiotemporal query specifies a spatiotemporal object and a temporal object, and how objects must interact with these to satisfy the query. The spatiotemporal object's history describes where an object must be at given times, and the temporal object specifies a portion of the history of the spatiotemporal object. The query can use any of the spatial constructs discussed above to specify spatiotemporal interac-

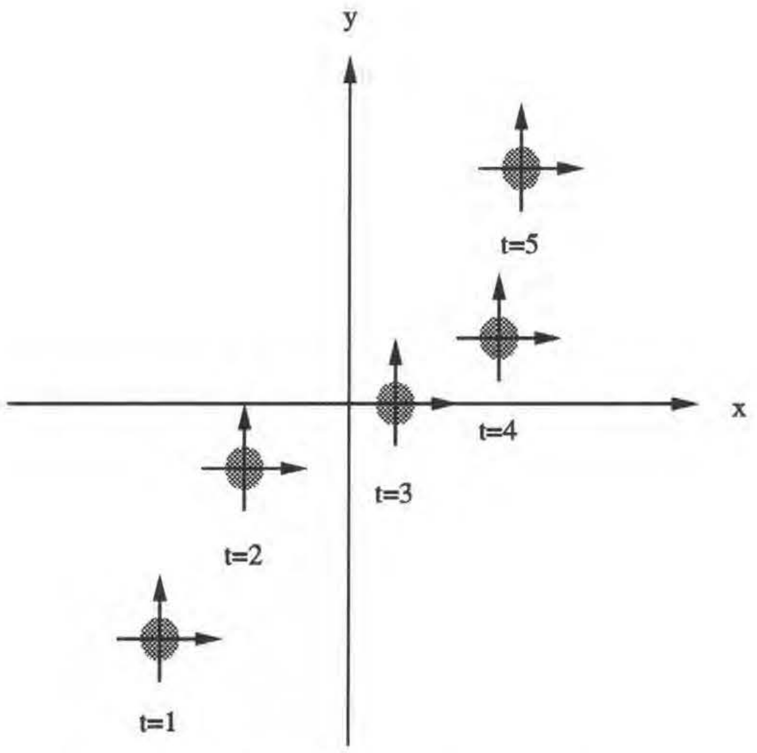


Figure 3-26: States of a spatiotemporal object

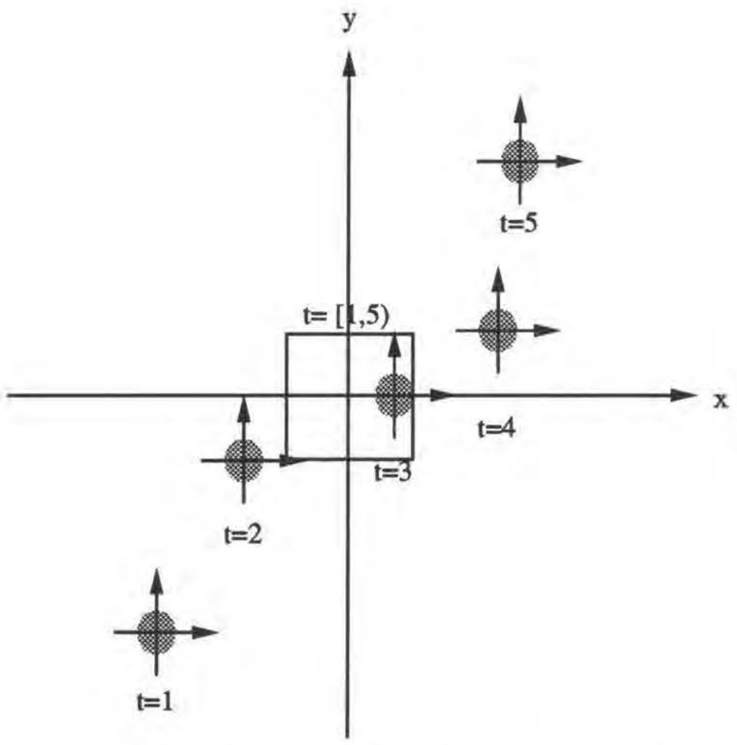


Figure 3-27: Joint spatial and temporal queries

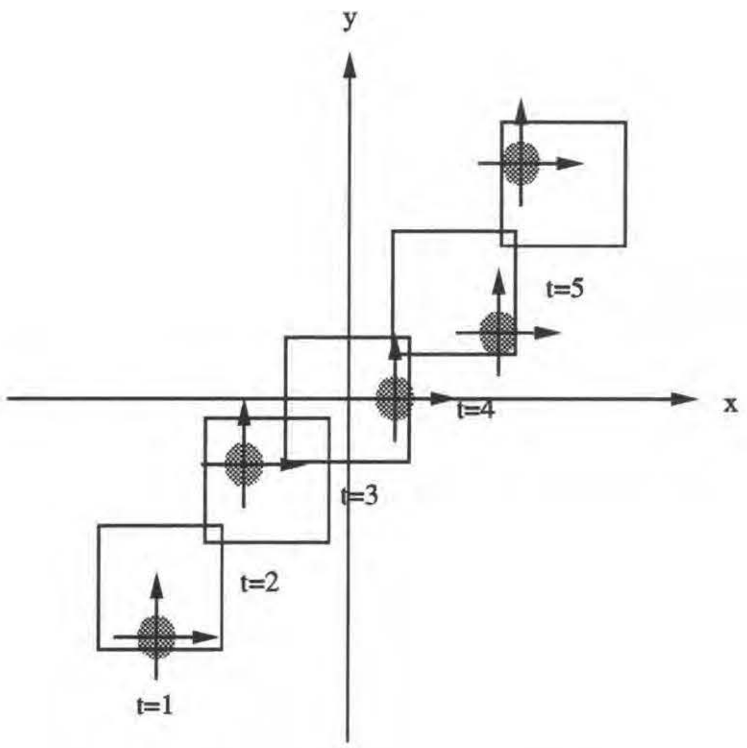


Figure 3-28: Spatiotemporal queries

tions. For example, consider the following query:

```
Select p from Person
  where p intersects %square
  during %times
```

This query construct retrieves versions of objects that intersect the square in its trajectory over a set of times. The query shown in Figure 3-28 uses as the spatiotemporal query object a square translating equally in the x and y dimensions over time. This query returns all five states of the object of Figure 3-26.

This powerful query construct can handle many complex queries, especially when combined with the join capability of the query language. For example,

```
Select p from Person
  during %times-1
  where p in
     (Select q from Person
       where q centroid within 3 of %spatiotemporal-spec
       during %times-2)
```

This query tracks all objects that came within 3 units of a given object on its trajectory during a certain set of valid times. Queries like this demonstrate the power of a query language extended with the visual memory spatiotemporal constructs.

## 3.8 Indices

The visual memory provides an indexing mechanism to quickly identify objects meeting sets of constraints. Indices tie in with both the query mechanism and the various spatial, temporal, and spatiotemporal operations described in preceding sections. For example, a spatial index can help identify solutions to an intersection query retrieving objects stored in the visual memory, and it can help identify intersecting memory-resident objects. The two types of indexing work similarly, so for conciseness this section primarily considers how indices can improve retrieval performance.

Indices maintain information allowing them to quickly eliminate objects that do not satisfy a query. They provide *conservative* approximate answers to queries; that is, they can mistakenly retrieve objects that do not satisfy a query, but they can never mistakenly leave out objects that do satisfy a query. The design of an index must trade off between how quickly the index can answer a query and how much overhead is necessary to maintain the indexing information. A well-designed index can greatly help query performance while adding minimal information overhead.

### 3.8.1 Mechanism

Visual memory indices are object-oriented: they are objects and they maintain information about objects. This yields a consistent approach to information representation. The database can store and retrieve indices just like other objects. Indices can keep track of other indices, a technique further discussed below. Finally, due to the extensible nature of the object-oriented approach, it provides flexibility in designing indices.

The purpose of an index is to maintain information to help it efficiently identify objects that might satisfy a query. In the visual memory design, this information consists of <OID, version number> records, each uniquely specifying a particular version of a particular object. An index structures these records so that it can quickly provide a set of records identifying the objects that meet specific constraints.

Indices maintain information in many different ways, such as tables, arrays, and trees. The visual memory can handle a very large index by retrieving only a necessary, manageable part at a time. However, an index must strive to minimize the amount of retrieval required to reach an answer, so that the the cost of using the index does not outweigh the query efficiency it yields.

An application can specify which sets of objects it wants to index and how it wants to index them. It simply specifies the class of the index desired and the set of objects for which it should maintain information. For example, consider the following examples of index specification:

```
Index temporal-btree on
  (Select p from Person)

Index spatial-grid on
  %my-set

Index spatial-quadtree on
  (Select o from Object
     where o intersects %my-room)
```

The first example establishes a temporal index for all people; the second establishes a spatial index on a specific set specified by a program variable; the third indexes all the objects in a certain scene. The visual memory maintains a list of all the indices in use and knows when to update them and for which queries they are appropriate.

The following sections present issues in the design of spatial, temporal, and spatiotemporal indices. Chapter 4 discusses additional indexing issues raised by one visual memory application and describes indices designed for the application.

### 3.8.2 Spatial Indices

Spatial indices organize information about the objects in a scene. The literature describes many different spatial indices; see [18] for descriptions of quite a few. Different spatial indices use different parts of an object's spatial representation and thus are most appropriate for different queries. For example, a point quadtree uses an object's centroid and works best with proximity queries, while an interval tree uses spatial intervals and is most suitable for intersection queries. An application must pick spatial indices applicable to its retrieval needs.

### 3.8.3 Temporal Indices

Temporal indices store information about object histories. The task of ordering the temporal component of an object is similar to that of ordering the spatial component, if time is viewed as just another dimension. Thus a lot of spatial indexing research applies to temporal indexing as well. For example, a spatial interval tree could store lists of versions valid during temporal intervals. However, temporal representation poses some concerns unique to temporal indexing.

Temporal indices must address the monotonicity of time. The visual memory allows applications to modify the past or predict the future, but some applications maintain an always-increasing sense of time. This could hurt the performance of some temporal indices; for example, a tree could become unbalanced. Temporal indices still need to support nonmonotonic temporal specification, but some could be optimized for the monotonic case.

Because a temporal index retains historic information, it constantly increases in size throughout its lifetime. A temporal index must not lose too much efficiency as it grows. Some temporal indices should even partition their data between short- and long-term storage, as in [9].

Temporal indices must be able to represent infinite temporal intervals. An infinite interval occurs, for example, when an application assumes that an object will be valid until otherwise notified and assigns the object a valid time extending to infinity. An infinite interval would cause problems for a temporal index representing intervals as collections of subintervals in a tree.

### 3.8.4 Spatiotemporal Indices

Spatiotemporal indices store spatial information about a scene as it varies in time. The interaction of space and time makes spatiotemporal indexing a complex problem.

There are two kinds of spatiotemporal indexing, corresponding to the discrete and abstract spatiotemporal classes discussed in Section 3.5.

The first type of spatiotemporal indexing stores information about versions of discrete spatiotemporal objects. The indexing is a two-step process: spatial indices maintain spatial descriptions of objects, and temporal indices maintain the temporal descriptions of the spatial indices. To perform a spatiotemporal query, the indexing mechanism finds the temporal description in the temporal indices, retrieves the corresponding versions of the spatial indices, finds the spatial description in them, and retrieves the corresponding spatiotemporal object versions.

Discrete spatiotemporal indexing must address some concerns. Spatiotemporal objects that move continuously cause constant index updates. This leads to large temporal indices, raising the issues previously discussed. The structure of a spatial index used in spatiotemporal indexing should not depend on the objects contained within it, since those objects move.

The second type of spatiotemporal indexing stores information about abstract spatiotemporal objects. An abstract spatiotemporal index could build up its own spatiotemporal function representing a set of object trajectories. Given a spatiotemporal specification, this function would return a list of those objects satisfying it. This function could grow very complex, so the index would have to devise some means of efficiently storing, retrieving, and evaluating it. In this manner an index could efficiently answer queries about abstract spatiotemporal objects.

# Chapter 4

# Implementation

To test the visual memory design, a subset of it was implemented in support of a real-time scene monitoring prototype. In this prototype, image processing using video cameras tracks objects and stores information about them in the visual memory. Through a graphical query interface, users can specify queries to the visual memory and view the results in various ways. Figure 4-1 shows the basic flow of information in the prototype. This chapter describes the implementation of the scene monitoring prototype and the visual memory supporting it.

Scene monitoring is a good testbed for the visual memory. Its constant updates and retrievals of information test the visual memory's performance. Multiple sensors and outputs test concurrency issues. The query interface tests the power of the query language by specifying a variety of queries, including spatial ("Watch for anything that comes within 3 feet of that button."), temporal ("Play back the last 10 seconds."), and spatiotemporal ("Did anybody come into the room between 12:00 and 1:00?"). Finally, the construction of such a prototype tests the usefulness of the visual memory spatiotemporal representations.

## 4.1 Database

An object-oriented database called Persistent C++, or PC++ for short [17], is the basis for the visual memory prototype. This database is a prototype for the DARPA

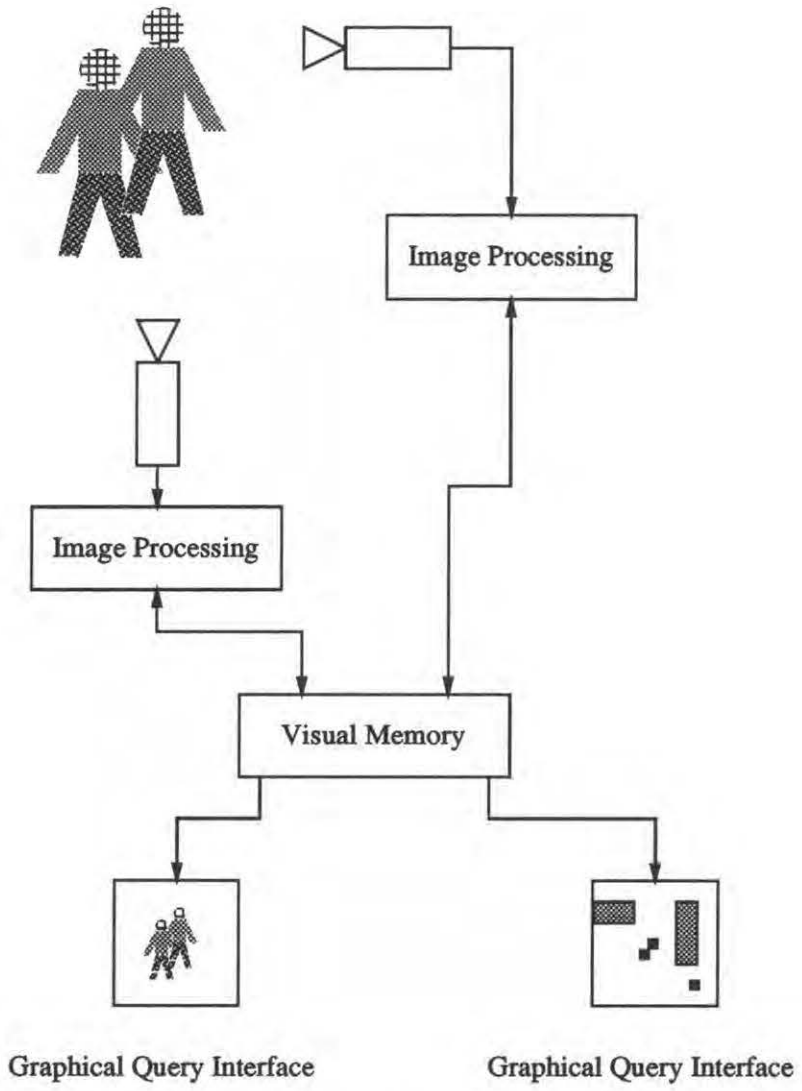


Figure 4-1: Scene monitoring prototype

Open Object-Oriented Database project at Texas Instruments [25]. PC++ has an open architecture, allowing the visual memory to add spatiotemporal extensions and take advantage of the features provided by other modules.

Some of the features provided by Persistent C++ are particularly useful to the visual memory. A versioning mechanism allows access to any previous state of any object. Transactions ensure atomicity, consistency, isolation, and durability. The object storage mechanism caches recently accessed information to increase performance.

A Persistent C++ preprocessor gathers information about the classes of objects to be stored in the database. This particular prototype preprocessor is somewhat limited, not allowing multiple inheritance or function pointers; these constraints limit the prototype in some situations. The preprocessor adds extra information to the class descriptions and forms actual C++ classes for an application to use. It adds function hooks into these classes so that the application can establish daemons to be executed when objects are stored or retrieved. Finally, when one object contains a pointer to another object, its class specification indicates either that the referenced object should be automatically retrieved with the referring object or that it should be retrieved only on demand.

Persistent C++ stores objects with the Exodus storage manager [4]. It stores a whole Exodus object for each version of a PC++ object, rather than storing differences between versions. This could hurt performance for objects that change very little from one version to the next. PC++ maintains a B-tree structure to map its OIDs to Exodus OIDs; this hurts performance as the number of OIDs grows large.

Persistent C++ can retrieve an object specified by OID and version or by a character string previously assigned to that object. It provides an object query language extension, OQL [2], but this query language does not interface well with the visual memory indexing mechanism. Thus the visual memory prototype has its own spatiotemporal query mechanism.

## 4.2 Spatiotemporal Representations

The prototype visual memory implements as Persistent C++ classes a number of the spatial, temporal, and spatiotemporal representations discussed in Chapter 3. These representations conform to the design except for some differences due to limitations in Persistent C++ and some optimizations and simplifications tailored to the scene monitoring application.

The prototype implements only the basic discrete classes. Since Persistent C++ cannot store functions, an instance cannot construct an arbitrary abstract function for its point set, temporal element, or trajectory function. In addition, the scene monitoring prototype does not need relative or probabilistic specifications.

To increase performance, the prototype uses a global coordinate system and a global clock. This eliminates the need for spatial transforms between coordinate systems and temporal transforms between clocks. Translation and rotation methods act on objects themselves rather than on their coordinate systems.

The prototype implements specific subclasses of the class *SpatiotemporalObject* to represent the objects tracked by the scene monitoring system. For example, the *Person* class adds a slot for estimations of the person's height; it could also store the person's name and other such information if it were connected to face recognition software.

## 4.3 Indices

### 4.3.1 Mechanism

Index updates occur in the visual memory prototype at transaction commit time, through Persistent C++ commit daemons. When the database stores an object, it automatically calls the object's commit daemons. The visual memory establishes commit daemons for all objects to update index information.

The visual memory prototype implements the discrete spatiotemporal indexing described in Section 3.8.4. Spatial indices store information about object locations, and temporal indices store information about the valid times of these spatial indices.

The visual memory prototype handles multiple indices. An application can create sets of indices and specify the types of information they should store and the types of queries they should answer. However, the prototype only implements start and stop control over indices; that is, an application can tell an index to start recording information about all objects committed, or to stop recording such information. This is a simpler approach than the specification of arbitrary index sets discussed in the design, but it is adequate for the prototype application.

### 4.3.2 Spatial Indices

The prototype spatial indices store information about the centroids of objects stored in the visual memory. This information allows them to efficiently answer locational and proximity queries, such as "Find everything in this square" and "Find everything within 5 units of this coordinate." Two such indices were implemented; this section describes the two-dimensional version of each.

The first spatial index is a simple fixed grid [18], dividing space into a number of cells. Each cell stores a list indicating those objects with centroids in the cell. The index can determine the correct cell for an object by rounding down the coordinates of the object's centroid, modulo the cell size. Figure 4-2 shows a fixed grid with a cell size of 5. Using the scheme described above, object G at spatial coordinate (14,18) belongs to cell (2,3).

To answer a spatial query, the grid determines relevant cells in the manner described above and retrieves the objects they list. A query for objects within the shaded rectangle in Figure 4-2 searches cells (2,3), (2,4), (3,3), (3,4), (4,3), and (4,4), and returns objects C, F, and G. The fixed grid index is most suitable for visual memory applications with unknown distributions of object positions and frequent needs for efficient updates.

The second spatial index implemented in the prototype is a bucket PR quadtree [18]. Each node in the tree keeps a bucket of object records for some region. The index initially consists of one node covering the entire indexed region and containing

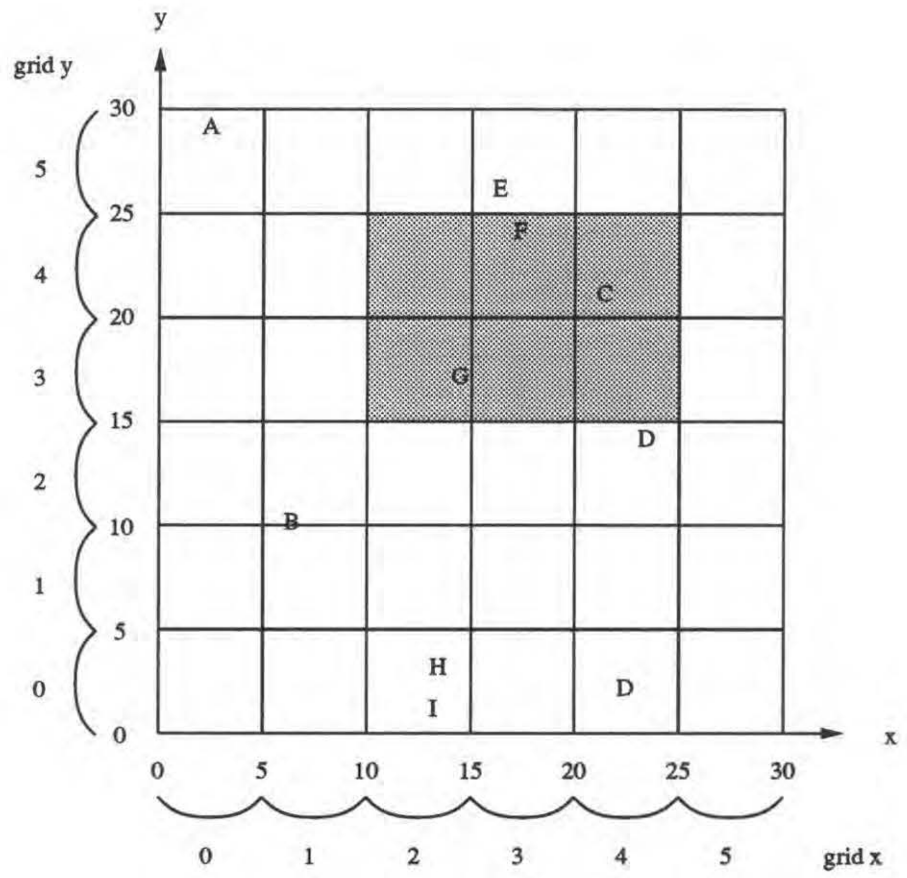


Figure 4-2: Fixed grid spatial index

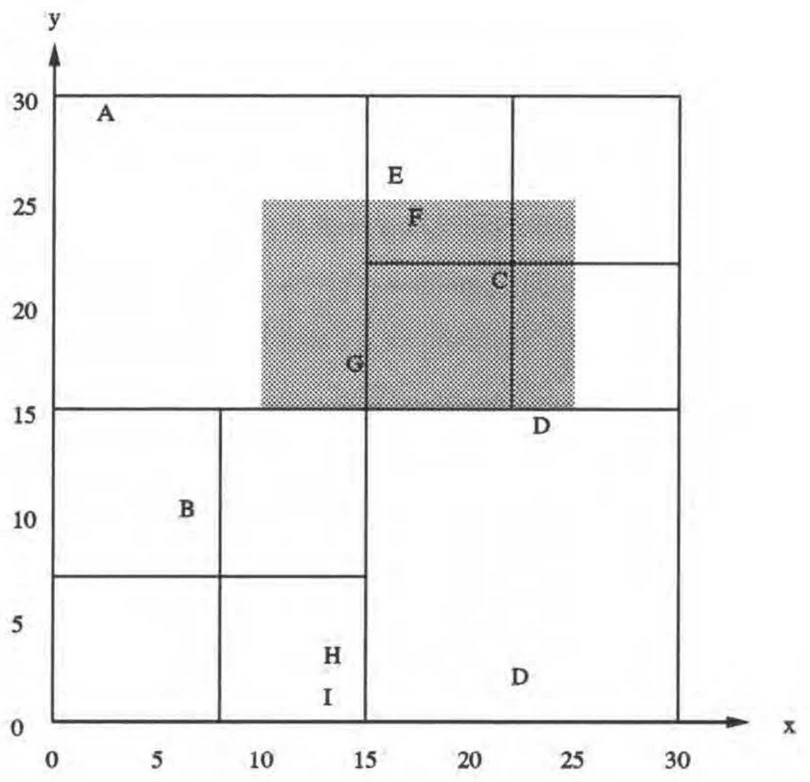


Figure 4-3: Segmented space for bucket PR quadtree

no records. As objects are added to a node's bucket, it eventually becomes full and the node must be split. A node is split into four children, one for each quadrant, and the node's bucket is appropriately divided among the children; full children are recursively split. Figure 4-3 shows how space would be segmented for a quadtree with bucket size of 2 and the given objects. Figure 4-4 shows the corresponding index structure.

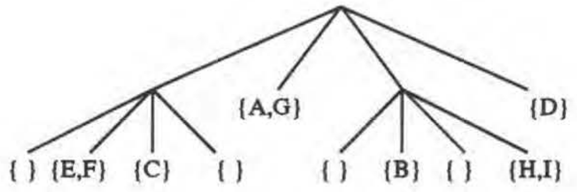


Figure 4-4: Data structure for bucket PR quadtree

A bucket PR quadtree answers a spatial query with a recursive search through all nodes intersecting the region of interest. A query for objects in the shaded rectangle in Figure 4-3 searches the left half of the tree in Figure 4-4, and it returns objects C, F, and G.

The bucket PR quadtree index is best suited for visual memory applications where objects are spread out and do not move often. In these cases, it has much less overhead than the fixed grid. Thus an application might use a quadtree to store static background information and a grid to store dynamic information.

### 4.3.3 Temporal Indices

The prototype temporal indices keep track of the valid times of object versions. They can efficiently answer temporal intersection queries, such as "Find all events that happened after work last Tuesday and Wednesday." The prototype implements two different temporal indices.

The first temporal index is a segment tree [18]. Each node in the tree represents a temporal interval and contains a list of all versions valid throughout the entire interval. The children of a node represent subintervals of their parent's interval, so that a version that is not valid throughout a node's interval can be stored in one of its descendants. For example, if version A were valid from time 35 to time 140, it would appear at the indicated nodes in Figure 4-5.

To answer a temporal intersection query, the temporal segment tree retrieves the versions referenced by all nodes with intervals intersecting the specified temporal element. To find all versions valid during [105, 118) in Figure 4-5, the index searches the darkened branches and returns versions A and E.

The second temporal index is a B+ tree [6] with times as its keys. Each leaf node maintains a start-list containing versions that become valid at the node's key time and a stop-list containing versions that stop being valid at that time. The keys in an internal node separate its children. Leaves are connected in a linked list, and the start-list for the first leaf of an internal node also indicates "carry-over" versions still valid after the last key in the previous node. In Figure 4-6, version A, valid from

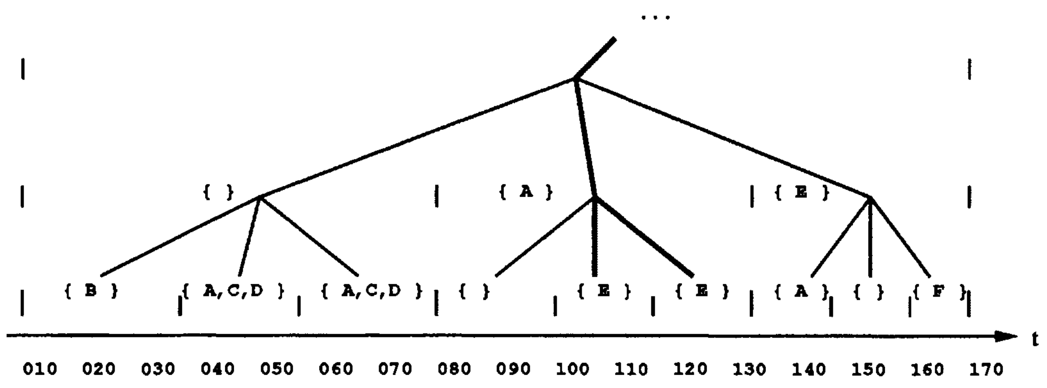


Figure 4-5: Temporal segment tree

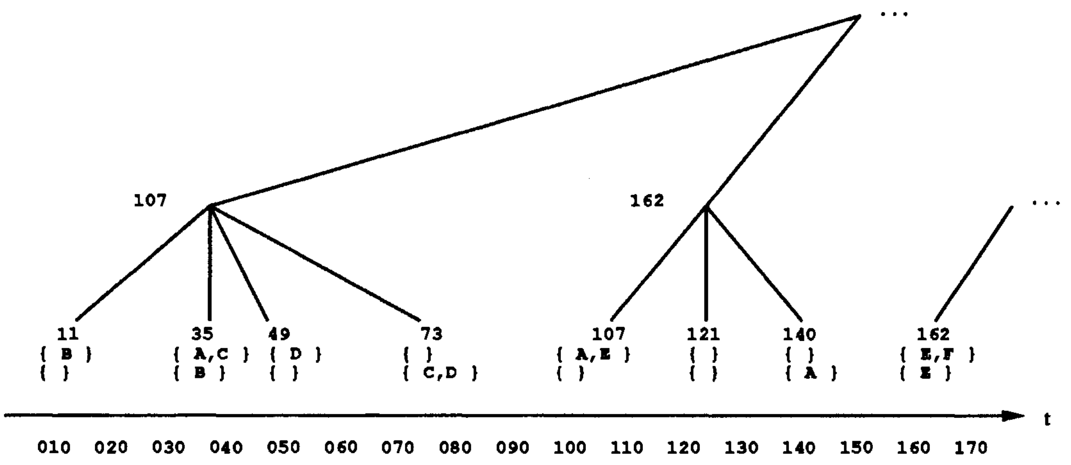


Figure 4-6: Temporal B+ tree

time 35 to time 140, has a start record at node 35, a stop record at node 140, and a carry-over record at node 107.

A temporal intersection query proceeds down the tree to the first leaf of an internal node with a time less than the earliest specified time. There it gathers the carry-over records and traverses the linked list to the earliest specified time to determine which carry-over versions are still valid then. Next it continues through the list to the latest specified time, noting which versions become valid during the temporal element. In Figure 4-6, a query for the interval [105,118) would go down to leaf node 11 and traverse the linked list to leaf node 107, noting that only version A was still valid at time 105. It would then proceed to leaf node 121 to find the remaining valid versions, finally returning versions A and E.

## 4.4 Queries

The prototype visual memory implements a functional query interface rather than a full query language. To pose a query, an application calls a visual memory function, passing it parameters specifying the query. For example, a spatial proximity query's parameters are a point and a radius, while a temporal intersection query takes a temporal element. The visual memory returns a set of <OID, version number> index records indicating objects that might satisfy the query. This set can be combined with other such sets to construct complex queries. Once a query has been fully specified, the query mechanism can retrieve the indicated objects. The indices provide only approximate answers, so the query mechanism filters the retrieved objects to return only those objects satisfying the specification. This query mechanism allows applications to pose fairly complex queries.

## 4.5 Input

The input for the scene monitoring prototype comes from real-time processing of CCD camera images. This software, which tracks people walking in its field of view, was implemented by Tom Bannon and Tom O'Donnell in the Image Understanding Branch at the Texas Instruments Computer Science Laboratory. Using a calibrated internal model of its field of view, the software estimates the positions and heights of people and updates the visual memory a few times per second. This yields enough information to test the visual memory's performance and to provide interesting data for queries to retrieve.

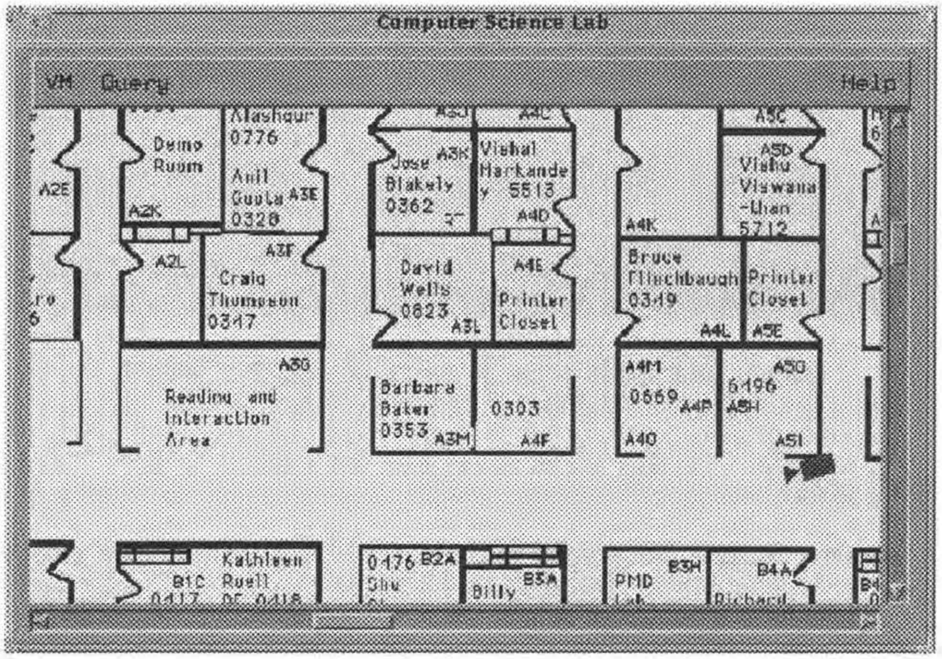


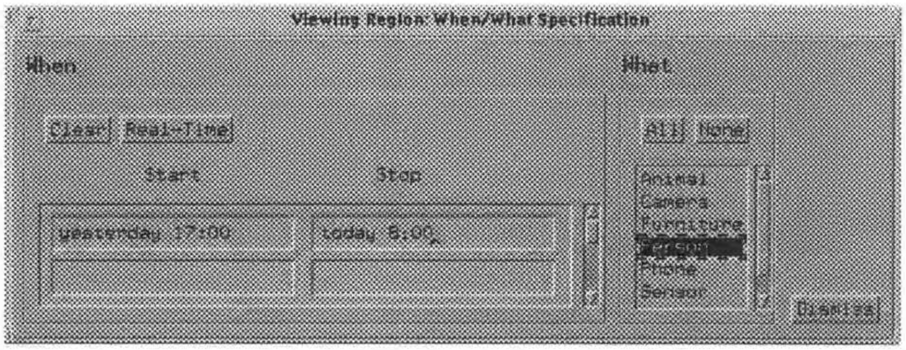
Figure 4-7: Graphical query interface viewing region

## 4.6 Graphical Query Interface

The scene monitoring prototype includes a graphical interface through which users can query the visual memory to retrieve information stored by the tracking software. A user establishes regions, times, and object types of interest, and the visual memory retrieves the corresponding objects. The query interface can display the results by dynamically stepping through the state changes of the objects, by displaying all the changes at once, or by displaying textual information about the objects.

The first step in posing a query is to select the query region. The query interface allows a user to step through a map hierarchy to select the map for the region of interest. The user can resize and scroll the query interface window to select an exact query region. This region specifies the spatial area for which objects should be retrieved. Figure 4-7 demonstrates a typical viewing region.

The next step is to establish alarm regions by shading rectangles on the map. In addition to displaying objects in the query region, the scene monitoring system alerts



| Start | Stop |
|---|---|
| yesterday 17:00 | today 8:00 |
| | |

Figure 4-8: Specification of query times and classes

the user to events in alarm regions. Alarm regions can be established all over the map, allowing the user to monitor a number of disjoint regions without having to watch them all.

Another step in query specification is to indicate a set of time intervals for each region, as shown in the left half of Figure 4-8. The system can parse times such as "3/8/93 8:00" and "today 13:00." It includes a special construct "..." to represent infinite queries retrieving all information after a given point. In addition, it provides the keyword "now" to signify a real-time query, one that constantly polls the database for new information.

An alarm region's temporal specification defaults to that of the query region. If an alarm region has an explicit temporal specification, that specification is conjoined with the query region's specification. This allows a user to specify, for example, that an alarm region should be active only during certain hours. The temporal specification for the query region identifies times of interest, and the temporal specification for an alarm region further restricts that specification to indicate exactly when the alarm should be active.

The user can specify for each region what types of objects are important, as shown in the right half of Figure 4-8. For example, the query region might return all objects, an indoor alarm region only people, and an outdoor alarm region both people and vehicles. Alarm regions default to the same type specification as the query region.

Associated with each alarm region is a delay specification that indicates how long

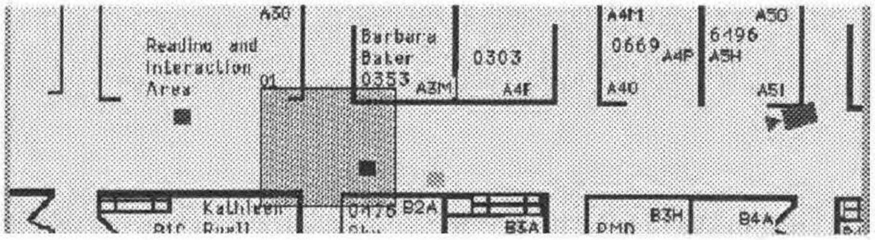


Figure 4-9: Graphical query results

an object must remain in that region before the system triggers an alarm. This lets the user specify that an alarm should fire only if an object remains in a region for a suspicious amount of time. The default value is 0 seconds, causing an alarm to be sounded as soon as an object enters the alarm region.

Once a query is fully specified, the results can be displayed in one of three ways: playback, event report, and trail trace. A playback steps through the retrieved information in temporal order, displaying moving blocks for moving objects and printing alarm information in another window. An event report textually describes alarms that were triggered. A trail trace displays blocks for all the retrieved information simultaneously and provides information about a certain object in response to a button press over its picture. Figure 4-9 shows part of a playback window, with one object inside an alarm region and two other objects also being monitored.

# Chapter 5

# Performance

One of the key requirements for the visual memory is to provide high-performance storage and retrieval of spatiotemporal information. The scene monitoring prototype described in Chapter 4 not only demonstrates the representational power of the visual memory design, it also provides a means for examining the performance of the prototype visual memory. This chapter studies some tests conducted to analyze the prototype's performance.

Visual memory performance can be measured in two main ways: by the number of objects stored and retrieved, and by the amount of time taken to store and retrieve those objects. The scene monitoring prototype is most concerned with how fast it can manipulate information, suggesting the use of temporal performance measurement. However, measuring the number of objects stored and retrieved can give an idea of the bottom-line visual memory performance and can help predict how changes in the storage and retrieval mechanism could affect the temporal performance. This chapter only discusses temporal performance, since both measurements follow approximately the same pattern and since timing measurements provide an intuitive benchmark.

The results of timing tests vary from machine to machine and from one execution to the next depending on system load, so they are most useful in providing comparative information. To reduce inaccuracy, times discussed here are the averages of three test executions. To provide more valid comparisons, the tests were run during the same time frame on a single machine with approximately the same system load.

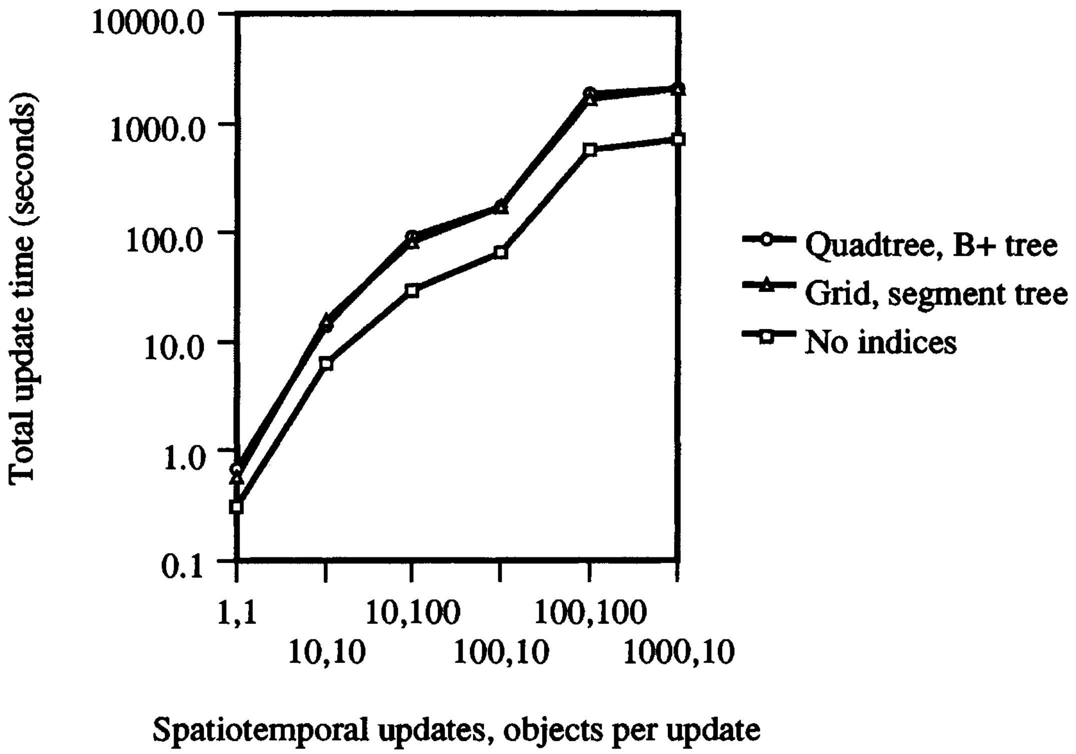


Figure 5-1: Spatiotemporal update performance

## 5.1 Spatiotemporal Object Storage and Retrieval

The prototype visual memory achieves reasonable spatiotemporal object storage performance. The underlying database limits the attainable performance, since it is responsible for actual object storage. With every spatiotemporal object update, the visual memory stores additional indexing information. A useful test of storage performance compares the time to store raw spatiotemporal objects with that to store both spatiotemporal objects and associated index information. The graph in Figure 5-1 shows storage times for spatiotemporal objects and different sets of indices as a function of the number of objects per update and the number of updates.

This graph shows that both raw storage time and indexed storage time steadily increase with the number of updates and the number of objects per update. Indexed storage costs a nearly constant factor of 2 to 3 times the raw update time. This overhead factor follows from the spatiotemporal indexing strategy discussed in Section 3.8.4, since for each update the visual memory stores spatiotemporal objects in a spatial index and the spatial index in a temporal index. While this seems to be a high price, it is necessary so that the visual memory can provide efficient spatiotemporal access to the stored information.

As a result of storing spatiotemporal index information, the visual memory can quickly answer spatiotemporal queries. Depending on indices, query complexity, and number of satisfying objects, the visual memory answered test spatiotemporal queries in 0.1 to 2.1 seconds. Clearly, retrieval performance is much better than storage performance.

## 5.2 Index Comparison

Chapter 4 describes two spatial and two temporal indices implemented in the visual memory prototype. The spatial indices can answer the same queries, but they differ in structure: the grid has a static structure built prior to execution, while the quadtree has a dynamic structure defined by the objects stored in it. Similarly, the temporal indices provide the same functionality, but the segment tree has a static structure and the B+ tree has a dynamic structure. The visual memory prototype provides a basis for comparing the performance of these indices.

Parameters such as branching factor and cell size affect index structure, so the tests must use comparable parameters. The spatial tests cover a 100-unit by 100-unit square. The quadtree has a bucket size of 10 objects and the fixed grid has a cell size of 10 units; this implies that the grid has 100 nodes and the quadtree has from 1 to a few hundred nodes. The temporal tests cover a time interval of up to 1000 seconds, and both temporal indices have a branching factor of 64.

In addition to the indices described above, each test also includes a "bucket" index. A bucket index simply maintains a list of all the objects stored in the visual memory. Since there is no overhead for the storage of complex index structure, a bucket index can achieve the highest update performance. A bucket index answers a query by retrieving all the objects in its list and checking them against the query specification.

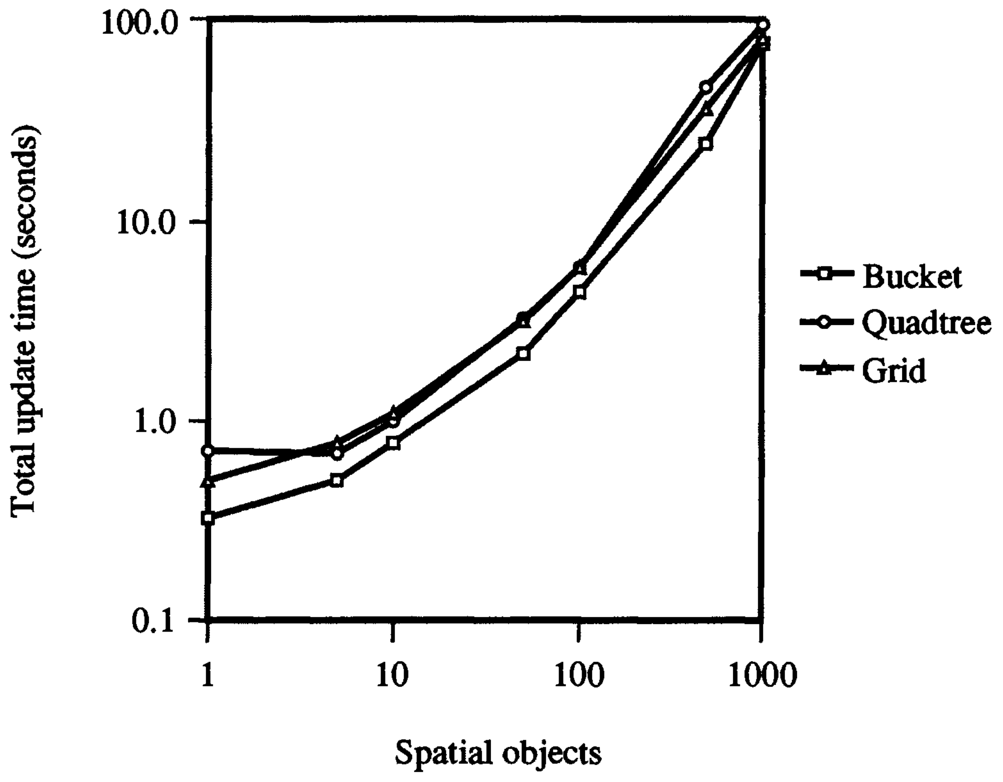


Figure 5-2: Spatial update performance

This is not an efficient query mechanism for large queries, but it provides a useful basis for comparing the performance of other indices.

One important performance measure compares how quickly indices can update information about objects. Figure 5-2 shows the update performance of spatial indices, and Figure 5-3 shows the update performance of temporal indices.

As expected, the bucket indices achieve the highest performance for small numbers of objects. However, the temporal bucket cannot store much more than 100 updates, since it saves an entire list with each update and quickly fills the database. Dynamic structures tend to perform slightly better than static structures for small numbers of objects, while static structures are better for large numbers of objects. This follows from the relative sizes of the structures; dynamic indices are initially small but grow as they store information about additional objects, while static indices maintain the same structure no matter how much information is stored.

Another important measure for index comparison is query performance. Timing

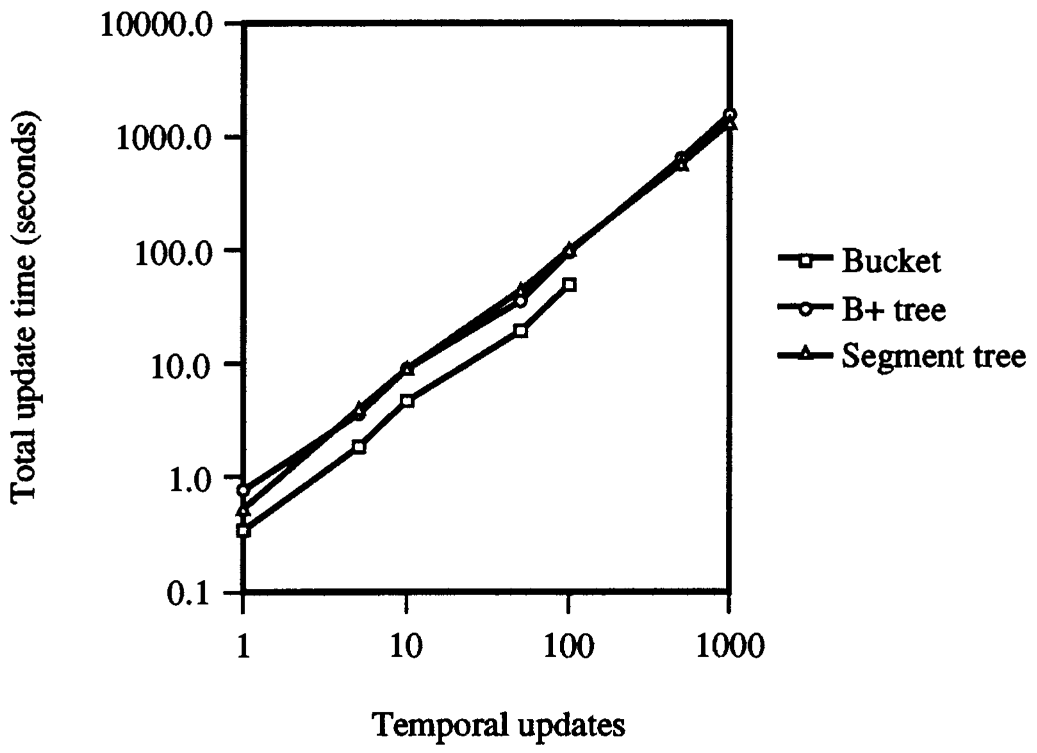


Figure 5-3: Temporal update performance

tests show that query performance follows a pattern similar to that of update performance: bucket indices achieve the best performance with small numbers of objects, dynamic structures work better than static structures with small numbers of objects, and static structures work better than dynamic structures with large number of objects. Figure 5-4 shows the performance for spatial indices with a 10-unit by 10-unit query square. Figure 5-5 shows the performance for temporal indices with a query interval of 10 seconds.

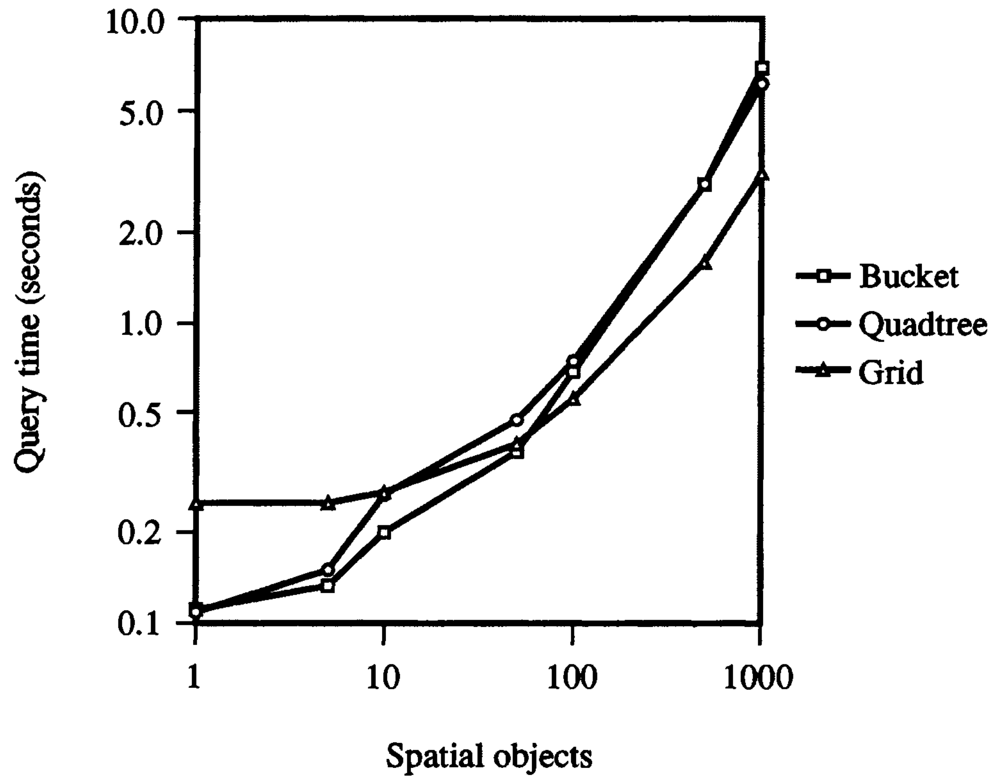


Figure 5-4: Spatial query performance

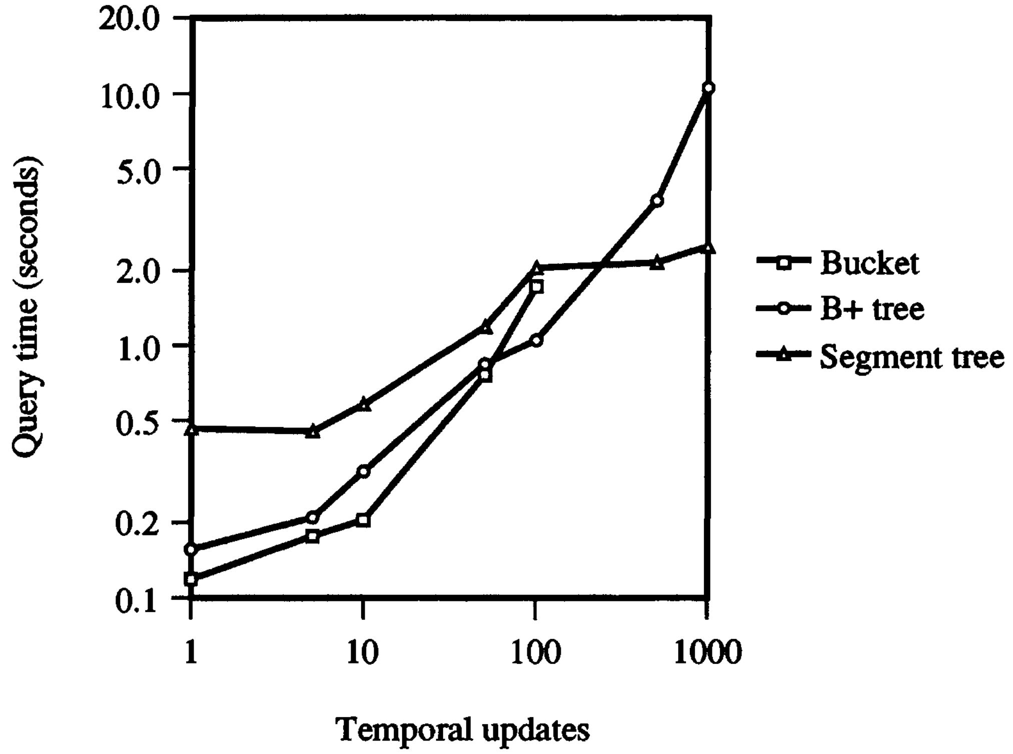


Figure 5-5: Temporal query performance

# Chapter 6

# Conclusion

The visual memory design presented in this thesis combines and extends spatial, temporal, and database research to meet the needs of a number of computer vision applications. It provides powerful and expressive spatiotemporal representations that it can efficiently manipulate, store, and retrieve. A prototype visual memory implemented in support of a scene monitoring prototype demonstrates the potential of this design. This prototype achieves useful storage and query performance and provides a basis for comparison of different indices.

Visual memory research could continue in many different directions. One step is to more fully implement the design. Some of the unimplemented spatiotemporal representations, such as probabilistic, relative, and abstract objects, could be beneficial to the scene monitoring prototype. The prototype visual memory could be connected to a number of different computer vision applications. Further implementation and testing would provide more feedback on the design and help identify areas for additional research.

The visual memory could furnish additional functionality if it used a different database. For example, if the database provided active rules, a security system could establish visual memory daemons to automatically check for alarms and to resolve old data. If the database provided real-time guarantees, a vehicle navigator could be sure that it would not crash because of visual memory performance. Finally, if the database provided data partitioning capabilities, applications that store large amounts of spatiotemporal data could make use of separate storage devices.

A number of extensions could improve the performance of the visual memory. Visual memory customization of caching and look-ahead could improve both storage and retrieval performance. Lightweight transactions could reduce overhead and increase storage performance for applications that continuously update the visual memory. Query optimization could increase retrieval performance by ordering parts of a query to reduce the number of retrievals. These extensions could help the visual memory reach its potential as high-performance system for manipulating spatiotemporal information.

# Bibliography

[1] American National Standard for Information Systems. Database language SQL. Technical Report ANSI-X3.138-1986, American National Standards Institute, October 1986.

[2] José A. Blakeley. ZQL[C++]: Extending a persistent C++ language with a query capability. Technical Report 91-06-01, Texas Instruments Information Technologies Laboratory, 1991.

[3] John Brolio et al. ISR: A database for symbolic processing in computer vision. *IEEE Computer*, December 1989.

[4] M. Carey, D. DeWitt, and E. Shekita. Storage management for objects in EXODUS. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley Publishing Company, 1989.

[5] Surajit Chaudhuri. Temporal relationships in databases. In *Proceedings of the Conference on Very Large Databases*, 1988.

[6] Douglas Comer. The ubiquitous B-tree. *ACM Computing Surveys*, June 1979.

[7] Randall Davis, Bruce Buchanan, and Edward Shortliffe. Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, February 1977.

[8] Ramez Elmasri, Ihab El-Assal, and Vram Kouramajian. Semantics of temporal data in an extended ER model. In *Proceedings of the Entity-Relationship Conference*, October 1990.

[9] Ramez Elmasri, Muhammad Jaseemuddin, and Vram Kouramajian. Partitioning of time index for optical disks. In *IEEE Data Engineering Conference*, February 1992.

[10] Ramez Elmasri and G. Wiederhold. GORDAS: A formal high-level query language for the ER model. In *Proceedings of the Entity-Relationship Conference*, October 1981.

[11] James D. Foley and Andries Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley Publishing Company, 1984.

[12] G. D. Held, M. R. Stonebraker, and E. Wong. INGRES: A relational data base system. In *Proceedings of the National Computer Conference*, May 1975.

[13] Vram Kouramajian and Ramez Elmasri. Support for uncertainty in a generalized temporal model. Available from the authors at {kouramaj,elmasri}@cse.uta.edu.

[14] Jon Lukasiewicz. Numerical interpretation of the theory of propositions. In Borkowski, editor, *Jon Lukasiewicz: Selected Works.* North-Holland Publishing Company, 1970.

[15] Frank Manola and Jack A. Orenstein. Toward a general spatial data model for an object-oriented DBMS. In *Proceedings of the Conference on Very Large Databases*, August 1986.

[16] J. Mundy et al. *The Image Understanding Environments Program*, June 1992.

[17] Edward Perez and Robert W. Peterson. Zeitgeist Persistent C++ user manual. Technical Report 90-07-02, Texas Instruments Information Technologies Laboratory, February 1992.

[18] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company, 1989.

[19] Arie Segev and Arie Shoshani. Logical modeling of temporal data. In *Proceedings of the ACM SIGMOD Conference*, June 1987.

[20] Steven A. Shafer, Anthony Stentz, and Charles E. Thorpe. An architecture for sensor fusion in a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1986.

[21] Richard Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, June 1989.

[22] Michael D. Soo. Bibliography on temporal databases. *ACM SIGMOD Record*, March 1991.

[23] Michael Stonebraker and Lawrence A. Rowe. The design of POSTGRES. In *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, 1986.

[24] Thomas M. Strat and Grahame B. Smith. Core Knowledge System: Storage and retrieval of inconsistent information. In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.

[25] David L. Wells, José A. Blakely, and Craig W. Thompson. Architecture of an open object-oriented database management system. *IEEE Computer*, October 1992.

## Full Record

**Permalink for this record:** http://library.mit.edu/item/000656176

**Results List** | **Add to Bookshelf** | **Save/Email**

Choose format: **Standard** | **Citation** | **MARC tags**

**Record 1 out of 1**

| | |
|---|---|
| **Author** | Kollogg, Christopher James. |
| **Title** | Visual memory / by Christopher James Kellogg. |
| **Shelf Access** | Find it in the library/Request item |
| **Shelf Location** | Institute Archives - Noncirculating Collection 1 \| Thesis E.E. 1993 M.S. |
| **Shelf Location** | Institute Archives - Noncirculating Collection 3 \| Thesis E.E. 1993 M.S. |
| **Shelf Location** | Barker Library - Microforms \| Thesis E.E. 1993 M.S. |
| **Published** | c1993. |
| **Description** | 92 leaves : ill. ; 29 cm. |
| **Format** | Book |
| **Thesis Note** | Thesis (M.S.)--Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1993. |
| **Thesis Supervisor** | Supervised by Alex P. Pentland. |
| **Bibliography** | Includes bibliographical references (leaves 90-92). |
| **Local System Number** | **000656176** |

## Full Record

**Permalink for this record:** http://library.mit.edu/item/000656176

**Results List** | **Add to Bookshelf** | **Save/Email**

Choose format: **Standard** | **Citation** | **MARC tags**

**Record 1 out of 1**

| | |
|---|---|
| **FMT** | BK |
| **LDR** | 00968nam  2200265K  45q0 |
| **003** | MCM |
| **005** | 20010608211241.0 |
| **008** | 930928s1993    xx a     b    000 0 eng d |
| **035** | \|a MITb10656176 |
| **035** | \|a (OCoLC)28904769 |
| **035** | \|a GLIS00656176 |
| **040** | \|a MYG \|c MYG |
| **099** | \|a Thesis E.E. 1993 M.S. \|a Thesis E.E. 1993 M.S. Mfch |
| **099** | \|a Thesis \|a E.E. \|a 1993 \|a M.S. |
| **1001** | \|a Kollogg, Christopher James. |
| **24510** | \|a Visual memory / \|c by Christopher James Kellogg. |
| **260** | \|c c1993. |
| **300** | \|a 92 leaves : \|b ill. ; \|c 29 cm. |
| **502** | \|a Thesis (M.S.)--Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1993. |
| **504** | \|a Includes bibliographical references (leaves 90-92). |
| **59900** | \|a Supervised by Alex P. Pentland. |
| **599** | \|a GRSN 656176 |
| **CAT** | \|a CONV \|b 00 \|c 20010620 \|l MIT01 \|h 1525 |
| **CAT** | \|a lti0904 \|b 00 \|c 20090523 \|l MIT01 \|h 2117 |
| **049** | \|a MYGG |
| **910** | \|a jlh |
| **949** | \|a MYTT \|b 39080008599745 \|a [Micro-] [fiche] MYTT \|b 39080008664614 \|a MYTE \|b 39080008666296 \|a [Micro-] [fiche] MYTE \|b 39080008664663 |
| **PST8** | \|0 Z30 \|1 000656176000020 \|b ARC \|c NOLN1 \|o BOOK \|d 02 \|y 00000 \|f N \|r MIT60-000584976 \|n 8 \|h Thesis E.E. 1993 M.S. \|k THESIS \|a MCM \|3 Book \|4 Institute Archives \|5 Noncirculating Collection 1 \|6 Room Use Only \|p Avail |
| **PST8** | \|0 Z30 \|1 000656176000010 \|b ARC \|c NOLN3 \|o BOOK \|d 02 \|y 00000 \|f N \|r MIT60-000579310 \|n 8 \|h Thesis E.E. 1993 M.S. \|k THESIS \|a MCM \|3 Book \|4 Institute Archives \|5 Noncirculating Collection 3 \|6 Room Use Only |
| **PST8** | \|0 Z30 \|1 000656176000030 \|b ENG \|c MFORM \|o BOOK \|d 12 \|y 00000 \|f N \|r MIT60-000584981 \|n 8 \|h Thesis E.E. 1993 M.S. \|k THESIS \|a MCM \|3 Book \|4 Barker Library \|5 Microforms \|6 Thesis Loan |
| **LDR** | nx   22    zn 4500 |
| **008** | 0106230u    0    4    uu    1 |
| **004** | 000656176 |
| **8528** | \|a MCM \|b ARC \|c NOLN3 \|h Thesis E.E. 1993 M.S. \|k THESIS \|z |
| **LDR** | nx   22    zn 4500 |
| **008** | 0106230u    0    4    uu    1 |
| **004** | 000656176 |
| **8528** | \|a MCM \|b ARC \|c NOLN1 \|h Thesis E.E. 1993 M.S. \|k THESIS \|z |
| **LDR** | nx   22    zn 4500 |
| **008** | 0106230u    0    4    uu    1 |
| **004** | 000656176 |
| **8528** | \|a MCM \|b ENG \|c MFORM \|h Thesis E.E. 1993 M.S. \|k THESIS \|z |

| | |
|---|---|
| **001** | 000656176 |
| **SFX01** | \|s 0-0-0-6-5-6-1-7-6 \|l MIT01 \|9 000 \|z ~~~~~~~~~~ \|p Avail \|f 000 |
| **LU564** | BK |
| **SYS** | 000656176 |

---

**Basic Search of Full Catalog**

Search type:

Keyword
Title begins with...
Title Keyword
Author (last name first)
Author Keyword
Call Number begins with...
----- Scroll down for more choices -----

Search for:

Search

Barton Questions: Ask Us! | Contact Us
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA

-- Quick Links --

# Image Understanding Workshop

## Proceedings of a Workshop held in Monterey, California

## November 20–23, 1998

## Volume I

**Sponsored by:**
**Defense Advanced Research Projects Agency**
**Information Systems Office**

This document contains copies of reports prepared for the DARPA Image Understanding Workshop. Included are Principal Investigator reports and technical results from the basic and strategic computing programs within DARPA/ISO-sponsored projects and certain technical reports from selected scientists from other organizations.

**Distributed by:**
**Morgan Kaufmann Publishers Inc.**
**340 Pine Street, 6th Floor**
**San Francisco, Calif. 94104-3205**
**ISBN: 1–55860–583–5**
**Printed in the United States of America**

# Table of Contents

## Volume I

# Event Recognition and Reliability Improvements for the Autonomous Video Surveillance System

**Frank Z. Brill, Thomas J. Olson, and Christopher Tserng**
Texas Instruments
P.O. Box 655303, MS 8374, Dallas, TX 75265
brill@csc.ti.com, olson@csc.ti.com, tserng@csc.ti.com

## Abstract

This report describes recent progress in the development of the Autonomous Video Surveillance (AVS) system, a general-purpose system for moving object detection and event recognition. AVS analyses live video of a scene and builds a description of the activity in that scene. The recent enhancements to AVS described in this report are: (1) use of collateral information sources, (2) camera hand-off, (3) vehicle event recognition, and (4) complex-event recognition. Also described is a new segmentation and tracking technique and an evaluation of AVS performing the best-view selection task.

## 1. Introduction

The Autonomous Video Surveillance (AVS) system processes live video streams from surveillance cameras to automatically produce a real-time map-based display of the locations of people, objects and events in a monitored region. The system allows a user to specify alarm conditions interactively, based on the locations of people and objects in the scene, the types of objects in the scene, the events in which the people and objects are involved, and the times at which the events occur. Furthermore, the user can specify the action to take when an alarm is triggered, e.g., to generate an audio alarm or write a log file. For example, the user can specify that an audio alarm should be triggered if a person deposits a briefcase on a given table between 5:00pm and 7:00am on a weeknight. Section 2 below describes recent enhancements to the AVS system. Section 3 describes progress in improving the reliability of segmentation and tracking. Section 4 describes an experiment that quantifies the performance of the AVS "best view selection" capability.

This research was sponsored in part by the DARPA Image Understanding Program.

## 2. New AVS functionality

The structure and function of the AVS system is described in detail in a previous IUW paper [Olson and Brill, 1997]. The primary purpose of the current paper is to describe recent enhancements to the AVS system. These enhancements are described in four sections below: (1) collateral information sources, (2) camera hand-off, (3) vehicle event recognition, and (4) complex-event recognition.

### 2.1. Collateral information sources

Figure 1 shows a diagram of the AVS system. One or more "smart" cameras process the video stream to recognize events. The resulting event streams are sent to a Video Surveillance Shell (VSS), which integrates the information and displays it on a map. The VSS can also generate alarms based on the information in the event streams. In recent work, the VSS was enhanced to accept information from other sources, or "recognition devices" which can identify the objects being reported on by the cameras. For example, a camera may report that there is a person near a door. A recognition device may report that the person near the door is Joe Smith. The recognition device may be a badge reader, a keypad in which a person types their PIN, a face recognition system, or other recognition system.

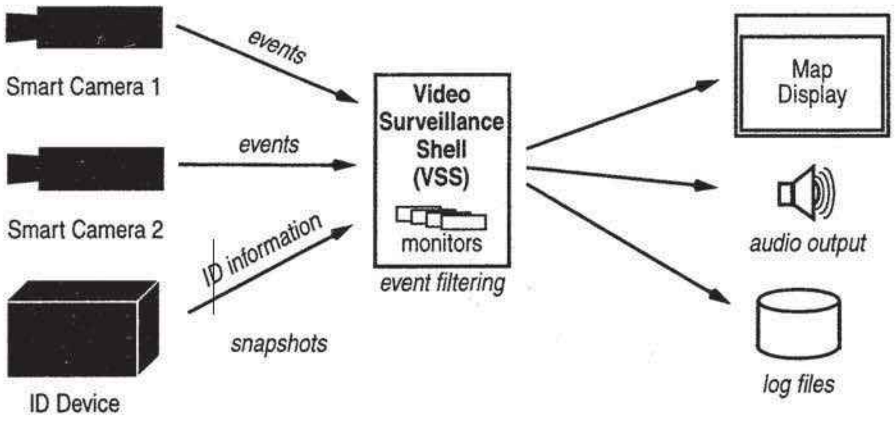


Figure 1: AVS system diagram

The recognition device we have incorporated is a voice verification system. The user stands in a predefined location in the room, and speaks his or her name. The system matches the utterance to previously captured examples of the person speaking their name, and reports to the VSS if there is a match. The VSS now knows the identity of the person being observed, and can customize alarms based on the person's identity.

A recognition device could identify things other than people, and could classify actions instead of objects. For example, the MIT Action Recognition System (MARS) recognizes actions of people in the scene, such as raising their arms or bending over. MARS is trained by observing examples of the action to be recognized and forming "temporal templates" that briefly describe the action [Davis and Bobick, 1997]. At run time, MARS observes the motion in the scene and determines when the motion matches one of the stored temporal templates. TI has obtained an evaluation copy of the MARS software and used it as an recognition device which identifies actions, and sends the result to the AVS VSS. We successfully trained MARS to recognize the actions of opening a door, and opening the drawer of a file cabinet. When MARS recognizes these actions, it sends a message to the AVS VSS, which can generate an appropriate alarm.

## 2.2. Camera hand-off

As depicted in Figure 1, the AVS system incorporates multiple cameras to enable surveillance of a wider area than can be monitored via a single camera. If the fields of view of these cameras are adjacent, a person can be tracked from one monitored area to another. When the person leaves the field of view of one camera and enters another, the process of maintaining the track from one camera view to another is termed *camera hand-off*. Figure 2 shows an area monitored by two cameras. Cam-

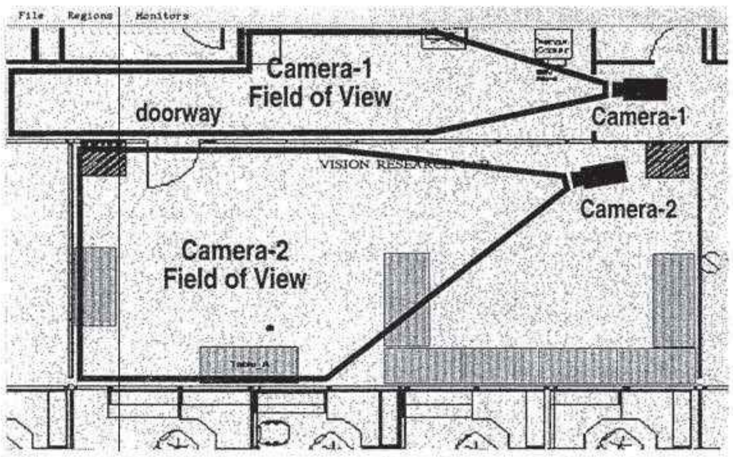


Figure 2: Multiple cameras with adjacent fields of view

Ex. D Page 5 of 20

era-1 monitors the hallway, and Camera-2 monitors the interior of the room. When a person moves through the doorway to enter the room from the hall or vice-versa, camera hand-off is necessary to enable the system to know that the person that was being monitored in the hall via Camera-1 is the *same* as the person being monitored in the room via Camera-2.

The AVS system accomplishes camera hand-off by integrating the information from the two cameras in the map coordinate system. The AVS "smart" cameras report the locations of the monitored objects and people in map coordinates, so that when the VSS receives reports about a person from two separate cameras, and both cameras are reporting the person's coordinates at about the same map location, the VSS can deduce that the two separate reports refer to the same person. In the example depicted in Figure 2, when a person is standing in the doorway, both cameras can see the person and report his or her location at nearly the same place. The VSS reports this as one person, using a minimum distance to allow for errors in location. When Camera-2 first sees a person at a location near the doorway and reports this to the VSS, the VSS checks to see if Camera-1 recently reported a person near the door. If so, the VSS reports the person in the room as the same one that Camera-1 had been tracking in the hall.

## 2.3. Vehicle event recognition

This section describes extensions to the existing AVS system that enable the recognition of events involving interactions of people with cars. These new capabilities enable smart security cameras to monitor streets, parking lots and driveways and report when suspicious events occur. For example, a smart camera signals an alarm when a person exits a car, deposits an object near a building, reenters the car, and drives away.

### 2.3.1. Scope and assumptions

Extending the AVS system to handle human-vehicle interactions reliably involved two separable subproblems. First, the system's vocabulary for events and objects must be extended to handle a new class of object (vehicle) and new event types. Second, the AVS moving object detection and tracking software must be modified to handle the outdoor environment, which features variable lighting, strong shadows, atmospheric disturbances, and dynamic backgrounds. The work described here in section 2.3 addresses the first problem, to extend the system for vehicle events in conditions of uniform overcast with little wind. Our approach to handling general outdoor lighting conditions is discussed in section 4.

The method is further specialized for imaging conditions in which:

1. The camera views cars laterally.
2. Cars are unoccluded by other cars.
3. When cars and people overlap, only one of the overlapping objects is moving
4. The events of interest are people getting into and out of cars.

### 2.3.2. Car detection

The first thing that was done to expand the event recognizing capability of the current system was to give the system the ability to distinguish between people and cars. The system classifies objects as cars by using their sizes and aspect ratios. The size of an object in feet is obtained using the AVS system's image coordinate to world coordinate mapping. Once the system has detected a car, it analyzes the motion graph to recognize new events.

### 2.3.3. Car event recognition

In principle, car exit and car entry events could be recognized by detecting characteristic interactions of blobs in difference images, in a manner similar to the way AVS recognizes DEPOSIT and REMOVE events. In early experiments, however, this method turned out to be unsatisfactory because the underlying motion segmentation method did not segment cars from people. Whenever the people pass near the car they appear to merge with it, and track is lost until they walk away from it.

To solve this problem, a new approach involving additional image differencing was developed. The technique allows objects to be detected and tracked even when their images overlap the image of the car. This method requires two reference images: one consists of the original background scene (background image), and the other is identical to the first except it includes the car. The system takes differences between the current video image and the original reference image as usual. However, it also differences the current video image with the reference image containing the car. This allows the system to detect objects which may be overlapping the car. Using this technique, it is easy to detect when people enter and exit a car. If an object disappears while overlapping with a car, it probably entered the car. Similarly, if an object appears overlapping a car, it probably exited the car.

### 2.3.4. Basic method

When a car comes to rest, the following steps are taken. First, the image of the car object is removed from its frame and stored. Then, the car image is merged with the background image, creating an updated reference image containing the car. (Terminology: a *reference car image* is the subregion of the updated reference image that contains the car.) Then, the *car background image*, the region of the original background image that is replaced by the car image, is stored.

For each successive frame, two difference images are generated. One difference image, the *foreground difference image*, is calculated by differencing the current video image with the updated reference image. The foreground difference image will contain all the blobs that represent objects other than the car, including ones that overlap the car. The second difference image, the *car difference image*, is calculated using the car background image. The car difference image is formed from the difference between the current frame and the car background image, and contains the large blob for the car itself. Figures 3 and 4 show the construction and use of these images.

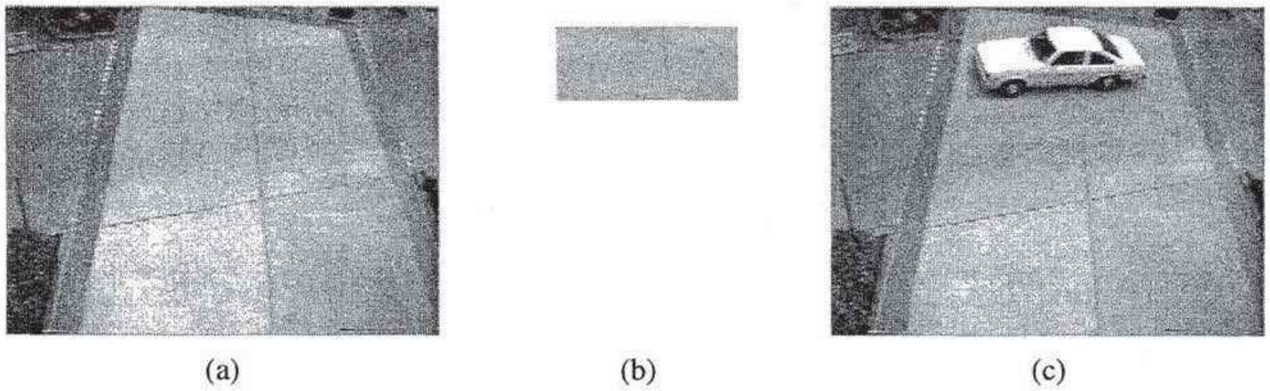


Figure 3: (a) Background image. (b) Car background image.
(c) Updated reference image

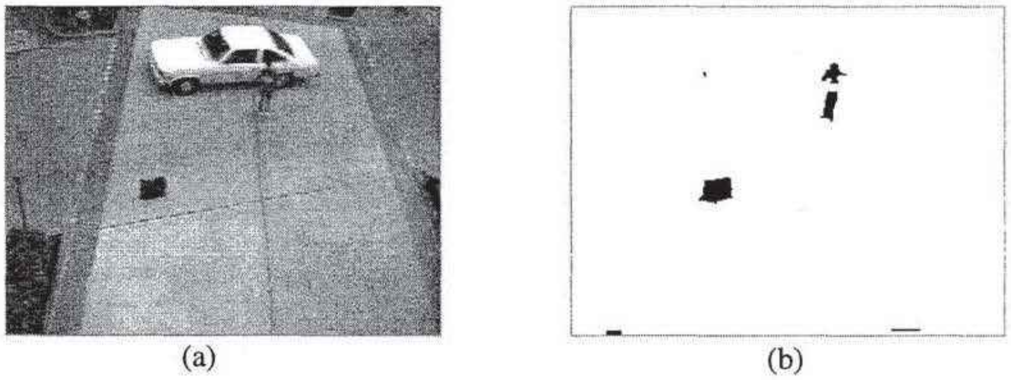


Figure 4: (a) Current video image. (b) Foreground difference image

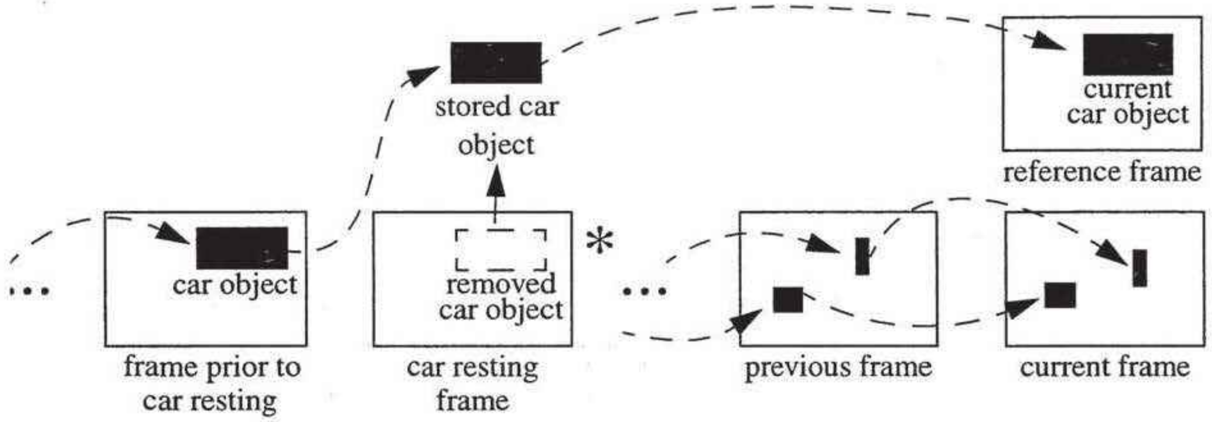


Figure 5: Creation of the motion graph.
The starred frame represents the frame prior to the background image being updated.

The blobs in the foreground difference image are grouped into objects using the normal grouping heuristics and placed in the current frame. The blobs in the car difference image necessarily represent the car, so they are all grouped into one current car object and placed in a special *reference frame*. Normal links occur between objects in the previous frame and objects in the current frame. Additionally, the stored car object, which was removed from its frame, (from Step 1) is linked to the current car object which is in the reference frame. In any given sequence, there is only one reference frame.

Figure 5 demonstrates the creation of this new motion graph. As indicated by the dotted lines, all objects maintain their tracks using this method. Notice that even though the car object disappears from future frames (due to the updated reference image), it is not detected to have exited because its track is maintained throughout every frame. Using this method, the system is able to keep track of the car object as well as any objects overlapping the car. If an object appears intersecting a car object, an INCAR event is reported. If an object disappears while intersecting a car object, an OUTCAR event is reported. Figure 6 shows the output of the system. The system will continue to operate in this manner until the car in the reference frame begins to move again.

When the car moves again, the system reverts to its normal single-reference-image state. The system detects the car's motion based on the movement of its centroid. It compares the position of the centroid of the stored car object with the centroid of the current car object. Figure 7 shows the slight movement of the car.

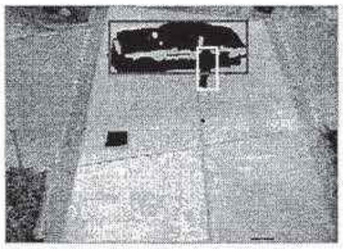


Figure 6: Final output of system

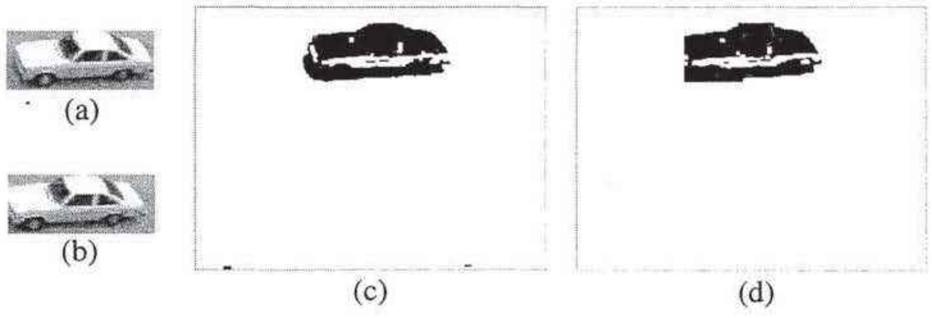


Figure 7: (a) Reference car image. (b) Moving car image.
(c) Reference car difference image. (d) Moving car difference image

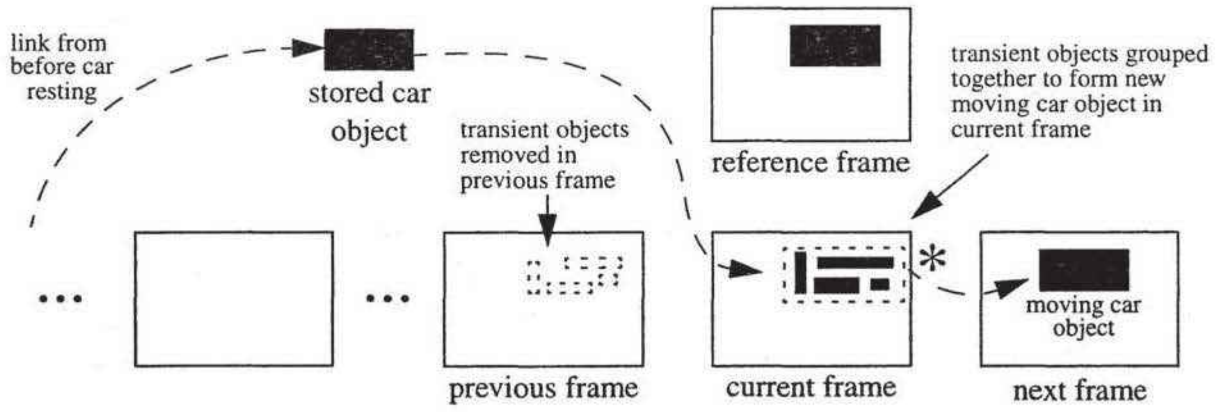

link from before car resting
stored car object
transient objects removed in previous frame
reference frame
transient objects grouped together to form new moving car object in current frame
*
moving car object
previous frame
current frame
next frame

Figure 8: Restoration of normal differencing. The starred frame represents the last frame prior to the original reference image being restored.

If the centroid locations differ by more than a threshold, the following sequence of events occur to restore the system to its original state:

1. An object representing the moving car is created in the current frame.
2. The stored car object is linked to this new moving car object in the current frame.
3. Objects in the previous frame that intersect the moving car are removed from that frame.
4. The car background image is merged with the updated reference image to restore the original reference image.
5. Normal differencing continues.

Figure 8 demonstrates how the system is restored to its original state. Note that there is one continuous track that represents the path of the car throughout.

When the car begins to move again, transient blobs appear in the foreground difference image due to the fact that the car is in the updated reference image as seen in Figure 9. Therefore, to create a new moving car object in the current frame, these transient objects, which are identified by their intersection with the location of the resting car, are grouped together as one car object. If there are no transient objects, a copy of the stored car object is inserted into the current frame. This way, there is definitely a car object in the current frame to link with the stored car object. Transient objects might also appear in the previous frame when a car is moving. Therefore, these transient objects must be removed from their frame in order to prevent them from being linked to the new moving car object that was just created in the current frame. After the steps described above occur, the system continues as usual until another car comes to rest.

### 2.3.5. Experiments: disk-based sequences

To test the principles behind the modified AVS system, three sequences of video that represented interesting events were captured to disk. These sequences represented events which the modified system should be able to recognize. Capturing the sequences to disk reduces noise and ensures that the system processes the same frames on every run, making the results deterministic. In addition to these sequences, longer sequences were recorded and run directly from videotape to test how the system would work under less ideal conditions.

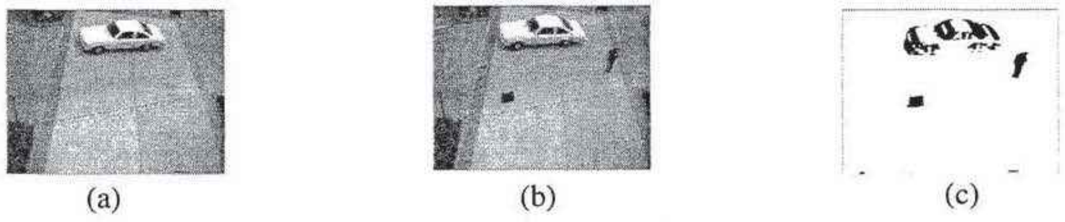


Figure 9: (a) Updated reference image. (b) Current video image. (c) Foreground difference image

*2.3.5.1. Simple sequence.* The first sequence was filmed from the 3rd story of an office building overlooking the driveway in front of the building. A car drives up and a person exits the car, walks away, deposits a briefcase, and finally reenters the car. Then, the car drives away. In this segment, the system successfully detects the person exiting the car. However, the person entering the car is missed because the person gets grouped with a second person walking near the car.

Further on in the sequence, the car drives up again and a person exits the car, walks away, removes the briefcase, and finally reenters the car. Again, the car drives away. In this segment, both the person entering and exiting the car are recognized. In both these sequences, there was only the one false negative mentioned earlier and no false positives.

*2.3.5.2. Pickup sequence.* This sequence was filmed in front of a house looking at the street in front of the house. In the sequence, a person walks into the scene and waits at the curb. A car drives up, picks up the person, and drives away. The system correctly detects the person entering the car. There are no false positives or negatives.

*2.3.5.3. Drop off sequence.* This sequence was filmed in the same location as the previous one. In this sequence, a car drives up and a person is dropped off. The car drives away with the person still standing in the same location. Then, the person walks off. The system correctly detects the person exiting the car and does not report a false enter event when the car moves away.

### 2.3.6. Experiments: videotaped sequences

These sequences were run on the system straight from videotape. These were all run at a higher threshold to accommodate noise on the videotape. However, this tended to decrease the performance of the system.

*2.3.6.1. Dark day.* This is a 15 minute sequence that was recorded from the 3rd floor of a building on a fairly dark day. In that time span, 8 cars passed through the camera's field of view. The system detected 6 cars correctly and one false car (due to people grouped together). One car that was not detected was due to its small size. The other car was undetected because the system slowed down (due to multiple events occurring) and missed the images with the car in them. In this sequence, two people entered a car. However, both events were missed because the car was not recognized as resting due to the dark lighting conditions on this rainy day.

*2.3.6.2. Cloudy day.* This is a 13 minute sequence in the same location as the previous sequence except it is a cloudy day. In this time span, 9 cars passed through the camera's field of view and all of them were detected by the system. There were a total of 2 people entering a car and 2 people exiting a car. The system successfully detected them all. Additionally, it incorrectly reported one person walking near a car as an instance of a person exiting a car.

*2.3.6.3. Cloudy day—extended time.* This is a 30 minute sequence in the same location as the previous two. In this time span, 28 cars pass through and all of them were detected. The system successfully detected one person exiting a car but missed two others. The two people were missed because the car was on the edge of the camera's field of view and so it was not recognized immediately as a car.

### 2.3.7. Evaluation of car-event recognition

The modified AVS system performs reasonably well on the test data. However, it has only been tested on a small number of videotaped sequences, in which much of the action was staged. Further experiments and further work with live, uncontrolled data will be required to make the system handle outdoor vehicle events as well as it handles indoor events. The technique of using multiple reference images is interesting and can be applied to other problems, e.g. handling repositioned furniture in indoor environments. For more detail on this method, see [Tserng, 1998].

## 2.4. Complex events

The AVS video monitoring technology enables the recognition of specific events such as when a person enters a room, deposits or picks up an object, or loiters for a while in a given area. Although these events are more sophisticated than those detected via simple motion detection, they are still unstructured events that are detected regardless of the context in which they occur. This can result in alarms being generated on events that are not of interest.

For example, if the system is monitoring a room or store with the intention of detecting theft, the system could be set up to generate an alarm whenever an object is picked up (i.e., whenever a REMOVE event occurs). However, no theft has occurred unless the person leaves the area with the object. A simple, unstructured event recognition system would generate an alarm every time someone picked up an object, resulting in many false alarms; whereas a system that can recognize complex events could be programmed to only generate an alarm when the REMOVE event is followed by an EXIT event. The EXIT event provides context for the REMOVE event that enables the system to filter out uninteresting cases in which the person does not leave the area with the object they picked up. This section describes the design and implementation of such a complex-event recognition system.

We use the term *simple event* to mean an unstructured atomic event. A *complex event* is structured, in that it is made up of one or more *sub-events*. The sub-events of a complex event may be simple events, or they may be complex, enabling the definition of event hierarchies. We will simply say *event* to refer to an event that may be either simple or complex. In our theft example above, REMOVE and EXIT are simple events, and THEFT is a complex event. A user may also define a further event, e.g., CRIME-SPREE, which may have one or more complex THEFT events as sub-events.

We created a user interface that enables definition of a complex event by constructing a list of sub-events. After one or more complex events have been defined, the sub-events of subsequently defined complex events can be complex events themselves.

### 2.4.1. Complex-event recognition

Once the user has defined the complex events and the actions to take when they occur, the event recognition system recognizes these events as they occur in the monitored area. For the purposes of this section, we assume *a priori* that the simple events can be recognized, and that the object involved in them can be tracked. In the implementation we will use the methods discussed in [Courtney, 1997, Olson and Brill, 1997] to track objects and recognize the simple events. In order to recognize a complex event, the system must keep a record of the sub-events that have occurred thus far, and the objects involved in them. Whenever the first sub-event in a complex event's sequence is recognized, an *activation* for that complex event is created. The activation contains the *ID* of the object involved in the event, and an *index*, which is the number of sub-events in the sequence that have been recognized thus far. The index is initialized to 1 when the activation is created, since the activation is only created when the first sub-event matches. The system maintains a list of current activations for each defined complex-event type. Whenever any new event is recognized, the list of current activations is consulted to see if the newly recognized (or *incoming*) event matches the next sub-event in the complex event. If so, the index is incremented. If the index reaches the total number of sub-events in the sequence, the complete complex event has been recognized, and any desired alarm can be generated. Also, since the complex event that was just recognized may also be a sub-event of another complex event, the activation lists are consulted again (recursively) to see if the indices of any other complex event activations can be advanced.

To return to our THEFT example, the complex THEFT event has two sub-events, REMOVE and EXIT. When a REMOVE event occurs, an activation for the THEFT event is created, containing the ID of the person involved in the REMOVE event, and an index set to 1. Later, when another event is recognized by the system, the activation is consulted to see if the event type of this new, incoming event matches the next sub-event in the sequence (in this case, EXIT). If the event type matches, the object ID is also checked, in this case to see if the person EXITing is the same as that of the person who REMOVEd the object earlier. This is to ensure that we do not signal a THEFT event when one person picks up an object and a different person exits the area. In a closed environment, the IDs used may merely be track-IDs, in which each object that enters the monitored area is assigned a unique track-ID, and the track-ID is discarded when the object is no longer being tracked. If both the event type and the object ID match, the activation's index is incremented to 2. Since there are only 2 sub-events in the complex event in this example, the entire complex-event has been recognized, and an alarm is generated if desired. Also, since the THEFT event has been recognized, this newly recognized THEFT event may be a sub-event of another complex event. When the complex THEFT event is recognized, the current activations are recursively checked to see if the theft is a part of another higher-level event, such as a CRIME-SPREE.

### 2.4.2. Variations and enhancements

We have described the basic mechanism of defining and recognizing complex events. There are several variations on this basic mechanism. One is to allow unordered events, i.e., complex events which are simply the conjunction or disjunction of their sub-events. Another is to allow negated sub-events, which can be used to cancel an activation when the negated sub-event occurs. For example, considering the definition for THEFT again, if the person pays for the item, it is not a theft. Also, if the person puts the item back down before leaving, no theft has occurred. A more complete definition of theft is one in which "a person picks up an item and then leaves without putting it back or paying." Assuming we can recognize the simple events REMOVE, DEPOSIT, PAY, and EXIT, the complex THEFT event can now be expressed as the ordered list (REMOVE, ~DEPOSIT, ~PAY, EXIT), where "~" indicates negation. Another application of the complex event with negated sub-events is to detect suspicious behavior in front of a building. The normal behavior may be for a person to park the car, get out of it, and then come up into the building. If the person parks the vehicle and leaves the area without coming up into the building, this may be a car bombing scenario. If we can detect the sub-events for PARK, OUTCAR, ENTER-BUILDING, and EXIT, we can define the car-bombing scenario as (PARK, OUTCAR, ~ENTER-BUILDING, EXIT).

Another variation is to allow the user to label the objects involved in the events, which facilitates the ability to specify that two object be different. Consider a different car bombing scenario in which two cars pull up in front of the building, and a person gets out of one car and into the other, which drives away. The event definition must specify that there are two *different* cars involved: the car-bomb and the getaway-car. This can be accomplished by labelling the object involved when defining the event, and giving different labels to objects which must be different.

Finally, one could allow multiple activations for the same event. For example, the desired behavior may be that a separate THEFT event should be signalled for each item stolen by a given person, e.g., if a person goes into a store and steals three things, three THEFT events are recognized. The basic mechanism described above signals a single THEFT event no matter how many objects are stolen. We can achieve the alternate behavior by creating multiple activations for a given event type, differing only in the ID's of the objects involved.

### 2.4.3. Implementation in AVS

We have described a method for defining and recognizing complex events. Most of this has been implemented and incorporated into the AVS system. This subsection describes the current implementation.

AVS analyzes the incoming video stream to detect and recognize events such as ENTER, EXIT, DEPOSIT, and REMOVE. The primary technique used by AVS for event recognition is motion graph matching as described in [Courtney, 1997]. The AVS system recognizes and reports these events in real time as illustrated in Figure 10. When the person enters the monitored area, an ENTER event is recognized as shown in the image on the left. When the person picks up an object, a REMOVE event is recognized, as depicted in the center image below. When the person exits the area, the EXIT

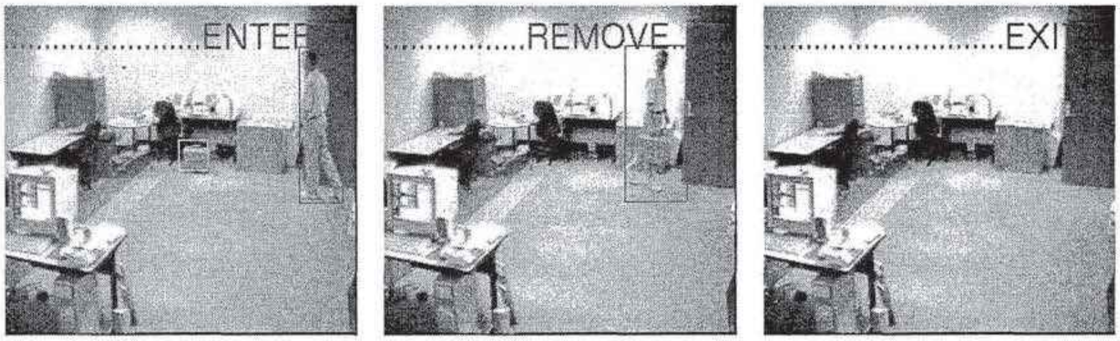


Figure 10: A series of simple events

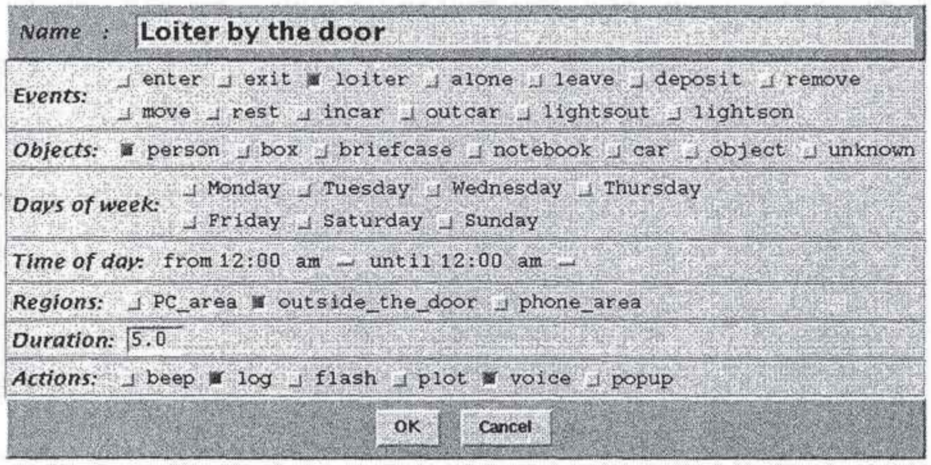
Ex. D Page 12 of 20

Canon Corrected Ex. 1007 Page 122 of 219 event is signalled as shown in the image on the right

While the AVS system recognizes numerous events as shown above, the user can select which events are of interest by providing the dialog box interface illustrated in Figure 11. The user selects the event type, object type, time, location, and duration of the event of interest using a mouse. The user can also select an action for the AVS system to take when the event is recognized. This dialog box defines one type of simple event; an arbitrary number of different simple event types can be defined via multiple uses of the dialog box. The illustration in Figure 11 shows a dialog box defining an event called "Loiter by the door" which is triggered when a person loiters in the area near the door for more than 5 seconds.

AVS will generate a voice alarm and write a log entry when the specified event occurs. If the event is only being defined in order to be used as a sub-event in a complex event, the user might not check any action box, and no action will be taken when the event is recognized except to see if it matches the next sub-event in a complex-event activation, or generate a new activation if it matches the first sub-event in a complex event.

After one or more simple events have been defined, the user can define a complex event via the dialog box shown in Figure 12. This dialog box presents two lists: on the left is a scrolling list of all the event types that have been defined thus far, and on the right is a list of the sub-events of the complex event being defined. The sub-event list is initially blank when defining a new complex event. When the user double-clicks with the left mouse button on an item in the event list on the left, it is added as the next item in the sub-event list on the right. When the user double-clicks with the right mouse button on an item in the event list on the left, that item is also added to the sub-event list on the right, but as a negated sub-event. The event name is prefixed with a tilde (~) to indicate that the event is negated.



Name : Loiter by the door

Events: enter exit [✓ loiter] alone leave deposit remove move rest incar outcar lightsout lightson

Objects: [✓ person] box briefcase notebook car object unknown

Days of week: Monday Tuesday Wednesday Thursday Friday Saturday Sunday

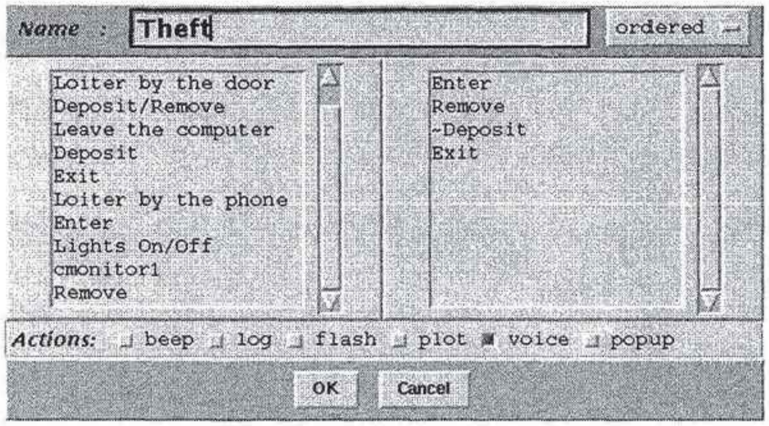
Time of day: from 12:00 am until 12:00 am

Regions: PC_area [✓ outside_the_door] phone_area

Duration: 5.0

Actions: beep [✓ log] flash plot [✓ voice] popup

Figure 11: Selecting a type of simple event



Name : Theft

ordered

| | |
|---|---|
| Loiter by the door | Enter |
| Deposit/Remove | Remove |
| Leave the computer | ~Deposit |
| Deposit | Exit |
| Exit | |
| Loiter by the phone | |
| Enter | |
| Lights On/Off | |
| cmonitor1 | |
| Remove | |

Actions: beep log flash plot [✓ voice] popup

Figure 12: Defining a complex event

In the upper right corner of the complex-event definition dialog box is an option menu via which the user indicates how the sub-events are to be combined. The default selection is "ordered" to indicate sequential processing of the sub-events. The other options are "all" and "any." If "all" is selected, the complex event will be signalled if all of the sub-events are matched, regardless of order, i.e., the complex event is simply the conjunction of the sub-events. If "any" is selected, the complex event occurs if any of the sub-events occurs, i.e., the complex event is the disjunction of the sub-events. At the bottom of the dialog box, the user can select the action to take when the complex event is recognized. The user can save the entire set of event definitions to a file so that they may be read back in at a later time.

Once a simple or complex event has been defined, the AVS system immediately begins recognition of the new events in real time, and taking the actions specified by the user. The AVS system, augmented as described, provides a functioning realization of the complex-event recognition method.

## 3. Advanced segmentation and tracking

In security applications, it is often necessary to track the movements of one or more people and objects in a scene monitored by a video camera. In real scenes, the objects move in unpredictable ways, may move close to one another, and may occlude each other. When a person moves, the shape of his or her image changes. These factors make it difficult to track the locations of individual objects throughout a scene containing multiple objects. The tracking capabilities of the original AVS system fail when there is mutual occlusion between the tracked objects. This section describes a new tracking method which overcomes this limitations of the previous tracking method, and maintains the integrity of the tracks of people even when they partially occlude one another.

The segmentation algorithm described here is related to tracking systems such as [Wren et al., 1997, Grimson et al., 1998, Cai et al., 1995] in that it extends the reference image to include a statistical model of the background. Our method further extends the tracking algorithm to reason explicitly about occlusion and maintain object tracks during mutual occlusion events. Unlike the capabilities described in previous sections, the new tracking method does not run in real time, and has not yet been integrated into the AVS system. Optimizations of the new method are expected to enable it to achieve real time operation in the future.

Figure 13 depicts an example scene containing two people. In (a), the two people are standing apart from each other, with Person-1 on the left, and Person-2 on the right. In (b), Person-1 moves to the right so that he is partially occluded by Person-2. Using a conventional technique such as background subtraction, it is difficult to maintain the separate tracks of the two people in the scene, since the images of the two people merge into a single large region.

Figure 14 shows a sequence of frames (in normal English reading order) in which it is particularly difficult to properly maintain the tracks of the two people in the scene. In this sequence, Person-2 moves from right to left and back again, crossing in front of Person-1. There are significant occlusions (e.g., in the third frame shown), and the orientations of both people with respect to the camera change significantly throughout the sequence,

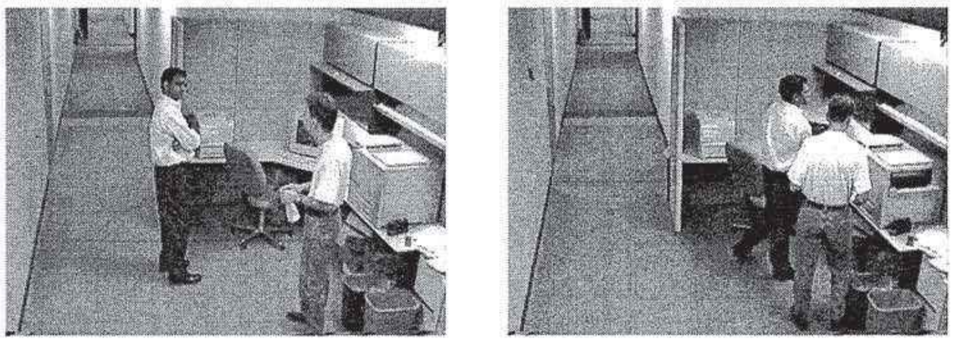

(a) (b)

Figure 13: An example scene containing two people with occlusion

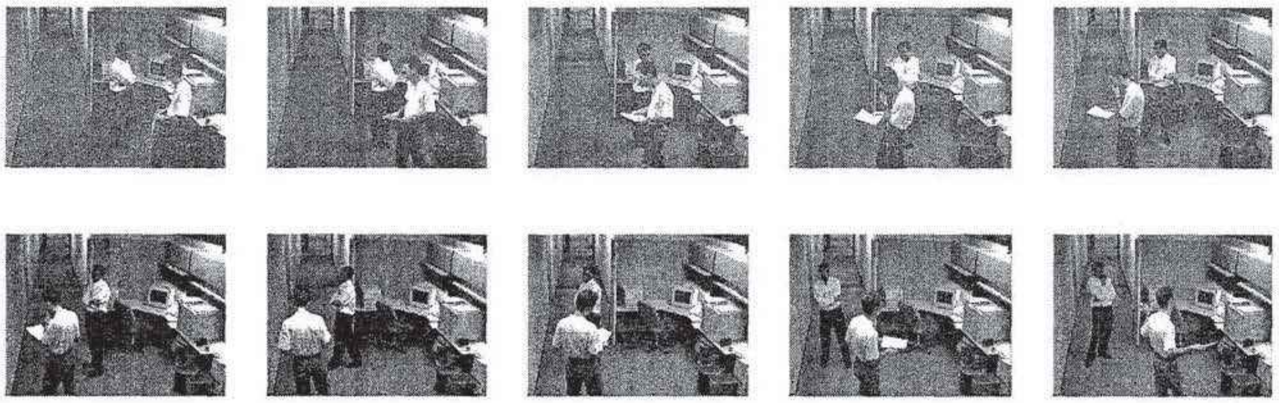


Figure 14: A difficult tracking sequence

making conventional template matching fail on this sequence.

A new tracking method is used to maintain tracks in sequences such as those depicted in Figures 13 and 14. The method maintains an estimate of the size and location of the objects being tracked, and creates an image which approximates the probability that the object intersects that pixel location. Figure 15b shows the probability images for the two person scene of Figure 13a, which is repeated here as 15a. The ellipse on the left indicates the estimated location of Person-1, and the ellipse on the right indicates the estimated location of Person-2. The brightness indicates the probability that the person's image intersects the given pixel, which is highest in the middle of the region, and falls off towards the edge. The black outlines represent the 50% probability contours. The size and shape of the regions are roughly the size and shape of a person standing at that location in the image.

We refer to the "person shaped" probability regions as *probabilistic templates* or simply *p-templates*. The path of the p-template through the scene represents the "track" of a given person which is maintained by the tracking system. P-templates can be used to reason about occlusion in a video sequence. While we only address the issue of p-templates for tracking people that are walking upright, the concept is applicable to tracking any object, e.g., vehicles and crawling people; although the shape of the p-template would need to be adapted to the type of object being tracked.

When the people in the scene overlap, the separate locations of the people can be maintained using the p-templates, and the region of partial occlusion can be detected. Figure 16 shows examples of such a situation. The two ellipses are maintained, even though the people are overlapping. The tracks of the people can be maintained through occlusions by tracking primarily on the basis of non-overlapping areas. This works for both the slight occlusion in Figures 16 (a) and (b), and often even for the very strong occlusions such as in Figures 16 (c) and (d). During the occlusions shown in Figure 14 and again in Figure 16 (c) and (d), the head of Person-1 is tracked, and the lower-body of Person-2 is tracked.

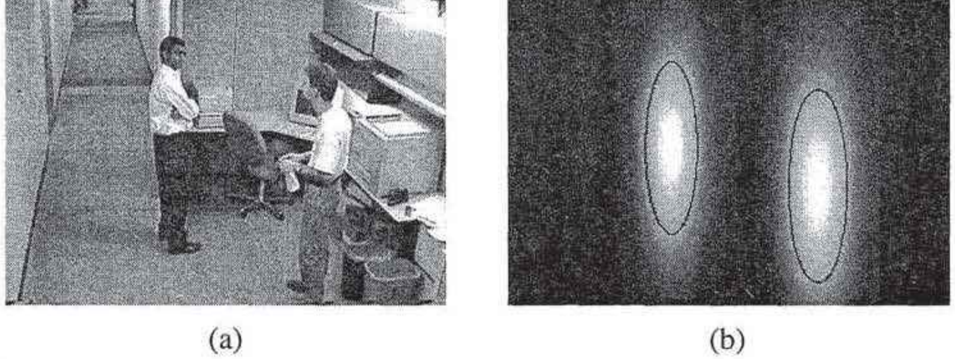

(a) (b)

Figure 15: Probability image for the locations of the people in the scene

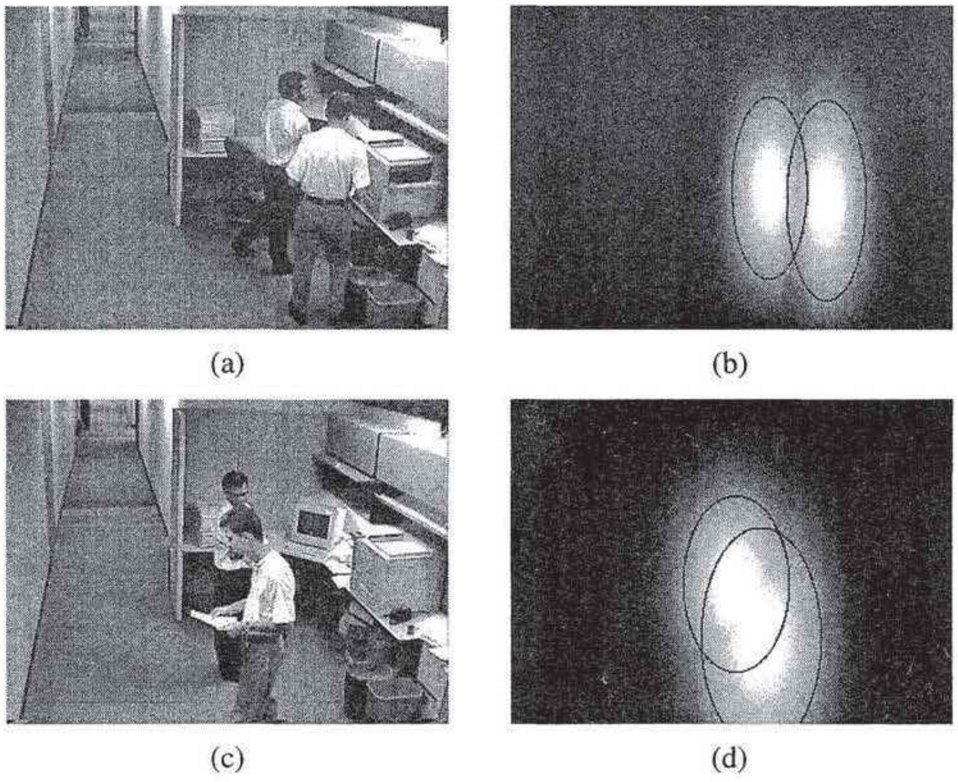


Figure 16: P-template images for partially occluding people

The new method requires a means of instantiating a new p-template when a person enters the scene, and updating the location of the region as the person moves through the scene. First we will describe the update mechanism, assuming that the p-templates have already been instantiated. The instantiation mechanism is described later.

The p-templates described above and depicted in Figures 15 and 16 represent the *prior* probabilities of the person locations, based on looking at the *previous* frame. These priors are then used to compute an estimate of the *posterior* probabilities of the person locations by looking at the new or *current* frame. The computation of the posterior probabilities takes into account both the prior probabilities and the information in the new frame. The posterior probabilities are used to update the locations of the people, and the new locations of the people are then used to compute the priors for the *next* frame.

Our current implementation computes the posteriors using a form of background differencing. Figure 17 shows the posteriors for the p-templates shown in Figure 16. Note that although there is significant overlap in the posterior estimates, especially in Figures 17 (e) and (f), there are significant differences in the brightnesses of the non-occluding areas. In Figure 17 (e), which represents the posteriors for Person-1, the head area of Person-1 is significantly brighter than in Figure 17 (f). Similarly, Figure 17 (f), which represents the posteriors for Person-2, is significantly brighter in the unoccluding area of Person-2's lower body.

Once the posteriors are computed, they are used to estimate the location of the tracked objects. In our implementation of a person tracker, we specifically need to estimate the location of the person's feet in the image, and their height in the image in pixels. Once the location and height are estimated, we can use the image-to-world coordinate transformation technique used in the original AVS system and described in [Olson and Brill, 1997]. That technique, called *quad-mapping*, computes the map locations of objects given the image locations of the bottom of the objects, e.g., in the case of a person, the location of the feet. Furthermore, if the scale of the map is known, the quad-mapping technique will estimate the size of the object, i.e., the height of a person being tracked.

If the lower portion of the p-template is unoccluded, foot locations are estimated directly from the image by looking at the bottom portion of the brightened region. If the upper portion is also unoccluded, the height can similarly obtained directly

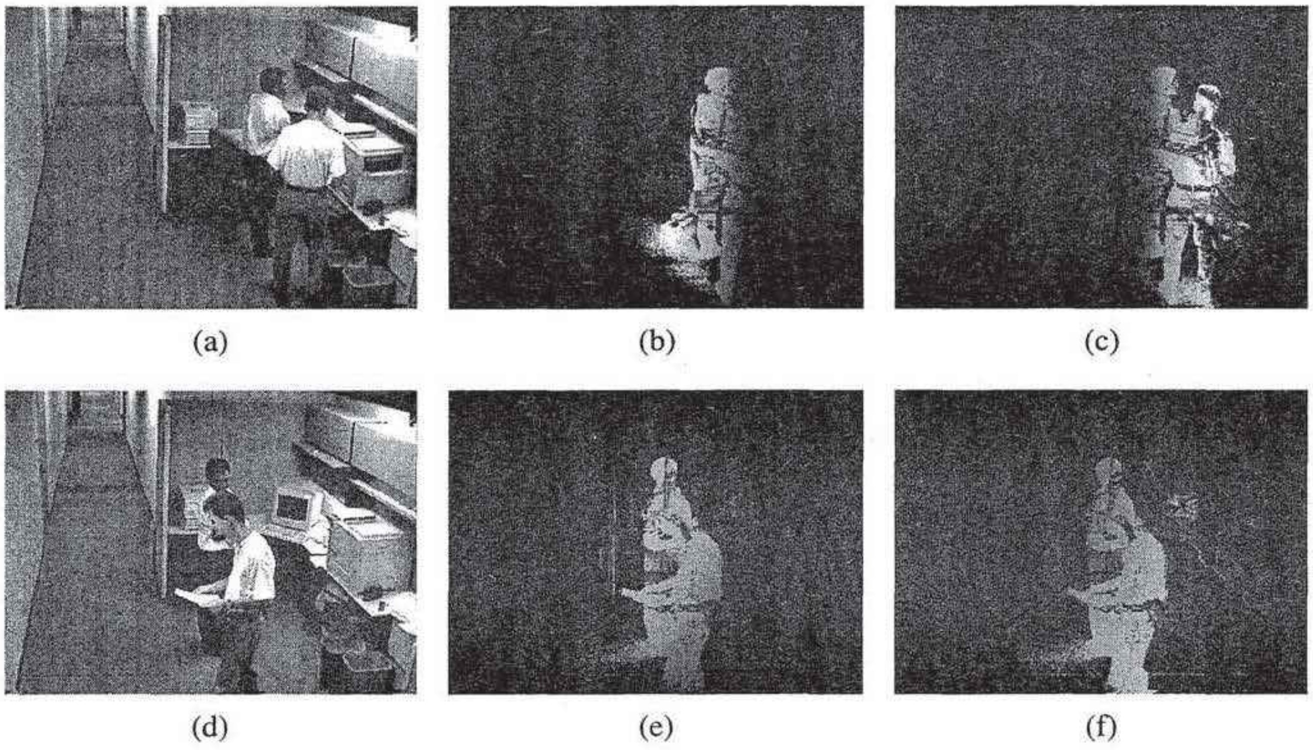


Figure 17: Posterior probability images for partially occluding people

from the image. If the upper part is occluded, but the lower part is not, the foot location is still determined directly from the image, but height is estimated using an estimate of the three-dimensional height of the person. The image height is then obtained by projecting the 3D height back into the image using the quad-mapping technique. If the lower portion is occluded, but the upper part is not, then the upper location is determined directly from the image, and then the 3D height is back-projected into the image to determine the foot location. If both the top and bottom are occluded, the location and height estimates are left unchanged from the previous frame.

Once the foot location and height of the person are computed, it is straightforward to compute the new location of the p-template, which is the Gaussian oval whose location and dimensions are determined by the foot location and image height computed above. The new p-template is then used to find the location of the person in the next frame, and the process repeats while the person remains in the scene.

A new p-template is instantiated whenever a new person enters the scene. Instantiation is best described in a Bayesian probabilistic framework. The p-templates constitute models of the objects in the environment. All of the pixels in the image are the result of a projection of some object in the environment—either from the background, or one of the people in the scene, or something else. The sum of the probabilities that the pixel is either from the background, from a person, or from "something else" must be 1.0. We maintain an "unknown" model to account for the probability that pixels may arise as a result of "something else." We compute the probability that each of the models caused the observed pixel value (where the unknown model is equally likely to produce any pixel value), and then use Bayes' formula to compute the inverse, i.e., the probability that the observed pixel value came from each of the models. When this computation is performed, for some of the pixels, the probability that the pixel came from the unknown model is the highest of all of the model probabilities. This results in a probability image for the unknown model, which represents pixels which probably came from something other than the objects the system knows about. At each frame, the probability image for the unknown model is computed, and this image is examined to see if adding a new person model would account for these unknown pixels. If so, a new person p-template is instantiated at the appropriate location, and the posteriors are recomputed.

Use of the procedure described above to track multiple people maintains tracks through occlusions where our previous technique could not. The robustness to occlusion of the new method enables video monitoring applications to improve tracking reliability in natural environments.

## 4. Best-view selection performance

Olson and Brill [1997] previously described the "best view selection" application of AVS technology. In this application, the system monitors and records the movements of humans in its field of view. For every person that it sees, it creates a log file that summarizes important information about the person, including a snapshot taken when the person was close to the camera and (if possible) facing it.

As the person is tracked through the scene, the tracker examines each image it captures of that person. If the new image is a better view of the person than the previously saved snapshot, the snapshot is replaced with the new view. In this manner, the system always contains the "best" view seen of the person thus far. When the person leaves the scene, the log entry is saved to a file. Each log entry records the time when the person entered the scene and a list of coordinate pairs showing their position in each video frame. The log entry also contains the "best" snapshot of the person while they were in the scene. Finally, the log entry file contains a pointer to the reference image that was in effect when the snapshot was taken. This information forms an extremely concise description of the person's movements and appearance while they were in the scene. An example of such a record in shown in Figure 18.

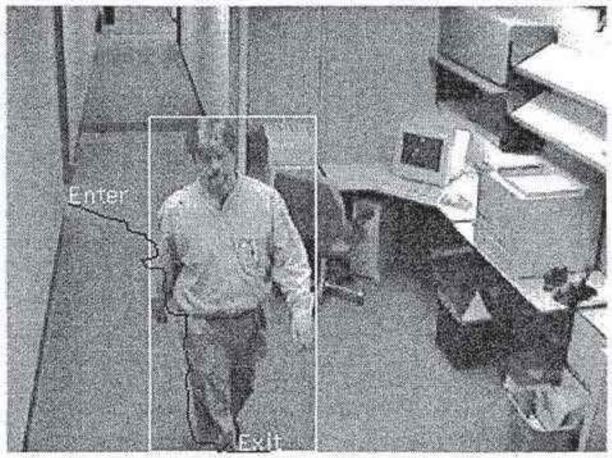


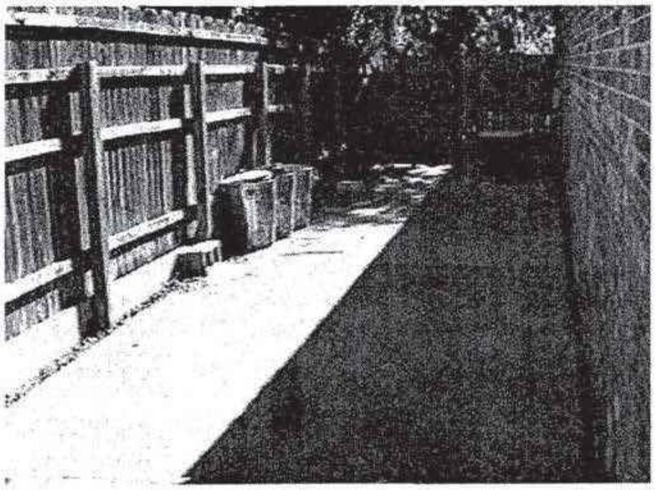
Figure 18: Example best view selection record

In an initial evaluation of this system, the system was installed in an uncontrolled office hallway and run for 118 hours. In this time, the system recorded 965 log entries in 35MB (uncompressed). The resulting records were examined to estimate the system performance, and we estimated 96% detection rate at 6% false alarm rate, with most errors due to segmentation and correspondence failure. However, for this initial experiment, there was no ground truth against which the performance could be measured.

Recently, we have evaluated the system against ground truth observations. The performance of the system was initially evaluated on four hours of indoor video data. The video was manually annotated to obtain ground truth, and the surveillance system was evaluated against this ground truth. For situations in which only one person was in the scene, the system recorded exactly one record for each person, i.e., no person passed undetected though the field of view, and there is exactly one record for each such person. In the indoor condition, we observed a 100% detection rate.

For situations involving more than one person, the system occasionally failed to maintain track through partial occlusions. The result of this is that the system took extra pictures of these people when their track was re-acquired after the occlusion. On other occasions, the system failed to recognize that a motion region contained two people, and so it only took one picture that contained both people. We expect to reduce these errors via the use of the new tracking algorithms described above, once these algorithms are running in real time and are incorporated into the AVS system.

In order to evaluate and improve the system performance in outdoor monitoring environments, we have adopted an iterative research methodology in which we record representative videotape (2-3 hours), ground truth it with respect to the 'person events' that occur in the scene. One 'person event' is defined to be a video sequence in which one person enters monitored area completely, walking upright, and then exits field of view completely. We then run AVS system on the videotape and measure the false positives and negatives on person events. We then improve system as necessary to eliminate errors on video sequence and repeat the process.


(a)

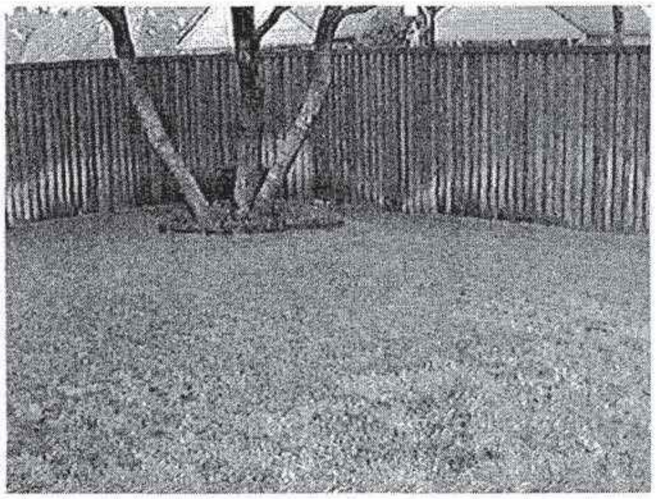

(b)

Figure 19: Outdoor environments

Outdoor environments can be particularly difficult for video monitoring systems that operate based on change detection, due to the outdoor lighting variation. Figure 19 depicts two outdoor environments used to evaluate AVS best-view-selection performance. In Figure 19 (a), there is a strong shadow line running down the center of the field of view, which moves as the sun angle changes. The shadow motion here is sufficient to cause problems for a fixed background subtraction system within 5 minutes. There are also a number of trees in the background which move when the wind blows. Moreover, the shadows of these trees fall directly into the rear of the monitored area, and these shadows move with the wind as well. The shadow of the tree in Figure 19 (b) has a similar behavior. Cloud movement also causes large changes in brightness throughout the images.

Our initial outdoor evaluation was conducted in the environment depicted in Figure 19 (a). We captured two hours of outdoor video with extremely difficult imaging conditions caused by wind blown vegetation and strong shadows, which produced a large amount of "noise" motion. Additionally, the gate at the rear of the scene often blew open and closed. We manually ground-truthed the video to determine that a person entered the scene 20 times during the two hour sequence. The system recorded 16 of these events, for a detection rate of 75%. The undetected people were "lost in the noise." The system also produced 16 false detections in the two hour period, caused by noise from the moving shadows.

We were able to improve on this performance using our iterative research methodology to achieve a 100% detection rate for the 20 events in this two hour sequence. The system still recorded 8 false positives on this sequence. Four of these were caused by the gate blowing open and closed. The other four were cases in which the system lost track of the person in the field of view, and therefore took two pictures of the person, one before losing track, and another after picking up the track again. These cases are therefore more properly referred to as "extra pictures" rather than false positives.

Having achieved improved performance in the environment depicted in Figure 19 (a), we proceeded to test the system in the environment of Figure 19 (b). One three separate days we captured 1-2 hours of video, for a total of 4 hours of test video data in the environment of Figure 19 (b). We ground-truthed this video to determine that it contained 115 person events. The AVS system processed this video using the best-view-selection algorithm, and the results were compared to ground truth. We observed a 100% detection rate and a 2.6% false positive rate as a result of three false positives, all of which were "extra pictures."

In general, system performance was excellent in the indoor condition, with the exception of scenes containing multiple people, which produced extra records. We expect to address the multi-person problem using the p-template technique described in section 3. No person entered the scene without being recorded, even when there were multiple people. The system performance degrades in difficult outdoor lighting conditions, but it has improved significantly in recent work.

## 5. Conclusion

We have described several improvements in the video monitoring capabilities of the AVS system. Some improvements, such as vehicle event recognition, increase the functionality of the system to enable it to recognize new classes of events. Other improvements, such as the advanced segmentation and tracking, increase the robustness of the system's ability to recognize events in the presence of complications such as occlusion. We will continue to make improvements in the two categories of increased functionality and increased robustness. For the functionality improvements, we expect to recognize new classes of events, especially events regarding vehicles. For the robustness improvements, we are pursuing techniques that enable the system to be robust to lighting variation. As the techniques become more complex, additional effort will be needed to optimize the algorithms for real time operation. Our advanced segmentation and tracking will be the subject of optimization efforts in the near future.

## References

[Cai, et al., 1995] Q. Cai, A. Mitiche, and J. K. Aggarwal. Tracking human motion in an indoor environment. In *Proc. of International Conference on Image Processing* (ICIP-95), 1995.

[Davis and Bobick, 1997] J. Davis and A. Bobick. Representation and recognition of human movement using temporal templates. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR-97)*, pp. 928-934, 1997.

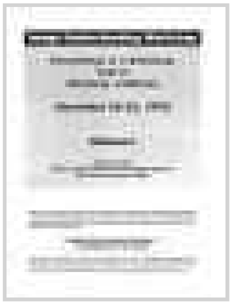
[Courtney, 1997] J. D. Courtney. Automatic video indexing via object motion analysis. *Pattern Recognition*, **30**(4), April 1997

[Grimson et al., 1998] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR-98)*, 1998.

[Olson and Brill, 1997] T. J. Olson, and F. Z. Brill. Moving object detection and event recognition algorithms for smart cameras. *Proceedings of the 1997 Image Understanding Workshop*, New Orleans, LA, May 11-14, 1997, p. 159-175.

[Tserng, 1998] C. H. Tserng. Event recognition in scenes involving people and cars. Master's Thesis, MIT, May 1998.

[Wren et al., 1997] A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 1997, **19**(7), pp. 780-785.



## Related Names

- DARPA Image Understanding Workshop (26th : 1998 : Monterey, CA)
- Lukes, George E
- United States Defense Advanced Research Projects Agency Information Systems Office

## Related Subjects

- Image Processing — Congresses
- Optical Pattern Recognition — Congresses
- Remote Sensing — Congresses
- Computer Vision — Congresses

## Item Details (Item 1 of 1)

← Return to search results | Start over  Add Star  **Item Actions**

# Image Understanding Workshop: Proceedings of a Workshop Held in Monterey, California, November 20-23, 1998

sponsored by Defense Advanced Research Projects Agency, Information Systems Office ; [edited by George E. Lukes]

| | |
|---|---|
| **Format** | Book |
| **Published** | San Francisco, Calif. : Distributed by Morgan Kaufmann Publishers, [1998] |
| **Language** | English |
| **Variant Title** | Proceedings, 1998 Image Understanding Workshop, 20-23 November, Hyatt Regency, Monterey, CA |

Ex. E Page 1 of 3

| | |
|---|---|
| **ISBN** | 1558605835 |
| **Description** | 2 v. : ill. ; 28 cm. |
| **Notes** | Includes bibliographical references and index. |
| **Technical Details** | Access in Virgo Classic<br>Staff View |

```
LEADER 01549nam a2200361Ia 4500
001 u2928081
003 SIRSI
005 19990805102441.0
008 981215s1998 cau 100 0 eng d
020         a| 1558605835
035         a| (Sirsi) i1558605835
035         a| (OCoLC)40497803
040         a| XRL c| XRL d| OCL
090         a| TA1632 b| .I57 1998
111  2      a| DARPA Image Understanding Workshop n| (26th : d| 1998 : c| Monterey, CA)
245  1   0  a| Image Understanding Workshop : b| proceedings of a workshop held in
            Monterey, California, November 20-23, 1998 / c| sponsored by Defense Advanced
            Research Projects Agency, Information Systems Office ; [edited by George E.
            Lukes].
246  1   4  a| Proceedings, 1998 Image Understanding Workshop, 20-23 November, Hyatt
            Regency, Monterey, CA
260         a| San Francisco, Calif. : b| Distributed by Morgan Kaufmann Publishers, c| [1998]
300         a| 2 v. : b| ill. ; c| 28 cm.
504         a| Includes bibliographical references and index.
596         a| 5
650      0  a| Image processing v| Congresses.
650      0  a| Optical pattern recognition v| Congresses.
650      0  a| Remote sensing v| Congresses.
650      0  a| Computer vision v| Congresses.
700  1      a| Lukes, George E.
994         a| Z0 b| VA@
710  1      a| United States. b| Defense Advanced Research Projects Agency. b| Information
            Systems Office.
852         b| SCI-ENG c| STACKS
866      0  8| 1 a| v.1-2
999         a| TA1632 .I57 1998 v.1 w| MONO-SER i| X004283491 l| STACKS m| SCI-
            ENG t|BOOK
999         a| TA1632 .I57 1998 v.2 w| MONO-SER i| X004283490 l| STACKS m| SCI-
            ENG t|BOOK
```

**See fewer details**

## Availability

Request LEO delivery

**Brown SEL**

STACKS

v.1-2

| Library | Location | Map | Availability | Call Number |
|---|---|---|---|---|
| Brown Science and Engineering | Stacks | N/A | AVAILABLE | TA1632 .I57 1998 v.1 |

| Library | Location | Map | Availability | Call Number |
|---|---|---|---|---|
| Brown Science and Engineering | Stacks | N/A | AVAILABLE | TA1632 .I57 1998 v.2 |

Ex. E Page 3 of 3

NCSU LIBRARIES

Library Catalog

Search for words: Search

Search within results Start Over[1]

**Image Understanding Workshop : proceedings of a workshop held in Monterey, California, November 20-23, 1998**
sponsored by Defense Advanced Research Projects Agency, Information Systems Office ; [edited by George E. Lukes].

| | |
|---|---|
| Author: | DARPA Image Understanding Workshop (26th : 1998 : Monterey, CA)[11] |
| Published: | San Francisco, Calif. : Distributed by Morgan Kaufmann Publishers, [1998] |
| Description: | 2 v. : ill. ; 28 cm. |
| Format: | Book |
| Summary | Reports and technical results presented at the biennial workshops of the DARPA Image Understanding research program. Content provided by Syndetic Solutions, Inc. Terms of Use (more) [5] |

Browse Shelf [12]

## Browse Related Subjects

- Image processing[13] -- Congresses[14]
- Optical pattern recognition[15] -- Congresses[16]
- Remote sensing[17] -- Congresses[18]
- Computer vision[19] -- Congresses[20]

MARC

| | | | |
|---|---|---|---|
| **000** | | | 01338cam a2200313Ia 4500 |
| **003** | | | OCoLC |
| **005** | | | 19990810132534.0 |
| **008** | | | 981215s1998 caua b 100 0 eng d |
| **020** | | | \|**a**1558605835 |
| **035** | | | \|**a**(OCoLC)40497803 |
| **040** | | | \|**a**XRL \|**c**XRL \|**d**OCL |
| **049** | | | \|**a**NRCC \|**x**ERJ |
| **090** | | | \|**a**TA1632 \|**b**.D35 1998 |
| **111** | 2 | | \|**a**DARPA Image Understanding Workshop \|**n**(26th : \|**d**1998 : \|**c**Monterey, CA) |
| **245** | 1 | 0 | \|**a**Image Understanding Workshop : \|**b**proceedings of a workshop held in Monterey, California, November 20-23, 1998 / \|**c**sponsored by Defense Advanced Research Projects Agency, Information Systems Office ; [edited by George E. Lukes]. |
| **246** | 1 | 4 | \|**a**Proceedings, 1998 Image Understanding Workshop, 20-23 November, Hyatt Regency, Monterey, CA |
| **260** | | | \|**a**San Francisco, Calif. : \|**b**Distributed by Morgan Kaufmann Publishers, \|**c**[1998] |
| **300** | | | \|**a**2 v. : \|**b**ill. ; \|**c**28 cm. |
| **504** | | | \|**a**Includes bibliographical references and index. |
| **596** | | | \|**a**14 |
| **650** | | 0 | \|**a**Image processing \|**v**Congresses. |
| **650** | | 0 | \|**a**Optical pattern recognition \|**v**Congresses. |
| **650** | | 0 | \|**a**Remote sensing \|**v**Congresses. |
| **650** | | 0 | \|**a**Computer vision \|**v**Congresses. |
| **700** | 1 | | \|**a**Lukes, George E. |
| **710** | 1 | | \|**a**United States. \|**b**Defense Advanced Research Projects Agency. \|**b**Information Systems Office. |
| **948** | | | \|**a**08/09/1999 \|**b**10/06/1999 |
| **919** | | | \|**a**AKD-4329 |
| **918** | | | \|**a**1089089 |

**Address and Phone Number:** 2 Broughton Drive, Raleigh, NC 27695-7111 (919) 515-3364 |

Ex. F Page 1 of 1

October 1995
Volume 13 Number 4

acm Series on
Computing
Methodologies

# Transactions
# nformation
# stems

## Special Issue on Video Information Retrieval

acm PRESS

Published by ACM, the First Society in Computing

# acm Transactions on Information Systems

ACM
1515 Broadway
New York, NY 10036
Tel: (212) 869-7440

ACM European Service Center
Avenue Marcel Thiry 204
1200 Brussels, Belgium
Tel: 32 2 774 9602
Fax: 32 2 774 9690
Email: acm_europe@acm.org

*ACM Transactions on Information Systems* (ISSN 1046-8188) is published 4 times a year in January, April, July, and October by the Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036. Second-class postage paid at New York, NY 10001, and at additional mailing offices. Postmaster: Send address changes to Transactions on Information Systems, ACM, 1515 Broadway, New York, NY 10036.

For subscription and submissions information, see inside back cover.

# Motion Recovery for Video Content Classification

NEVENKA DIMITROVA and FOROUZAN GOLSHANI
Arizona State University, Tempe

Like other types of digital information, video sequences must be classified based on the semantics of their contents. A more-precise and completer extraction of semantic information will result in a more-effective classification. The most-discernible difference between still images and moving pictures stems from movements and variations Thus, to go from the realm of still-image repositories to video databases, we must be able to deal with motion. Particularly, we need the ability to classify objects appearing in a video sequence based on their characteristics and features such as shape or color, as well as their movements By describing the movements that we derive from the process of motion analysis, we introduce a dual hierarchy consisting of spatial and temporal parts for video sequence representation. This gives us the flexibility to examine arbitrary sequences of frames at various levels of abstraction and to retrieve the associated temporal information (say, object trajectories) in addition to the spatial representation. Our algorithm for motion detection uses the motion compensation component of the MPEG video-encoding scheme and then computes trajectories for objects of interest. The specification of a language for retrieval of video based on the spatial as well as motion characteristics is presented.

Categories and Subject Descriptors. H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.5 1 [**Information Interfaces and Presentation**]· Multimedia Information Systems; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*motion*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Content-based retrieval of video, motion recovery, MPEG compressed video analysis, video databases, video retrieval

## 1. INTRODUCTION

Applications such as video on demand, automated surveillance systems, video databases, industrial monitoring, video editing, road traffic monitoring, etc. involve storage and processing of video data. Many of these applications can benefit from retrieval of the video data based on their content. The problem is that, generally, any content retrieval model must have the capability of

This article is a revised version with major extensions of an earlier paper which was presented at the ACM Multimedia '94 Conference.

Authors' addresses: N Dimitrova, Philips Laboratories, 345 Scarborough Road, Briarcliff Manor, NY 10562; email: nvd@philabs.philips com, F. Golshani, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406; email golshani@asu.edu.

dealing with massive amounts of data. As such, classification is an essential step for ensuring the effectiveness of these systems.

Motion is an essential feature of video sequences. By analyzing motion of objects we can extract information that is unique to the video sequences. In human and computer vision research there are theories about extracting motion information independently of recognizing objects. This gives us support for the idea of classifying sequences based on the motion information extracted from video sequences regardless of the level of recognition of the objects. For example, using the motion information we can not only submit queries like "retrieve all the video sequences in which there is a moving pedestrian and a car" but also queries that involve the exact position and trajectories of the car and the pedestrian.

Previous work in dynamic computer vision can be classified into two major categories based on the type of information recovered from an image sequence: recognition through recovering structure from motion and recognition through motion directly. The first approach may be characterized as attempting to recover either low-level structures or high-level structures. The low-level structure category is primarily concerned with recovering the structure of rigid objects, whereas the high-level structure category is concerned primarily with recovering nonrigid objects from motion. Recovering objects from motion is divided into two subcategories: low-level motion recognition and high-level motion recognition. Low-level motion recognition is concerned with making the changes between consecutive video frames explicit (this is called optical flow [Horn and Schunck 1981]). High-level motion recognition is concerned with recovering coordinated sequences of events from the lower-level motion descriptions.

Compression is an inevitable process when dealing with large multimedia objects. Digital video is compressed by exploiting the inherent redundancies that are common in motion pictures. Compared to encoding of still images, video compression can result in huge reductions in size. In the compression of still images, we take advantage of spatial redundancies caused by the similarity of adjacent pixels. To reduce this type of redundancy, some form of transform-based coding (e.g., Discrete Cosine Transform, known as DCT) is used. The objective is to transform the signal from one domain (in this case, spatial) to the frequency domain. DCT operates on $8 \times 8$ blocks of pixels and produces another block of $8 \times 8$ in the frequency domain whose coefficients are subsequently quantized and coded. The important point is that most of the coefficients are near zero and after quantization will be rounded off to zero. Run-length coding, which is an algorithm for recording the number of consecutive symbols with the same value, can efficiently compress such an object. The next step is coding. By using variable-length codes (an example is Huffman tables), smaller code words are assigned to objects occurring more frequently, thus further minimizing the size.

Our aim in the coding of video signals is to reduce the temporal redundancies. This is based on the fact that, within a sequence of related frames, except for the moving objects, the background remains unchanged. Thus to reduce temporal redundancy a process known as motion compensation is used. Motion compensation is based on both predictive and interpolative coding.

MPEG (Moving Pictures Expert Group) is the most general of the numerous techniques for video compression [Furht 1994; LeGall 1991; Mattison 1994]. In fact, the phrase "video in a rainbow" is used for MPEG, implying that by adjusting the parameters, one can get a close approximation of any other proposal for video encoding. Motion compensation in MPEG consists of predicting the position of each $16 \times 16$ block of pixels (called a macroblock) through a sequence of predicted and interpolated frames. Thus we work with three types of frames—namely, those that are fully coded independently of others (called reference frames or I-frames), those that are constructed by prediction (called predicted frames or P-frames), and those that are constructed by bidirectional interpolation (known as B-frames). It begins by selecting a frame pattern which dictates the frequency of I-frames and the intermixing of other frames. For example, the frame pattern IBBPBBI indicates (1) that every seventh frame is an I-frame, (2) that there is one predicted frame in the sequence, and (3) that there are two B-frames between each pair of reference and/or predicted frames. Figure 1 illustrates this pattern.

Our approach to extracting object motion is based on the idea that during video encoding by the MPEG method, a great deal of information is extracted from the motion vectors. Part of the low-level motion analysis is already performed by the video encoder. The encoder extracts the motion vectors for the encoding of the blocks in the predicted and bidirectional frames. A macroblock can be viewed as a coarse-grained representation of the optical flow. The difference is that the optical flow represents the displacement of individual pixels while the macroblock flow represents the displacement of macroblocks between two frames. At the next, intermediate level, we extract macroblock trajectories which are spatiotemporal representations of macroblock motion. These macroblock trajectories are further used for object motion recovery. At the highest level, we associate the event descriptions to object/motion representations.

Macroblock displacement in each individual frame is described by the motion vectors which form a coarse optical-flow field. We assume that our tracing algorithm is fixed on a moving set of macroblocks and that the correspondence problem is elevated to the level of macroblocks instead of individual points. The advantage of this elevation is that even if we lose individual points (due to turning, occlusion, etc.) we are still able to trace the object through the displacement of a macroblock. In other words, the correspondence problem is much easier to solve and less ambiguous. Occlusion and tracing of objects which are continuously changing are the subject of our current investigations.

In Section 2 of this article we survey some of the research projects related to our work. In Section 3 we present the object motion analysis starting from the low-level analysis through the high-level analysis. We discuss the importance of motion analysis and its relevance to our model which is presented in Section 3.4. Section 4 introduces the basic OMV structures (object, motion,

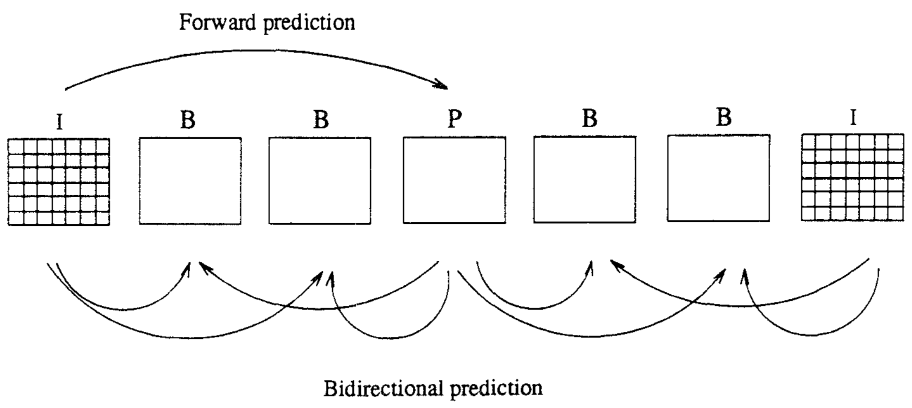


Fig. 1. Forward and bidirectional prediction in MPEG.

video-sequence), as the basis for the video information model. The basic retrieval operators, the OMV-language specification, and some examples are given. Empirical results are outlined in Section 5, and Section 6 presents some concluding remarks.

## 2. RELATED WORK

The research presented in this article builds on the existing results in two areas: dynamic computer vision and digital video modeling.

A current trend in computational vision is influenced by the idea that motion analysis does not depend on complex-object descriptions. Our work follows this trend and is based on the recent publications that are in agreement with this idea in computational vision. The idea of object/event recognition regardless of the existence of object representations can be traced back to the early 70's when Johansson [1976] introduced his experiments with *moving-light displays*. The idea was to attach lights to the joints of a human subject dressed in dark-colored clothing and observe the motion of lights against a dark background. The audience not only could recognize the object (human being) but could also describe the motion and the events taking place. Goddard [1992] investigated the high-level representations and computational processes required for the recognition of human motion based on moving-light displays. The idea is that recognition of any motion involves indexing into stored models of the movement. These stored models, called scenarios, are represented based on coordinated sequences of discrete motion events. The structures and the algorithms are articulated in the language of structured connectionist models. Allmen [1991] introduced a computational framework for intermediate-level and high-level motion analysis based on spatiotemporal surface flow and spatiotemporal flow curves. Spatiotemporal surfaces are projections of contours over time. Thus, these surfaces are direct representations of object motion.

In the dynamic computer vision literature there are general models for object motion estimation and representation, as well as domain-restricted models. A general architecture for the analysis of moving objects is proposed by Kubota et al. [1993]. The process of motion analysis is divided into three stages: moving-object candidate detection, object tracking, and final motion analysis. The experiments are conducted using human motion. Another approach to interpretation of the movements of articulated bodies in image sequences is presented by Rohr [1994]. The human body is represented by a three-dimensional model consisting of cylinders. This approach uses the modeling of the movement from medical motion studies. Koller et al. [1993] discuss an approach to tracking vehicles in road traffic scenes. The motion of the vehicle contour is described using an affine motion model with a translation and a change in scale. A vehicle contour is represented by closed cubic splines. We make use of the research results in all these domain-specific motion analysis projects. Our model combines the general area of motion analysis with individual frame (image) analysis.

In case of video modeling, the video footage usually is first segmented into shots. Segmentation is an important step for detection of cut points which can be used for further analysis. Each video shot can be represented by one or more key frames. Features such as color, shape, and texture could be extracted from the key frames. An approach for automatic video indexing and full video search is introduced by Nagasaka and Tanaka [1992]. This video-indexing method relies on automatic cut detection and selection of first frames within a shot for content representation. Otsuji and Tonomura [1993] propose a video cut detection method. Their projection detection filter is based on finding the biggest difference in consecutive-frame histogram differences over a period of time. A model-driven approach to digital video segmentation is proposed by Hampapur et al. [1994]. The paper deals with extracting features that correspond to cuts, spatial edits, and chromatic edits. The authors present an extensive formal treatment of shot boundary identification based on models of video edit effects. In our work, we rely on these methods for the initial stages of video processing, since we need to identify shot boundaries to be able to extract meaningful information within a shot.

One representation scheme of segmented video footage uses key frames [Arman et al. 1994]. The video segments can also be processed for extraction of synthetic images, or layered representational images, to represent closely the meaning of the segments. A methodology for extracting a representative image, *salient video stills*, from a sequence of images is introduced by Teodosio and Bender [1993]. The method involves determining the optical flow between successive frames, applying affine transformations calculated from the flow-warping transforms, such as rotation, translation, etc., and applying a weighted median filter to the high-resolution image data resulting in the final image. A similar method for synthesizing panoramic overviews from a sequence of frames is implemented by Teodosio and Mills [1993].

Swanberg et al. [1993] introduced a method for identifying desired objects, shots, and episodes prior to insertion in video databases. During the insertion process, the data are first analyzed with image-processing routines to identify the key features of the data. In this model, episodes are represented using finite automata. Only video clips with inherently well defined structure can be represented. The model exploits the spatial structure of the video data without analyzing object motion. Zhang et al. [1994] presented an evaluation and a study of knowledge-guided parsing algorithms. The method has been implemented for parsing of television news, since video content parsing is possible when one has an a priori model of a video's structure.

Another system, implemented by Little et al. [1993], supports content-based retrieval and playback. They define a specific schema composed of movie, scene, and actor relations with a fixed set of attributes. Their system requires manual feature extraction. It then fits these features into the schema. Querying involves the attributes of movie, scene, and actor. Once a movie is selected, a user can browse from scene to scene beginning with the initial selection. Weiss [1994] presented an algebraic approach to content-based access to video. Video presentations are composed of video segments using a *video algebra*. The algebra contains methods for temporally and spatially combining video segments, as well as methods for navigation and querying. Media Streams is a visual language that enables users to create multilayered iconic annotations of video content [Davis 1993]. The objects denoted by icons are organized into hierarchies. The icons are used to annotate the video streams in a *Media Time Line*. The Media Time Line is the core browser and viewer of Media Streams. It enables users to visualize video at multiple time scales simultaneously, in order to read and write multilayered, iconic annotations, and it provides one consistent interface for annotation, browsing, query, and editing of video and audio data.

The work presented here follows from a number of efforts listed above. Specifically, we use low- and intermediate-level motion analysis methods similar to those offered by Allmen [1991] and others. Our object recognition ideas have been influenced by the work of Jain and his students [Gupta et al. 1991a; 1991b], Grosky [Grosky and Mehrotra 1989], and the research in image databases. Several lines of research such as those in Little et al. [1993], Swanberg et al. [1993], Zhang et al. [1994], and Weiss [1994] provided many useful ideas for the modeling aspects of our investigations. An early report of our work was presented in Dimitrova and Golshani [1994].

## 3. MOTION RECOVERY IN DIGITAL VIDEO

In this section we describe in detail each level of the motion analysis pipeline. At the low-level motion analysis we start with a domain of motion vectors. During intermediate-level motion analysis we extract motion trajectories that are made of motion vectors. Each trajectory can be thought of as an n-tuple of motion vectors. This trajectory representation is a basis for various other trajectory representations. At the high-level motion analysis we associate an activity to a set of trajectories of an object using domain knowledge rules.

### 3.1 Low-Level Motion Extraction: Single Macroblock Tracing

In MPEG, to encode a macroblock in a predicted or a bidirectional frame, we first need to find the best matching macroblock in the reference frames, then find the amount of $x$ and $y$ translation (i.e., the motion vector), and finally calculate the error component [Patel et al. 1993]. The motion vector is obtained by minimizing a cost function that measures the mismatch between a block and each predictor candidate. Each bidirectional and predicted frame is an abundant source of motion information. In fact, each of these frames might be considered a crude interpolation of the optical flow. Thus, the extraction of the motion vectors of a single macroblock through a sequence of frames is similar to low-level motion analysis.

Tracing a macroblock can continue until the end of the video sequence if we do not impose a stopping criterion. We have a choice: to stop after a certain number of frames, stop after the object (macroblock) has come to rest, stop if the block comes to a certain position in the frame, stop if the macroblock gets out of the scene, or stop if the macroblock is occluded.

The algorithm for tracing the motion of a single macroblock through one frame pattern for MPEG encoding is given in Figure 2. In Dimitrova [1995], we describe object motion tracing for video databases in more detail. The algorithm takes the forward and backward motion vectors that belong to a particular macroblock and computes the macroblock's trajectory. The algorithm computes the macroblock's position in a B-frame by averaging the positions obtained from: (1) the previous block coordinates and forward motion vectors and (2) next (predicted) block coordinates and the backward motion vector. The position of a macroblock in a P-frame is computed using only block coordinates and forward motion vectors. If during the tracing procedure the initial macroblock moves completes out of its position, then we have to extract motion vectors for the new macroblock position, which implies that we are continuing by tracing the macroblock whose position coincides with the $(x, y)$ coordinates of the initial macroblock. In the rest of this article, we will use $\tau$ to indicate the set of all possible motion vectors.

3.1.1 *Trajectory Description.* Various motion retrieval procedures have specific requirements for retrieving desired objects. These requirements depend on the characteristics of the retrieval which may be flexible to strict. The choice of trajectory representation may dictate the manner in which retrieval is conducted. Given a set of motion vectors for a macroblock, a number of mechanisms exist for trajectory representation. Below we present a sample list:

(1) *Point Representation:* A trajectory in this case is a set of points represented by the absolute or relative frame coordinates of the position of the object, say

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where $(x_i, y_i)$ is derived by projecting $(x, y, i)$ onto the image plane. $(x, y, i)$ denotes the position of an object, i.e., $(x, y)$, at time instant $i$.

(2) *Curve Representation:* A parametric B-spline curve $P(u)$ can be computed that passes through each of the trajectory points $(x_i, y_i)$ (see Farin [1990] for a detailed discussion). The first step involves generating a parameteri-

```
Given: frames F =  Fi i = 0,...,n;
motion vectors V = (fmx(i),fmy(i)),(bmx(i),bmy(i)) i= 1,n
initial block coordinates bx, by
 Initialize $R = \emptyset$,
  for i=1,..., n
    if F(i) $\neq$ I then
      if F(i) == P then
        if previousType == I
          cx = bx - fmx(i)/2;
          cy = by - fmy(i)/2;
          nextblockx = cx; nextblocky = cy;
        if previousType == P
          givenx = futurex;
          giveny = futurey;
          futurex = futurex - fmx(i)/2;
          futurey = futurey - fmy(i)/2;
      if F(i) == B then
        cx=((givenx-fmx(i)/2)+(futurex-bmx(i)/2))/2;
        cy=((giveny-fmy(i)/2)+(futurey-bmy(i)/2))/2;
      if block(bx,by) $\cap$ block(cx,cy) == $\emptyset$ then
        extract(mx(i),my(i)) for (cx,cy)
      R = R $\cup$ {(mx(i),my(i))}
        if F(i) is the last in a group of B frames before a P frame
          cx = futurex;
          cy = futurey;
      if block(bx,by) $\cap$ block(cx,cy) == $\emptyset$ then
        extract(mx(i),my(i)) for (cx,cy)
      R = R $\cup$ {(mx(i),my(i))}
    if F(i) == I then
      (bx,by) $\leftarrow$ bestMatch(bx,by) in I
    if stopping criteria == true, then
        return R;
   endfor
```

Fig. 2. Algorithm for tracing the motion of a macroblock.

zation or *knot sequence* $u_1 \leq u_2 \leq \cdots \leq u_n$. A commonly used approach employs cumulative chord lengths defined by the points $(x_i, y_i)$. The next step involves setting up and solving a tridiagonal linear system of equations whose unknowns are the control points $d_i$ of the B-splines $N_i(u)$. The linear system depends on the $x_i$, $y_i$, and $u_i$ values. This linear system can be efficiently solved in $\mathscr{O}(n)$ time using standard techniques for tridiagonal matrices. The B-spline curve has the form:

$$P(u) = \sum d_i N_i(u),$$

and it satisfies the following:

(a) $P(u_i) = (x_i, y_i)$;

(b) $P(u)$ is a piecewise cubic polynomial, i.e., for $u_i \le u \le u_{i+1}$, $P(u)$ is a polynomial of degree less or equal to three; and
(c) the first and the second derivatives of $P(u)$ are continuous.

(3) *Chain Code Representation:* We develop a piecewise linear approximation to the trajectory using a set of orientation primitives. Given a set of discrete trajectory orientation primitives, we use a zig-zag line representation of the trajectory to generate the code. Another way of viewing this approach is derived from a neighborhood matrix with each neighbor coded to correspond to the primitives in the figure [Schalkoff 1989].
(4) *Differential Chain Code Representation:* Each segment is coded relative to the next line segment using the direction (left or right) and the length. For example, we can have a code for: right shorter–1, right equal–2, right longer–3, left shorter–4, left equal–5, left longer–6 [Schalkoff 1989]. This scheme is useful for approximate matching of object trajectories. It is a rotation-, scaling-, and translation-invariant scheme.

Figure 3 illustrates these methods used for the representation of an arbitrary movement. Figure 3(a) is an exact coordinate representation; 3(b) is a B-spline curve representation. Figure 3(c) represents the chain-coding process, and 3(d) shows the differential chain code representation of the trajectory.

Note that in the coordinate representation and B-spline and chain code representation schemes we have a way of representing zero motion, i.e., when the motion vector is a null vector. If the macroblock does not move over a certain number of frames, the point will be repeated. In the B-spline representation, the knot (i.e., the control point) will have a multiplicity greater than one. In the chain code representation, the zero motion is represented by the code "0." So, in all these representations the trajectory is not only a spatial representation of the object's motion (the path) but also a temporal characterization of the motion. By keeping track of the zero motion we are able to describe stationary objects as well.

The diversity of the trajectory representations makes the querying process more flexible. The actual method of representation does not have a significant impact on the querying process as long as modeling, representation, and querying are all done in the same fashion.

3.1.2 *Trajectory-Matching Functions.* Applications such as automated surveillance may require retrieval of either video sequences or objects contained in these sequences based on the object trajectories. For example, queries of the type "retrieve objects that have a motion trajectory whose point of origination is the main gallery door and terminate at the Juan Miro's picture on the opposite wall" may help in the identification of the person who damaged the picture.

Matching functions used for motion retrieval depend on the method employed for trajectory representation, as described below.

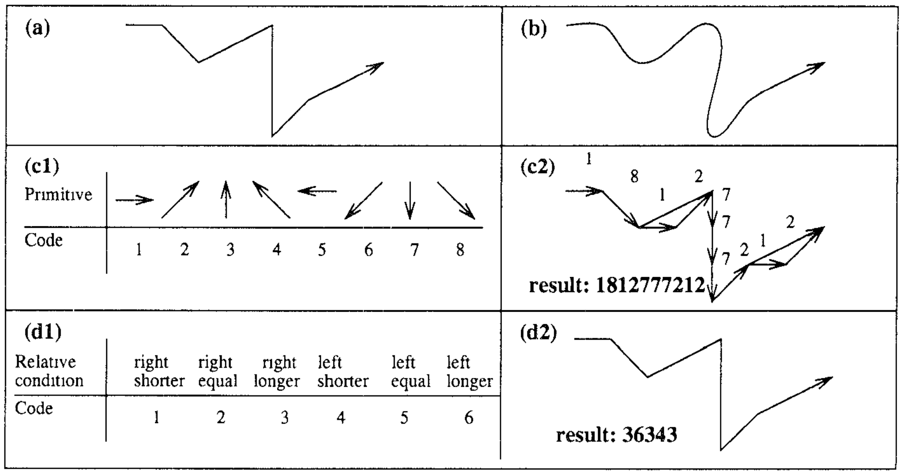


Fig. 3. Alternatives for object motion representation: (a) motion trajectory; (b) B-spline curve representation; (c1) chain-coding scheme; (c2) chain code representing the trajectory; (d1) differential chain-coding scheme; (d2) resulting differential chain code.

— Exact matching function that uses absolute frame coordinates (least-square minimization problem). This matching function has two variations:

(1) exact start position and exact trajectory match

(2) any start position and exact trajectory match.

— Exact matching function that uses relative coordinates. This function is used when the initial position of the object is not important.

— Curve comparison based on the curve-fitting approach used for interpolated trajectory representation.

— Approximate matching that uses chain code:

(1) exact start position and inexact trajectory match

(2) any start position and inexact trajectory match.

The chain code matching translates the problem of trajectory matching into a pattern-matching problem.

— Qualitative matching that uses differential chain code.

The result in each case is a similarity factor between the input trajectory and a target trajectory in the set of object trajectories.

## 3.2 Intermediate-Level Motion Analysis

A macroblock trajectory is the spatiotemporal representation of the macroblock's motion. These trajectories are further used for extracting object motion. This process is different for rigid and nonrigid bodies. A rigid object consists of one solid part to which motion trajectory is associated. If the object consists of several parts which themselves represent rigid objects with independent movements, then, such a nonrigid object is represented as a set of rigid objects with their respective trajectories. At the highest level of motion analysis, we associate "activities" with the object trajectory representations.

Rigid-object motion is represented by a single trajectory. The trajectory is one common representation of the trajectories of all the component macroblocks. Finding the most-representative trajectory is not a simple task. In the simplest case we can take the trajectory of the object centroid as the reference object trajectory. A more-complicated case occurs if we decide to create a common trajectory by processing all of the macroblock trajectories or by examining only a subset of all macroblock trajectories.

Mean averaging of all trajectories of the macroblocks of the object is an alternative to choosing the object centroid's trajectory. The averaging of the trajectories in the exact form is pointwise averaging of the trajectories at each frame.

The following two assumptions make the object motion recovery feasible:

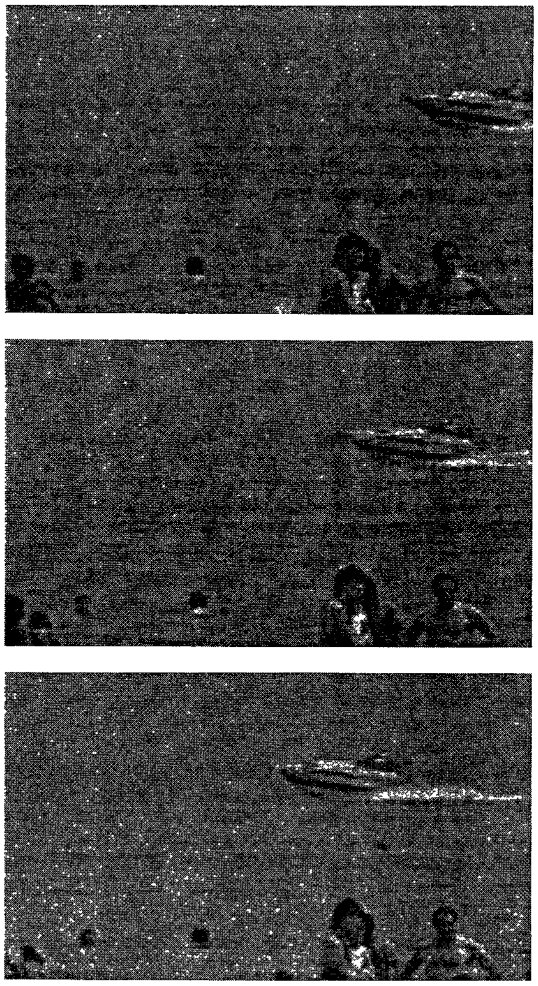
(1) *Integrity of Objects:* We assume objects are rigid or consist of rigid parts connected to each other. We do not consider situations in which objects disintegrate. This assumption is important because we only use object trajectory representation.
(2) *Motion Continuity:* Each macroblock under consideration has continuous motion. This assumption is important for the trajectory representation, since every trajectory segment represents continuation of the previous trajectory segment.

Averaging trajectories is used for determining a representation of a nonrigid body motion. For nonrigid objects, we must determine the number of trajectory clusters and their locations. Each cluster corresponds to a single coherent motion that represents a moving part (i.e., a rigid object). We use a hierarchical clustering algorithm (due to Duda and Hart [1973]) for determining the number of rigid object parts. Initially, the algorithm begins with clusters that contain only one trajectory each. At each subsequent step, we attempt to merge those neighboring clusters that have a similar trajectory. Individual trajectories, in this case, will be averaged to compute a trajectory for the extended cluster.

An example of a traced object through 20 encoded frames using the IBBPBBBP frame pattern is given in Figure 4. Figure 4(a) contains first, middle, and last frames of a video sequence capturing a water skiing scene. Figure 4(b) contains the motion trace for the moving yacht. The axes in Figure 4(b) correspond to the $x$ and $y$ axes of the video frames where the (0, 0) coordinate is at the top left corner.

Figure 5 shows six out of the 60 frames of the "Walfky" video sequence used for our next experiment. The object being traced is a small toy which performs very uneven motion. Figure 6(a) shows how the tracing of a macroblock progresses when every frame in the sequence is used. The frame pattern IBBBPBBB is used for video encoding when macroblock trajectories are extracted in Figure 6. This experiment shows that the macroblock tracing

(a)



(b)

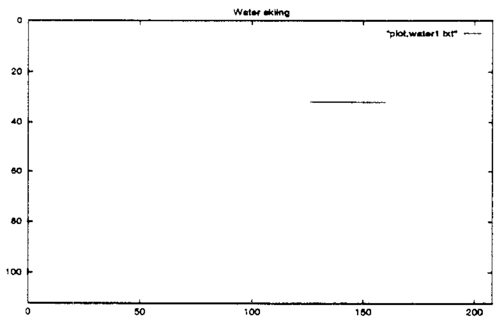


Fig. 4. Tracing a moving yacht. (a) first, middle, and last frames in video sequences; (b) motion trace of the yacht.

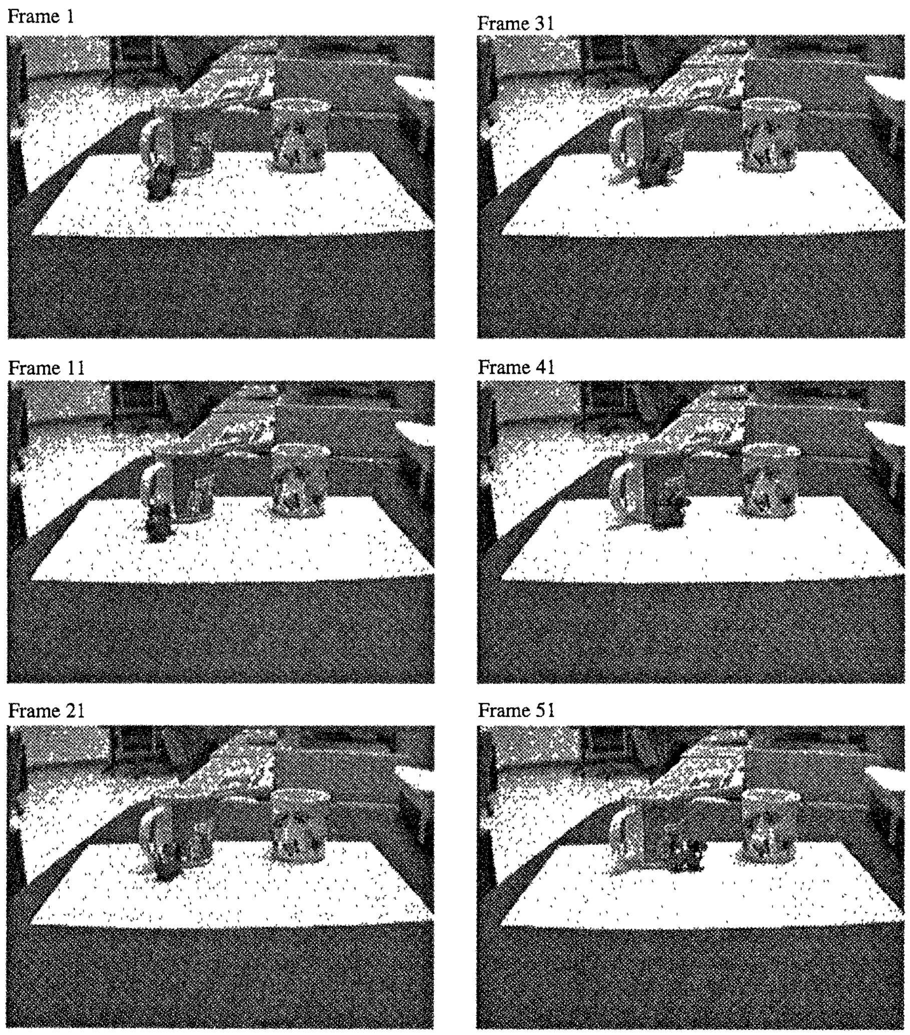


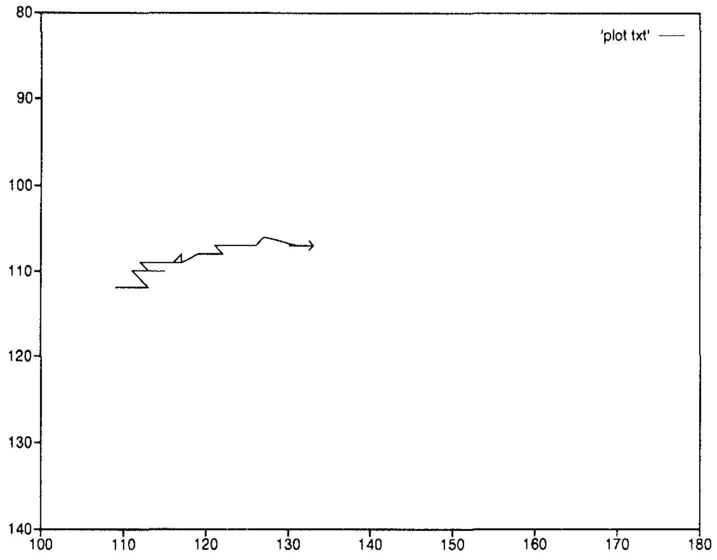
Fig. 5. Snapshots from a video sequence.

is possible when the objects exhibit jerky motion. As expected the trajectory is not only curved but also has the properties of a zig-zag line. In this case the macroblock with coordinates (14, 14) is traced. In terms of absolute frame coordinates, these coordinates correspond to (112, 112). Figure 6(b) shows tracing of the same video sequence in the case when every other frame is used for encoding and tracing.

We use the notation $T$ to indicate the set of object trajectories. Each member of $T$ is a sequence[1] whose range is the set of all motion vectors $\tau$, i.e., $\forall t \in T(t : \mathcal{N} \rightarrow \tau)$, where $\mathcal{N}$ is the set of natural numbers. In other words

[1] A sequence is simply a function whose domain is the natural numbers.

(a) Trace of the macroblock (14,14) using all frames



(b) Trace of the macroblock (14,14) using every other frame

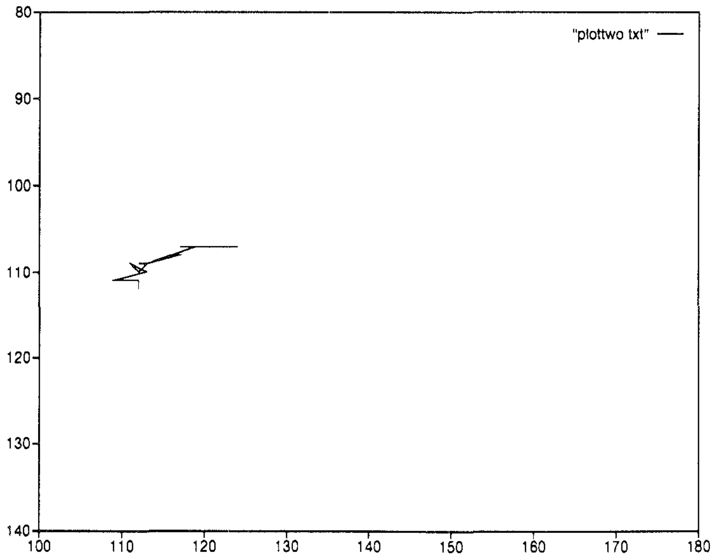


Fig. 6. Traced trajectories in the Walfky video sequence. (a) all frames; (b) only every other frame is used.

each object trajectory is a sequence of motion vectors identifying macroblock displacement for the components of the object. As discussed previously, the actual appearance of the members of $T$ depends on the choice of the representation scheme.

## 3.3 High-Level Motion Analysis

At the highest level of motion analysis, we associate domain-dependent "activities" with the object trajectory representations. An activity can be recognized by the system based on a predefined set of procedures, or it can be designated by the user. We realize that recognizing activities is one of the most-difficult tasks in any vision system. Such undertaking requires information on:

(1) Relative positioning between rigid subparts
(2) Relative timing of the parts movements
(3) Actual and perceived interaction of object parts.

The two main problems in recovering high-level motion representation are (1) the fact that multiple sequences are occurring simultaneously (for example, arm movements and leg movements in human motion) and in a coordinated fashion and (2) tempo changes are global (in the case of the human body, the changes apply to all four limbs and occur slowly).

An activity involves both spatial and temporal representations of the objects of interest. We must identify the object components (shape and other features) and their respective trajectories (as we did in the previous section) at the intermediate-level motion analysis and then assemble activities. The temporal information is needed for discrimination of activities of the same type, for example, strolling, walking, hurrying, etc. After assembling object activities, based on additional knowledge, we can infer event information.

We use $\mathscr{A}$ to symbolize the set of activities. We assume the existence of a knowledge base $\mathscr{K}$ whose contents include all the necessary rules, constraints, and the procedures for deriving activities from lower-level descriptions.

Each member $a$ of $\mathscr{A}$ is a "composition" of $t_1, t_2, \ldots, t_n$, where for every $1 < i < n$ we have:

— $t_i \in T$ and
— $t_i$ satisfies every constraint in $\mathscr{C}_a$, where $\mathscr{C}_a \in \mathscr{K}$ represents the constraints governing the activity $a$.

High-level event representation and manipulation call for the use of either temporal Petri nets, an event-based approach to temporal objects, or other event representation and manipulation schemes.

## 3.4 Spatiotemporal Hierarchical Representation

We use a semantic multiresolution hierarchy for spatiotemporal representation (Figure 7) because it helps video analysis at various resolution levels, with coarser resolutions used for high-level event/scenario descriptions. The

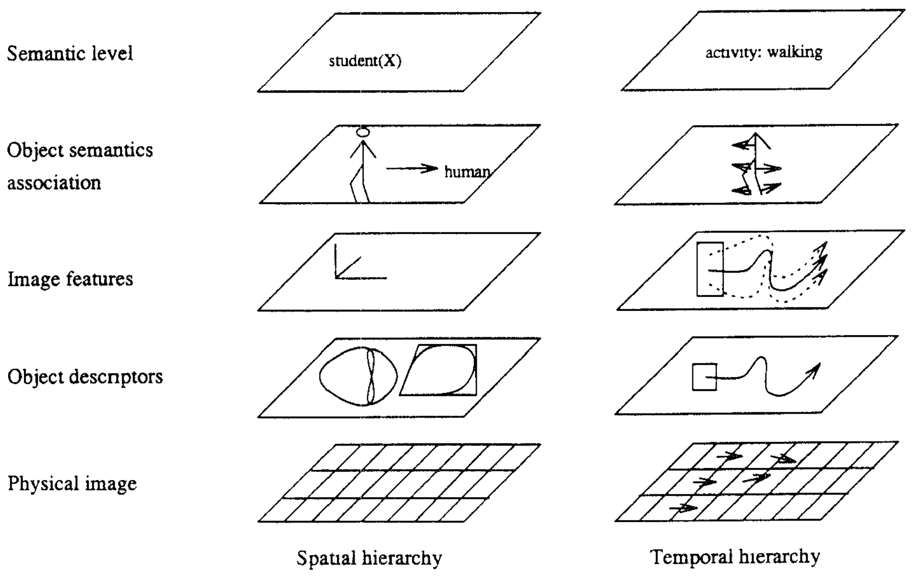


Fig. 7. Multiresolution hierarchy for spatial and temporal video representation.

advantage of multiresolution representation is that it offers a mechanism to make the trade-off between the competing demands of fine spatial/temporal resolution and low computational complexity. The idea of representational hierarchy for still images has been utilized by several image data models [Grosky and Mehrotra 1989; Gupta et al. 1991a; 1991b].

At successive time intervals, a frame is inserted at the base of the spatial hierarchy, and the features are computed for the next levels. The motion features are computed starting at the frame, and the temporal part of the hierarchy is filled with the appropriate motion descriptions. Motion analysis starts with the motion vector recovery (bottom of the temporal hierarchy, Figure 7). At the next level, individual macroblock trajectories are traced. At the intermediate level, rigid-body motion is recovered, followed by nonrigid-motion recovery. Finally, at the highest level of motion analysis, description of activities is derived from previously computed motion features (top of the temporal hierarchy, Figure 7).

The temporal part of the hierarchy can be used for various kinds of motion retrieval ranging from full-object trajectory-based matching to single-macro-block trajectory matching. Since we provide exact and inexact trajectory representation, our retrieval functions can take inputs in terms of precise spatial coordinates, orientation coordinates, and qualitative descriptions.

## 4. INFORMATION FILTERING AND DIGITAL VIDEO

The motion-tracing and representation scheme introduced in previous sections serves as a basis for the classification and retrieval of video sequences. Video sequences may be retrieved using the temporal (motion) part of the hierarchy or the combination of spatial and the temporal representations. The idea of this representation is that we can compute the spatial and temporal features independently of each other. We emphasize that temporal features coupled with spatial features are important in discriminating and classifying video sequences.

Like other knowledge representation cases, we do not attempt to have a universal system that can recognize and distinguish all possible objects. Such general-purpose (i.e., domain independent) representations have been shown to be too complex for present technologies. Thus, we assume that the domain of interest is known a priori and that the video classification system will be confined to working on only those objects. Consider a domain D, called the "scope," containing all objects of interest. Formally, the elements of D are defined as object-oriented structures with potentially complex internal components. Similar to any object-oriented representation, the user can identify the objects of D by their attributes, such as object ID, image descriptions, name, and shape (or convex hull), or a combination of these. Thus, the user may provide any available information on any of the attributes of desired object (for example, object ID, or shape together with a partial description), and the system will attempt to identify the intended object. Although we do not make any assumptions on how the elements of D and their attributes are represented, we offer the following example as an indication of a typical structure.

*Example* 4.1. A walking human may be represented as a moving object $a_1 = (o_1, m_1, v_1)$ where

$$\begin{aligned} o_1 &= (\mathit{category} : \mathit{human}, \mathit{convexHull} : o_6 : \mathit{skeleton} : o_7, \\ &\qquad \mathit{parts} : \{\mathit{head} : o_2, \mathit{torso} : o_3\}), \\ m_1 &= (\mathit{trajectory} : 2467332, \mathit{activity} : \mathit{walking}), \text{ and} \\ v_1 &= (v\# : 234, \mathit{firstFrame} : 45, \mathit{lastFrame} : 485). \end{aligned}$$

Similarly, the head and torso also have their spatial and motion descriptions.

In a database containing only "still" images, a correspondence table of the form (O, I)—where O stands for the object, and I stands for the image—will suffice. In a video database, we have the added parameter of temporal changes. Although the motion of each object can be modeled as an attribute of the object, say, "dog, big, brown, running," it is more appropriate to separate objects and their motions as two different parameters. Note that if motion is considered as just another attribute of the object, then in case the same object appears more than once, each time with a different motion, we would need multiple, different entries into the database. For example, there would be multiple entries for the big brown dog: running right, running left, running in circles, and jumping.

Video sequences are identified by objects present in the scene and their respective motion. The goal of the motion analysis is to extract activity and event representation. An index entry of an activity in a video sequence has the following form:

$$(O, M, V) = (objectRep, motionRep, vid)$$

— *objectRep:* an object represented by its extracted features (convex hull, object skeleton, centroid, texture, set of macroblocks covering the object, etc.; see left side of Figure 7). An object representation might include a set of object representations of the constituting parts (e.g., objects that represent a human figure include head, torso, arms, and legs object representations).

— *motionRep:* an object trajectory specified by objectRep, velocity, trajectory curvature, torsion, and activity description (see right side of Figure 7).

— *vid:* identity of the video subsequence to which an object belongs to: *vid* consists of (*viSeqId*, *firstFrame*, *lastFrame*).

(1) *viSeqId* is a video sequence identity which is unique for a sequence across the whole video database.
(2) *firstFrame:* the first frame in which the specified object appears.
(3) *lastFrame:* the last frame in which the specified object appears.

## 4.1 Content-Filtering Operators

The OMV triplet is the basis for the query functions. There are many possibilities for the selection of filters (in this context, query functions.) A sample selection is presented below. These operators may be used in a relational form, mostly in a table lookup mode, or may be embedded into a more-elaborate query language, as presented in Section 4.2. Recall that $\mathscr{P}(A)$ is used to denote the powerset of the set A, i.e., the set of all subsets of A.

$$V\_Seq : O \times M \to \mathscr{P}(V)$$

This function takes any description that can be provided at any level of the spatial hierarchy. The input might be a characterization of the object in terms of its bounding polygon, stick figure, a name, or concept. At this point we need to emphasize that we use the properties of the object-oriented nature of the representation of the objects. For example, the expression

$$V\_Seq(O\_category = pet, (Activity = walking, Trajectory = t_1))$$

translates into "retrieve all the video sequences in which a pet walks and makes a trajectory $t_1$." The answer will include all the objects (animals) that are classified as pets: cats, dogs, fish called Wanda, etc. It is important to note that here we discuss only the formal framework in an informal way and that these functions are implemented within an interactive window-based graphical query interface which we discuss in Section 5.1.

The function

$$Object\_motion : O \times V \to \mathscr{P}(M)$$

takes any object description and a particular video sequence and returns a set of motion descriptions related to that object. In order to detect which objects performed a particular type of motion, i.e., the agent of the action, we use a function of the following family:

$$Agents: M \times V \rightarrow \mathscr{P}(O).$$

The next function is used to get a detailed description of all the objects and their respective motions in a video sequence:

$$Describe\_Video: V \rightarrow \mathscr{P}(O \times M).$$

If we just want information about the spatial characteristics of objects in a sequence, we use the function

$$Objects: V \rightarrow \mathscr{P}(O).$$

This is equivalent to the "agents" function where the first argument is unimportant. Thus, given a $v_i \in V$, $Objects(v_i) = Agents(any, v_i)$.

The above functions allow for inexactness, and by default, they return results that are approximately similar to the precise answer. To make the operator exact, we use a higher-order operator that converts the query function in the desired manner, in this case, makes it exact. There are several types of these operators, e.g., exact, partial, and similar.

For making the retrieval exact, the symbol ! is placed in front of the query function. For example, "!*Agents*" returns only objects that have exactly the same motion description as the one given in M. Similarly, "!*Object_motion*" returns only motion descriptions of objects whose spatial characteristics (for example, exact bounding polygon, texture) match the spatial characteristics of a given object.

The # symbol placed in front of the query function is used for partial retrieval. Partial retrieval means that any of the motion or temporal characteristics of the given object should match. For example, "#*Agents*" will return all objects that match at least one of the motion descriptors.

## 4.2 The Query Language

The retrieval functions introduced in the previous section are embedded into the framework of a multimedia functional query language called EVA, described in Golshani and Dimitrova [1994]. EVA is the interface to a multimedia database system capable of storage, retrieval, management, analysis, and delivery of objects of various media types, including text, audio, images, and moving pictures. The language deals with the temporal and spatial aspects of multimedia information retrieval and delivery, in addition to the usual capabilities provided by the ordinary database languages. EVA has five groups of operators, namely: operations for querying and updating (i.e., editing) the multimedia information, operations for screen management, temporal operators, operators for specifying rules and constraints, and aggregation (computational) operators. EVA is an extension of a functional query language whose notation is based on that of conventional set theory. Both the original language and its extensions are formally defined in an algebraic framework. EVA is object oriented and supports objects, object classes, attributes and methods of objects, and relationships between objects. It has been ported onto several different platforms.

EVA provides a wide collection of operators that deal with text, graphics, scanned images, audio, and video. In addition, there are numerous type-independent operators such as set operators like union and set membership. One of the general operations is: the set construction operator. Generally, this has the form $\{f(x) \mid P(x)\}$, where $f(x)$ denotes the desired output objects, and $P(x)$ denotes the retrieval predicate which has to be true for those objects.

— *set operation symbols:* isin, isSubsetOf, isTrueSubsetOf, union, intersection, difference, Union, Intersection, noOf.
— *equality operators:* is, isnot.
— *temporal synchronization* (*for all media types*): sim, before, meets, equals, starts, at, finishes.
— *spatial composition* (*applied only to graphics, images, and video*): left, right, bottom, up, showIn, arrange.
— media-dependent operation symbols include
  —*text:* appendPar, cutPar, eqPar, keyword, isKeywordIn, parSim.
  —*graphics:* insPatch, pictureSum, fill, domain, colors, getPatch, getColor, restriction, scale, translate, dot, lineSeg, box, coincident, contains, disjoint, visible, bounded.
  —*images:* shift, zoom, superimpose, overlay, imageSim.
  —*audio:* intensity, extract, audioIns, audioLen, audioSim.
  —*video:* videoLen, pace, videoClip, videoIns.
—*integer operation symbols:* $+$, $-$, $*$, $<$, $>$, $<=$, $>=$, min, max, ave, sum, prod.
—*string operation symbols:* concat, strLen.
—*logical operation symbols:* and, or, implies, not.

To demonstrate the capabilities of EVA and how queries are constructed, we present a simplified example. The first part of the example will demonstrate the language without the OMV extensions. The video content retrieval extensions will be discussed once the appropriate distinctions are made.

We present the schema of a multimedia database system and then provide a few sample queries. The schema is represented as a graph whose nodes are object classes (in algebraic terms, sorts) and whose arcs are the relationships between object classes (represented as functions). Readers familiar with the algebraic framework would recognize this as a many-sorted algebra.

Illustrated in Figure 8, the schema models a college basketball multimedia database. The *basic types* in this system are String, Integer, Audio, Text, Video, while Player and School are *user-defined data types*. The highlighted portion appearing in dotted lines relates to the extension for video content retrieval described in Section 4.3.

The main difference between these basic and user-defined types is that the former constitute the application-independent constituents of any schema, whereas the user-defined types depend on each individual application. In

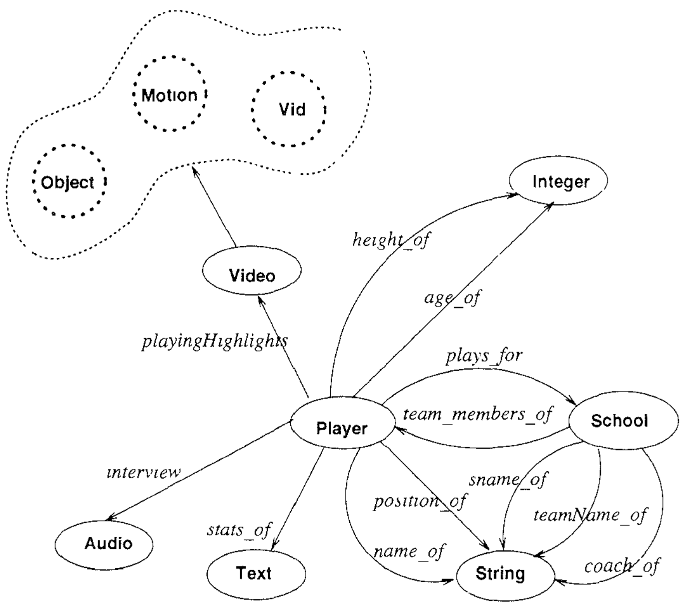


Fig 8. Basketball schema.

Figure 8, all object types, both basic and user defined, appear in ovals. The attributes of objects and their relationships that are captured by arrows are the following functions:

| | |
|---|---|
| $name\_of$ | $: Player \rightarrow String$ |
| $position\_of$ | $: Player \rightarrow String$ |
| $state\_of$ | $: Player \rightarrow Text$ |
| $interview$ | $: Player \rightarrow Audio$ |
| $playingHighlights$ | $: Player \rightarrow Video$ |
| $height\_of$ | $: Player \rightarrow Integer$ |
| $age\_of$ | $: Player \rightarrow Integer$ |
| $plays\_for$ | $: Player \rightarrow School$ |
| $teammembers\_of$ | $: School \rightarrow \mathcal{P}(School)$ |
| $sname\_of$ | $: School \rightarrow String$ |
| $teamName\_of$ | $: School \rightarrow String$ |
| $coach\_of$ | $: School \rightarrow String$ |

Here are some queries on this database.

(1) List all guards who are taller than 190cm.

$$\{name\_of(P) \mid position\_of(P) is\text{“}Guard\text{”}\ \ and\ height\_of(P) > 190\}$$

This is a simple query on the nonmultimedia portion of the database. $P$ is a variable of type *Player*. The result is a list of names of players who satisfy the given conditions.

(2) Play video clips of all centers, and simultaneously display their statistics.

$$\{(name\_of(P), stats\_of(P))\, sim\ playingHighlights\_of(P) \mid position\_of(P) is \text{“}Center\text{”}\}$$

While variable $P$ ranges over the elements of type *Player*, whenever the condition on position is satisfied, the name, statistics, and the corresponding video clip of the qualified player are displayed. “sim,” standing for “simultaneously,” is one of the synchronization operators that ensure proper semantics for presentations.

(3) Display the statistics of all Phoenix State University guards, and show their highlights before playing their interviews.

$$\{(name\_of(P), stats\_of(P))\, sim(playingHighlights\_of(P)\ \mathrm{before} interview(P)) \mid plays\_for(P) is \text{“}PhoenixStateUniversity\text{”} and\ position\_of\ (P) is \text{“}Guard\text{”}\}$$

The result of this query is that, for every guard of the appropriate school, while their name and statistics are displayed, their video clip is presented first, and then their respective interview is played. The term “before” is another synchronization operator.

## 4.3 Querying Video Contents

Note that in the above queries, we treated *Video*, *Audio*, and *Text* as basic types in a similar manner to the type Integer, i.e., as objects whose contents can be displayed or presented, but no further specific characteristics are known about the contents. Our motion recovery algorithm and specifically the OMV functions enable us to treat *Video* in a different way, as described below.

Grosky's [1994] categorization makes a distinction between the physical basic data types and the conceptual data types. He adopts a generic model to represent *content-independent* and *content-based* properties of multimedia objects. Content-independent properties are related to the physical data object itself (uninterpreted data) as well as synchronization and storage information. Content-based properties refer to relationships between nonmultimedia real-world application entities and multimedia objects. The content-based properties associate semantics to the object at various levels.

A binary object containing the video stream that corresponds to the playing highlights of a particular player is an instance of physical data type. The extracted spatial and motion characteristics are stored in the conceptual data type. The queries on the content of the video data are directed to the conceptual video data type.

The *conceptual video data type* is molded from the spatiotemporal hierarchy presented in Figure 7 using the object-motion-video structures. The OMV retrieval functions augment EVA's retrieval capabilities since they turn the physical object Video into a conceptual one, i.e., an object with its own specific set of properties that can be incorporated into queries for more-precise questions. The extension to the schema which enhances the video type to be a conceptual type appears in the dotted line in Figure 8. Below is a list of operators that augment the retrieval capabilities based on the OMV retrieval functions:

— *Function Composition:* given functions $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, the composition is $f \circ g(x) = g(f(x))$. For *example*, Given an object (any characteristic) in a video sequence $v_1$, retrieve objects in another video sequence $v_2$ which have similar motion.

$$Agents(Object\_motion(o_1, v_1), v_2)$$

— *Temporal Combination Functions:* $f\theta g$ where $\theta \in$ {*before, meets, simultaneously, starts, finishes*}. Although the same syntax is used, these should not be mistaken for the synchronization operators. In this case, no confusion is expected since the context would determine the designation of the operator. An example of usage for this type of operator is the query "retrieve all the sequences in which a tall person is waving while the president walks."

— *Spatial Combination Functions:* $f\xi g$ where $\xi \in$ {*next, behind, inFront, left, right*}.

Using the above combinators and the OMV structure, many new types of queries that refer to the contents of video sequences can be specified. Specifically, we can express queries that refer to the contents of video sequences. Examples include the following:

(1) "Retrieve all the video sequences with the longest successful shots." This query translates into "retrieve all the video sequences for which the length of the trajectory of the ball is maximum."
(2) "Spell out all the details of movements of the players whose height is greater than 200cm." This query is good for analyzing the pattern in which certain players move and achieve the score.
(3) "Find the video sequences in which the player is wearing a blue shirt." The "blue shirt" is inferred using image analysis.

The target language is a visual one that allows for inclusion of spatial properties (sketches) and exact and inexact images. The notation presented in this article is the basis for the visual query interface [Dimitrova 1995; Michael 1994].

## 5. AN ARCHITECTURE FOR VIDEO CLASSIFICATION AND RETRIEVAL

In the previous sections we introduced a model for video classification which exploits motion recovery and representation. In this section, we discuss a general architecture for video database retrieval based on the model. The proposed architecture, as presented in Figure 9, consists of:

— Insertion module

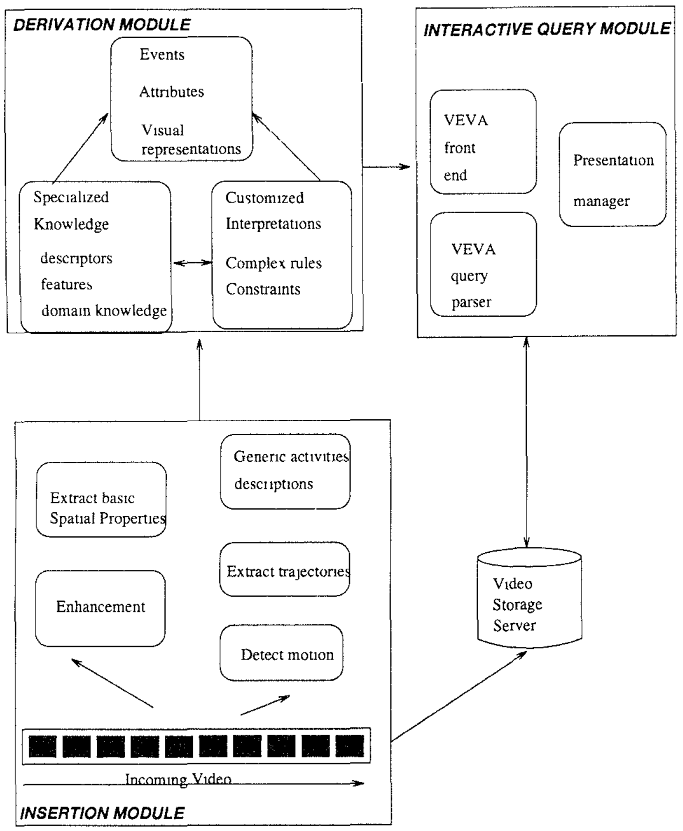


Fig. 9. An architecture for video classification and retrieval.

— Derivation module
— Interactive query module
— Video storage server.

The insertion module is responsible for initial analysis of the incoming video signal. It consists of a suite of operators for image enhancement, operators for the extraction of basic spatial properties, and operators for motion detection and the extraction of motion trajectories. With respect to the spatiotemporal hierarchy, this module is an implementation of the operators between the lowest level of the hierarchy (raw physical data) to the intermediate representation. Currently, the functionalities for spatial analysis are supplied by the Khoros computer vision environment [Rasure et al. 1990]. The extraction of image features, finding regions, and thinning operators are performed by calls to Khoros functions. Although features are automatically extracted, the process of feature selection is manual. For example, we can apply an operator for image segmentation and find the regions in a video frame. However, the selection of regions of importance is decided by the application designer. The automation of this whole process is possible for strictly limited application domains such as industrial monitoring, domain-specific video editing, camera surveillance, and others. The motion detection and tracing operators are also part of the insertion module. The implementation of the motion-tracing algorithm is given in Section 5.2.

The derivation module consists of operators for translation of the extracted features into meaningful descriptions for retrieval. Each application typically defines its own set of meaningful entities and events and has its own interpretation of the same. In our video model and language, the extracted properties are represented by predicates. The derivation module provides the mapping between the visual properties extracted form the video sequences which are geometric by nature and the algebraic representation which is used for querying.

The query module consists of a visual front-end for query composition, a visual query parser, a schema designer, and a presentation manager. The schema designer and the visual front-end are incorporated into the visual query language VEVA [Dimitrova 1995]. The VEVA prototype serves as a testbed for development of new algorithms for video/image segmentation, video parsing, feature selection, and classification. The prototype has been implemented in Tcl/Tk [Osterhout 1994] with the added image and video widgets on top of an existing MPEG encoder [Rowe and Smith 1992].

The video storage server is envisioned to be a disk array serving as a repository of the video sequences. At this point we use a simple file system for storing a limited number of MPEG compressed video sequences.

## 5.1 The Visual Query Language VEVA

Spatial and motion characteristics of objects, derived from images and video sequences respectively, are inherently visual. In this section, we outline the design of a multimedia database language which has well-defined semantics in both character-based and icon-based paradigms.

Defined within the algebraic framework described above, VEVA is a visual query language that provides all the necessary constructs for retrieval and management of multimedia information. The basis for the language is a schema (algebraic signature) which contains entity types (both user-defined and application-independent types) and the associated operators [Golshani and Dimitrova 1994]. By using these operators, the user can visually specify a query for the desired objects in a simple way. VEVA has a formal grammar with which the set of acceptable expressions can be generated. The grammar for the visual language VEVA is given using visual rules in the style of a picture description language which was developed within the syntactic approach to pattern recognition [Schalkoff 1989]. The grammar rules contain nonterminal and terminal icons. The rules are given as graph-rewriting rules where the left-hand side is a nonterminal icon, and the right-hand side is a graph containing nonterminal and terminal icons connected with customized links.

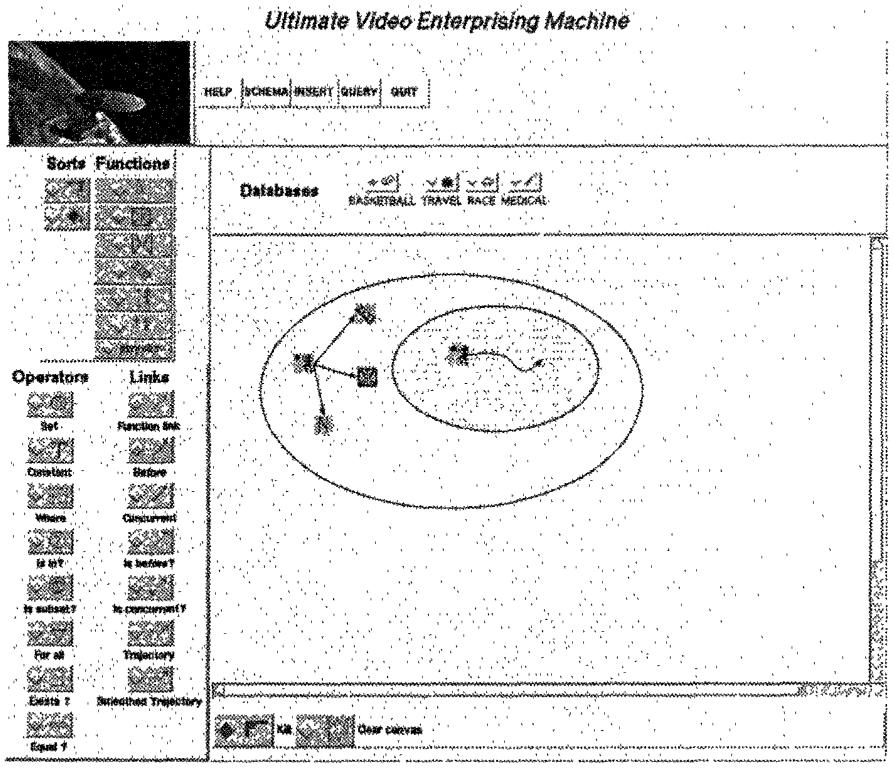


Fig. 10. Visual query involving a trajectory description.

Parsing of visual expressions in VEVA is a process of determining the structure of the workspace. Note that parsing is the first step of the VEVA language processing, because lexical analysis is not necessary. All available icon symbols can be drawn from the given pallette and connected by a set of permissible links. Thus, every expression that is drawn is lexically correct. The execution process begins by parsing the contents of the VEVA workspace. The algorithm finds the top-level set expressions which may contain other set expressions. Translated into visual terms, this algorithm finds the enclosed visual expressions or other iconic elements within a given oval. The algorithm calls the set evaluation procedure recursively for the sets that are contained in it, until there are single sets with simple function-predicate expressions left. The evaluated sets can be connected with temporal links which prescribe the order in which the resulting objects should be presented by the presentation manager. If the evaluated expression contains temporal links, then the parsed execution order is delivered to the presentation manager.

An example query is given in Figure 10. As we stated earlier, VEVA allows for visual queries in which we can specify the path of a moving object. In this example the input trajectory for the player is given as a smoothed trajectory. The visual query given in Figure 10 will select those video sequences from the

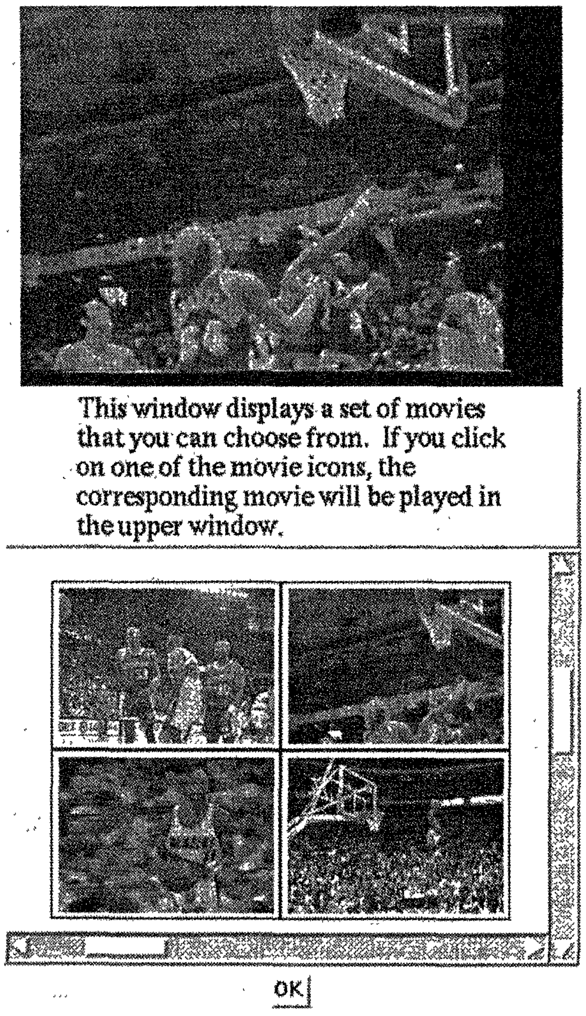

This window displays a set of movies that you can choose from. If you click on one of the movie icons, the corresponding movie will be played in the upper window.

Fig. 11. Results from the visual query. Courtesy of NBA Entertainment.

repository in which the player's trajectory is similar to the one drawn by the user and display the name and the position of the player. The result of the query is shown in Figure 11. The user can browse and play the selected video segments.

Various models have been proposed for temporal synchronization, composition, and presentation in multimedia applications, for example, Buchanan and Zellweger [1993] and Little et al. [1991]. On the other hand, a number of models for content-based access of digital video has been proposed [Arman et al. 1994; Bobick 1993; Rowe et al. 1994; Swanberg et al. 1993; Zhang et al. 1994]. However, a general formal model and a language for content representation, composition, and querying of digital video based on the temporal and the spatial properties of objects found in the video sequences has not been

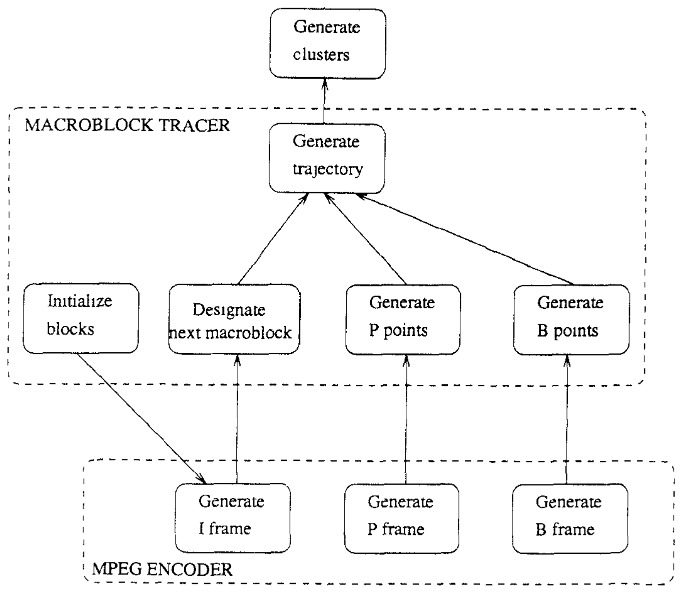


Fig. 12. Macroblock motion extraction.

offered yet. Our video model and language VEVA attempts to unify the presentation aspects as well as content representation aspects of multimedia objects.

### 5.2 Implementation of Macroblock Tracing

The motion-tracing algorithm is a part of the derivation module in our video classification architecture. We have tested our ideas by implementing the motion-tracing and extraction algorithm under Solaris 2.3 using the MPEG encoder produced by the Digital Video research team at the University of California, Berkeley. A functional view of the MPEG-based motion extraction is given in Figure 12. We have introduced functions for extraction of motion vectors during the generation of P- and B-frames. We use the motion-tracing algorithm to compute the macroblock trajectories.

The performance results are shown in Figure 13. We have tested our motion-tracing algorithm by ranging the number of macroblocks being traced from zero to all macroblocks. The input video sequence is the standard table tennis sequence, which consists of 10 frames, each of size 352 by 240 pixels. This sequence is a good performance test case, because it has background and foreground motion. The encoding frame pattern is IBBBPBBBBP. This means that all the input frames are used for video encoding. If only encoding is performed without any motion-tracing algorithm, the total elapsed time is 32.6 seconds (+/− 0.05 seconds). With the motion-tracing algorithm, the time increase is evident with the increase of the number of blocks. Starting

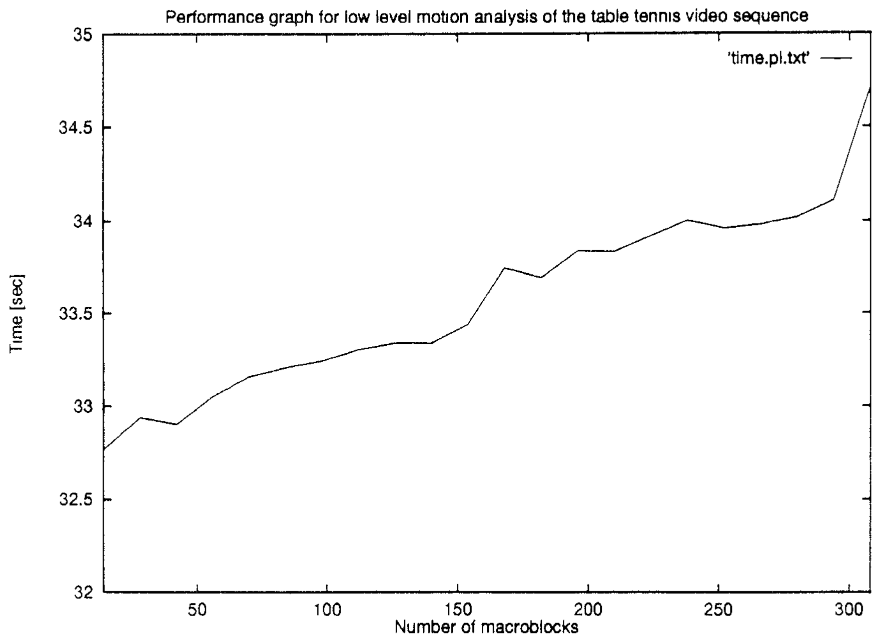


Fig. 13. Performance of object motion tracking.

with one macroblock, we get elapsed time of 32.76 seconds for encoding and tracing which is 0.16 seconds more than the previous case. As shown in Figure 13, when the number of traced macroblocks increases up to 300, the elapsed time goes up to 34.72 seconds. This shows that even if we keep track of the motion of all the macroblocks we have a time increase of 6%.

The gain in MPEG compression is mostly achieved by exploiting temporal redundancy. MPEG avoids coding the same block twice by storing/sending over the displacement vector from the previous image. Thus, the basic assumption is that the frame pattern used for MPEG compression is going to contain P- and B-frames.

Our algorithm for motion tracing would have very limited application if the stream to be encoded is using only I-frames. In that case, there the motion algorithm cannot find any motion vectors to take advantage of. If high quality of encoded video is crucial to the application at hand, then the algorithm has to be rewritten, so that motion estimation is performed using some imaginary frame pattern which would not have any impact on the encoded video stream. Then the motion-tracing algorithm would be performed on the obtained motion estimates. In this case, the motion information that is obtained from the encoder is in the forward vectors of the P-frames only. From the P-frame to the next I-frame we do not have any motion information. We have several choices:

(a) We can make a prediction for the motion vector between the P-frame and the next I-frame. This prediction is a guess that we can use the same motion vector as the vector for the P-frame. This solution does not introduce additional overhead. The problem is that it relies not only on the assumption for the continuity of motion but also assumes that the motion is constant.

(b) We can perform the actual search and compute the motion vector for the blocks from the P- to the next I-frame. This means that we will be adding much more compute cycles than it is necessary for the encoding process.

We also need much more complicated motion models to recover the true motion of the objects in the case of complicated camera motion. For example, when we have the camera focus on a moving object, then the object appears to be stationary. The motion of the object is implied by the macroblock vectors of the background. More-sophisticated relative-motion detection algorithms are needed. This work is part of our ongoing SunSet Multimedia Information System project [Golshani and Dimitrova 1994; Michael 1994].

## 6. CONCLUSIONS

From the point of view of video retrieval, the video technology has not seen much progress from the days when film editors examined each and every frame by hand in order to find the exact place of each cut. In fact, despite the introduction of many video editing systems such as VideoShop and Adobe's Premiere, much of retrieval is done by either time pointers (e.g., the frame counter), visual proxies, or various types of graphical or descriptive pointers. What is clearly missing from the video technology is the ability to locate and retrieve video clips that contain an object with specific characteristics, particularly with respect to movements. Video databases can be useful to many application areas such as education, business, medicine, and more prominently, entertainment. As such, the value of better and more-equipped video systems are becoming clearer. While many aspects of video systems, such as presentation editing tools, have seen significant improvement, our progress on content-based retrieval has not been as forthcoming.

We believe that our attempts to address the above needs must start with a modeling mechanism that allows for the representation of semantic knowledge from both spatial and temporal features of the objects in video sequences. Computing high-level motion description can be done independently of recognizing objects [Allmen 1991]. We elaborate on this property by showing that the recovery of object trajectories can be performed without prior knowledge of objects undergoing motion. The goal is to have both: independent retrieval along the temporal and the spatial hierarchies as well as retrieval of combined features from the spatial and the temporal hierarchies. We treat motion vectors extracted during the motion compensation phase of video encoding as coarse-level optical flow that is further used for intermediate- and high-level motion description. Motion information extraction is then carried out at low level by motion vector detection, at the intermediate level by motion tracing, and the high level by associating an object and a set of trajectories with recognizable activities.

In our object motion representations, we provide various levels of precision of trajectory representation. Retrieval functions based on these representations offer a wide spectrum of approximation in the process of matching. We need to relate the motion at a higher level of abstraction of the object to the detailed motion of parts of objects. Events can be represented in a form that is common in the image-understanding and interpretation area: predicates, temporal networks, etc.

ACKNOWLEDGMENTS

The authors gratefully acknowledge assistance from the guest editors. We would like to thank Tom Foley for clarifying our ideas on B-spline representation. We would like to thank Guru Banavar for the numerous discussions and insightful comments on the visual language. The authors would also like to thank K. P. Lee for suggesting Tcl/Tk as the implementation platform for the interface.

## REFERENCES

ALLMEN, M C. 1991. Image sequence description using spatiotemporal flow curves Toward motion-based recognition. Ph.D. thesis, Univ. of Wisconsin, Madison, Wisc

ARMAN, F., DEPOMMIER, R. HSU, A., AND CHIU, M.-Y. 1994. Content-based browsing of video sequences In *Proceedings of ACM Multimedia '94* (San Francisco, Calif). ACM Press, New York, 97–103.

BOBICK, A. F. 1993. Representational frames in video annotation. In *Proceedings of the 27th Annual Conference on Signals. Systems and Computers*. IEEE Computer Society Press, Los Alamitos, Calif.

BUCHANAN, M. C. AND ZELLWEGER, P. T. 1993. Automatically generating consistent schedules for multimedia documents. *Multimedia Syst. J. 1*, 2 (Sept.).

DAVIS, M. 1993. Media streams: An iconic visual language for video annotation. In *Proceedings of the IEEE Symposium on Visual Languages* (Bergen, Norway). IEEE, New York, 196–202.

DIMITROVA, N 1995. Content classification and retrieval of digital video based on motion recovery. Ph.D. thesis, Arizona State Univ., Tempe, Ariz.

DIMITROVA, N. AND GOLSHANI, F. 1994. Rx for semantic video database retrieval. In *Proceedings of ACM Multimedia '94* (San Francisco, Calif.). ACM Press, New York, 219–226.

DUDA, R. O. AND HART, P. E. 1973 *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York.

FARIN, G. 1990. *Curves and Surfaces for Computer Aided Geometric Design* Academic Press, New York.

FURHT, B. 1994. Multimedia systems: An overview. *IEEE Multimedia 1*, 1 (Spring), 47–59.

GODDARD, N. 1992. The perception of articulated motion: Recognizing moving light displays. Ph.D. thesis, Univ. of Rochester, Rochester, N.Y.

GOLSHANI, F. AND DIMITROVA, N. 1994. Retrieval and delivery of information in multimedia database systems. *Inf. Softw Tech. 36*, 4 (May), 235–242.

GROSKY, W. AND MEHROTRA, R 1989. Image database management *IEEE Comput. 22*, 12 (Dec.), 7–8.

GROSKY, W. I. 1994. Multimedia information systems. *IEEE Multimedia 1*, 1 (Spring), 12–24.

GUPTA, A., WEYMOUTH, T., AND JAIN, R. 1991a. Semantic queries in image databases. In *Visual Database Systems II*, E Knuth and L. Wegner, Eds. Elsevier Science Publishers (North-Holland), Amsterdam, 201–215.

GUPTA, A., WEYMOUTH, T., AND JAIN, R. 1991b. Semantic queries with pictures: The VIMSYS Model. In the *Conference on Very Large Data Bases*. VLDB Endowment, Saratoga, Calif.

HAMPAPUR, A., WEYMOUTH, T., AND JAIN, R. 1994. Digital video segmentation. In *Proceedıngs of ACM Multimedia '94* (San Francısco, Calif). ACM Press, New York, 357–364.

HORN, B. K. AND SCHUNCK, B. G. 1981. Determining optical flow. *Artıf. Intell. 17*, 1–3, 185–203.

JOHANSSON, G. 1976. Spatio-temporal dıfferentiatıon and ıntegration in visual motion perception. *Psychol. Res. 38*, 4, 379–393.

KOLLER, D., WEBER, J., AND MALIK, J. 1993. Robust multiple car tracking with oclusion reasoning. Tech. Rep. CSD-93-780, EECS Dept., Univ. of California, Berkeley, Calif. Nov.

KUBOTA, H., OKAMOTO, Y., MIZOGUCHI, H., AND KUNO, Y. 1993. Vision processor system for moving-object analysis. *Mach. Vis. Appl. 7*, 1, 37–43.

LEGALL, D. 1991. MPEG: A video compressıon standard for multimedia applicatıons. *Commun. ACM 34*, 4 (Apr.), 46–58.

LITTLE, T., AHANGER, G., FOLZ, R., GIBBON, J., REEVE, F., SCHELLENG, D., AND VENKATESH, D. 1993. A digital on-demand video service supporting content-based queries. In *Proceedıngs of ACM Multımedıa '93* (Anaheim, Calif.). ACM Press, New York, 427–436.

LITTLE, T., GHAFOOR, A., CHEN, C., CHANG, C., AND BERRA, P. 1991. Multimedia synchronization. *Data Eng. 14*, 3, 26–35.

MATTISON, P. E. 1994. *Practical Dıgıtal Video with Programmıng Examples ın C.* John Wiley and Sons, New York.

MICHAEL, N. 1994. VEENA—a visual query language. M.S. thesis, Arizona State Univ., Tempe, Ariz.

NAGASAKA, A. AND TANAKA, Y. 1992. Automatic video indexing and full-video search for object appearances. In *Visual Database Systems II*, E. Knuth and L. Wegner, Eds. Elsevier Science Publishers (North-Holland), Amsterdam, 113–127.

OSTERHOUT, J. K. 1994. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, Mass.

OTSUJI, K. AND TONOMURA, Y. 1993. Projection detection filter for video cut detection. In *Proceedıngs of ACM Multımedıa '93* (Anaheim, Calif.). ACM Press, New York, 251–257.

PATEL, K., SMITH, B. C., AND ROWE, L. A. 1993. Performance of a software MPEG video decoder. In *Proceedings of ACM Multımedia '93* (Anaheim, Calif.). ACM Press, New York, 75–82.

RASURE, J., ARGIRO, D., SAUER, T., AND WILLIAMS, C. 1990. Visual language and software development environment for image processing. *Int. J. Imaging Syst. Tech. 2*, 2, 183–199.

ROHR, K. 1994. Towards model-based recognition of human movements in image sequences. *CVGIP: Image Understandıng 59*, 1 (Jan.), 94–115.

ROWE, L. A. AND SMITH, B. C. 1992. A continuous media player. In the *3rd International Workshop on Network and OS Support for Digital Audio and Video.* Springer-Verlag, Berlin, 334–344.

ROWE, L. A., BORECZKY, J. S., AND EADS, C. A. 1994. Indexes for user access to large video databases. In *Proceedings of SPIE IS and Symposıum on Storage and Retrıeval for Image and Video Databases* (San Jose, Calif.). SPIE.

SCHALKOFF, R. J. 1989. *Digital Image Processing and Computer Visıon.* John Wiley and Sons, New York.

SWANBERG, D., SHU, C.-F., AND JAIN, R. 1993. Knowledge guided parsing in video databases. In *Image and Video Processıng Conference; Symposıum on Electronıc Imagıng: Scıence and Technology.* Vol. 1908. IS & T/SPIE, 13–24.

TEODOSIO, L. AND BENDER, W 1993. Sallient video stills: Content and context preserved. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York.

TEODOSIO, L. AND MILLS, M. 1993. Panoramic overviews for navigating real-world scenes. In *Proceedıngs of ACM Multimedıa '93* (Anaheim, Calif.). ACM Press, New York.

WEISS, R. 1994. Content-based access to algebraic video. Tech. Rep., Massachusetts Inst. of Technology, Cambridge, Mass.

ZHANG, H., GONG, Y., SMOLIAR, S., AND TAN, S. Y. 1994. Automatic parsing of news video. In *Proceedings of the Internatıonal Conference on Multimedia Computıng and Systems* (Boston, Mass.). IEEE Computer Society Press, Los Alamitos, Calif., 45–54.

Received August 1994; revised February 1995; accepted July 1995

October 1995
Volume 13 Number 4

[illegible]

acm Series on
Computing
Methodologies

# acm Transactions on Information Systems

## Special Issue on Video Information Retrieval

acm PRESS

Published by ACM, the First Society in Computing

# acm Transactions on Information Systems

ACM
1515 Broadway
New York, NY 10036
Tel: (212) 869-7440

ACM European Service Center
Avenue Marcel Thiry 204
1200 Brussels, Belgium
Tel: 32 2 774 9602
Fax: 32 2 774 9690
Email: acm_europe@acm.org

*ACM Transactions on Information Systems* (ISSN 1046-8188) is published 4 times a year in January, April, July, and October by the Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036. Second-class postage paid at New York, NY 10001, and at additional mailing offices. Postmaster: Send address changes to Transactions on Information Systems, ACM, 1515 Broadway, New York, NY 10036.

For subscription and submissions information, see inside back cover.

# Introduction to the Special Issue on Video Information Retrieval

This special issue of the *ACM Transactions on Information Systems* is focused on the topic of digital video in information systems. Multimedia in general and digital video in particular can deliver more information, more effectively, than any scheme developed to date. But more than just delivering information, effective digital video systems require a deep understanding of how users interact with huge volumes of information in many forms.

Early and most current multimedia applications incorporating digital video tend to be based on one of two models: real-time video for communications (desktop video conferencing) or the selection and playback of digital video clips (interrupted video). Because they treat the video segment as a black box, they are inadequate for access to extremely large digital video libraries or for creating truly interactive video applications.

Recent developments in consumer electronics and communications can provide a tighter integration of digital video and information system technologies. Computer manufacturers in cooperation with game companies are delivering 64-bit, 100MHz processors, for home machines, costing under $250. While most commercial "Information Superhighway" prototypes focus on video-on-demand, emerging technologies are enabling digital video, multimedia solutions to an ever-widening variety of commercial products and research projects. These advanced, digital video applications have the potential to transform how people work, learn, and play.

The articles included in this special issue address topics related to video indexing, analysis, content-based retrieval, delivery, and architecture. For example, in the article by **Tat-Seng Chua** and **Li-Qun Ruan** a novel system is described that is designed to support the entire process of video information management from segmenting and indexing of video to its retrieval and reuse. The article by **Nevenka Dimitrova** and **Forouzan Golshani** looks at extracting semantic information from video, in particular the application of motion analysis to retrieve temporal information associated with video. **Dick Bulterman's** article reports on control issues when applications manipulate video data. And finally, the article by **Ralf Keller, Wolfgang Effelsberg,**

# Introduction to the Special Issue on Video Information Retrieval

This special issue of the *ACM Transactions on Information Systems* is focused on the topic of digital video in information systems. Multimedia in general and digital video in particular can deliver more information, more effectively, than any scheme developed to date. But more than just delivering information, effective digital video systems require a deep understanding of how users interact with huge volumes of information in many forms.

Early and most current multimedia applications incorporating digital video tend to be based on one of two models: real-time video for communications (desktop video conferencing) or the selection and playback of digital video clips (interrupted video). Because they treat the video segment as a black box, they are inadequate for access to extremely large digital video libraries or for creating truly interactive video applications.

Recent developments in consumer electronics and communications can provide a tighter integration of digital video and information system technologies. Computer manufacturers in cooperation with game companies are delivering 64-bit, 100MHz processors, for home machines, costing under $250. While most commercial "Information Superhighway" prototypes focus on video-on-demand, emerging technologies are enabling digital video, multimedia solutions to an ever-widening variety of commercial products and research projects. These advanced, digital video applications have the potential to transform how people work, learn, and play.

The articles included in this special issue address topics related to video indexing, analysis, content-based retrieval, delivery, and architecture. For example, in the article by **Tat-Seng Chua** and **Li-Qun Ruan** a novel system is described that is designed to support the entire process of video information management from segmenting and indexing of video to its retrieval and reuse. The article by **Nevenka Dimitrova** and **Forouzan Golshani** looks at extracting semantic information from video, in particular the application of motion analysis to retrieve temporal information associated with video. **Dick Bulterman's** article reports on control issues when applications manipulate video data. And finally, the article by **Ralf Keller, Wolfgang Effelsberg,**

# Motion Recovery for Video Content Classification

NEVENKA DIMITROVA and FOROUZAN GOLSHANI
Arizona State University, Tempe

Like other types of digital information, video sequences must be classified based on the semantics of their contents. A more-precise and completer extraction of semantic information will result in a more-effective classification. The most-discernible difference between still images and moving pictures stems from movements and variations. Thus, to go from the realm of still-image repositories to video databases, we must be able to deal with motion. Particularly, we need the ability to classify objects appearing in a video sequence based on their characteristics and features such as shape or color, as well as their movements. By describing the movements that we derive from the process of motion analysis, we introduce a dual hierarchy consisting of spatial and temporal parts for video sequence representation. This gives us the flexibility to examine arbitrary sequences of frames at various levels of abstraction and to retrieve the associated temporal information (say, object trajectories) in addition to the spatial representation. Our algorithm for motion detection uses the motion compensation component of the MPEG video-encoding scheme and then computes trajectories for objects of interest. The specification of a language for retrieval of video based on the spatial as well as motion characteristics is presented.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*motion*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Content-based retrieval of video, motion recovery, MPEG compressed video analysis, video databases, video retrieval

## 1. INTRODUCTION

Applications such as video on demand, automated surveillance systems, video databases, industrial monitoring, video editing, road traffic monitoring, etc. involve storage and processing of video data. Many of these applications can benefit from retrieval of the video data based on their content. The problem is that, generally, any content retrieval model must have the capability of

This article is a revised version with major extensions of an earlier paper which was presented at the ACM Multimedia '94 Conference.

Authors' addresses: N. Dimitrova, Philips Laboratories, 345 Scarborough Road, Briarcliff Manor, NY 10562; email: nvd@philabs.philips.com; F. Golshani, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406; email: golshani@asu.edu.

dealing with massive amounts of data. As such, classification is an essential step for ensuring the effectiveness of these systems.

Motion is an essential feature of video sequences. By analyzing motion of objects we can extract information that is unique to the video sequences. In human and computer vision research there are theories about extracting motion information independently of recognizing objects. This gives us support for the idea of classifying sequences based on the motion information extracted from video sequences regardless of the level of recognition of the objects. For example, using the motion information we can not only submit queries like "retrieve all the video sequences in which there is a moving pedestrian and a car" but also queries that involve the exact position and trajectories of the car and the pedestrian.

Previous work in dynamic computer vision can be classified into two major categories based on the type of information recovered from an image sequence: recognition through recovering structure from motion and recognition through motion directly. The first approach may be characterized as attempting to recover either low-level structures or high-level structures. The low-level structure category is primarily concerned with recovering the structure of rigid objects, whereas the high-level structure category is concerned primarily with recovering nonrigid objects from motion. Recovering objects from motion is divided into two subcategories: low-level motion recognition and high-level motion recognition. Low-level motion recognition is concerned with making the changes between consecutive video frames explicit (this is called optical flow [Horn and Schunck 1981]). High-level motion recognition is concerned with recovering coordinated sequences of events from the lower-level motion descriptions.

Compression is an inevitable process when dealing with large multimedia objects. Digital video is compressed by exploiting the inherent redundancies that are common in motion pictures. Compared to encoding of still images, video compression can result in huge reductions in size. In the compression of still images, we take advantage of spatial redundancies caused by the similarity of adjacent pixels. To reduce this type of redundancy, some form of transform-based coding (e.g., Discrete Cosine Transform, known as DCT) is used. The objective is to transform the signal from one domain (in this case, spatial) to the frequency domain. DCT operates on $8 \times 8$ blocks of pixels and produces another block of $8 \times 8$ in the frequency domain whose coefficients are subsequently quantized and coded. The important point is that most of the coefficients are near zero and after quantization will be rounded off to zero. Run-length coding, which is an algorithm for recording the number of consecutive symbols with the same value, can efficiently compress such an object. The next step is coding. By using variable-length codes (an example is Huffman tables), smaller code words are assigned to objects occurring more frequently, thus further minimizing the size.

Our aim in the coding of video signals is to reduce the temporal redundancies. This is based on the fact that, within a sequence of related frames, except for the moving objects, the background remains unchanged. Thus to reduce temporal redundancy a process known as motion compensation is used. Motion compensation is based on both predictive and interpolative coding.

MPEG (Moving Pictures Expert Group) is the most general of the numerous techniques for video compression [Furht 1994; LeGall 1991; Mattison 1994]. In fact, the phrase "video in a rainbow" is used for MPEG, implying that by adjusting the parameters, one can get a close approximation of any other proposal for video encoding. Motion compensation in MPEG consists of predicting the position of each 16 × 16 block of pixels (called a macroblock) through a sequence of predicted and interpolated frames. Thus we work with three types of frames—namely, those that are fully coded independently of others (called reference frames or I-frames), those that are constructed by prediction (called predicted frames or P-frames), and those that are constructed by bidirectional interpolation (known as B-frames). It begins by selecting a frame pattern which dictates the frequency of I-frames and the intermixing of other frames. For example, the frame pattern IBBPBBI indicates (1) that every seventh frame is an I-frame, (2) that there is one predicted frame in the sequence, and (3) that there are two B-frames between each pair of reference and/or predicted frames. Figure 1 illustrates this pattern.

Our approach to extracting object motion is based on the idea that during video encoding by the MPEG method, a great deal of information is extracted from the motion vectors. Part of the low-level motion analysis is already performed by the video encoder. The encoder extracts the motion vectors for the encoding of the blocks in the predicted and bidirectional frames. A macroblock can be viewed as a coarse-grained representation of the optical flow. The difference is that the optical flow represents the displacement of individual pixels while the macroblock flow represents the displacement of macroblocks between two frames. At the next, intermediate level, we extract macroblock trajectories which are spatiotemporal representations of macroblock motion. These macroblock trajectories are further used for object motion recovery. At the highest level, we associate the event descriptions to object/motion representations.

Macroblock displacement in each individual frame is described by the motion vectors which form a coarse optical-flow field. We assume that our tracing algorithm is fixed on a moving set of macroblocks and that the correspondence problem is elevated to the level of macroblocks instead of individual points. The advantage of this elevation is that even if we lose individual points (due to turning, occlusion, etc.) we are still able to trace the object through the displacement of a macroblock. In other words, the correspondence problem is much easier to solve and less ambiguous. Occlusion and tracing of objects which are continuously changing are the subject of our current investigations.

In Section 2 of this article we survey some of the research projects related to our work. In Section 3 we present the object motion analysis starting from the low-level analysis through the high-level analysis. We discuss the importance of motion analysis and its relevance to our model which is presented in Section 3.4. Section 4 introduces the basic OMV structures (object, motion,

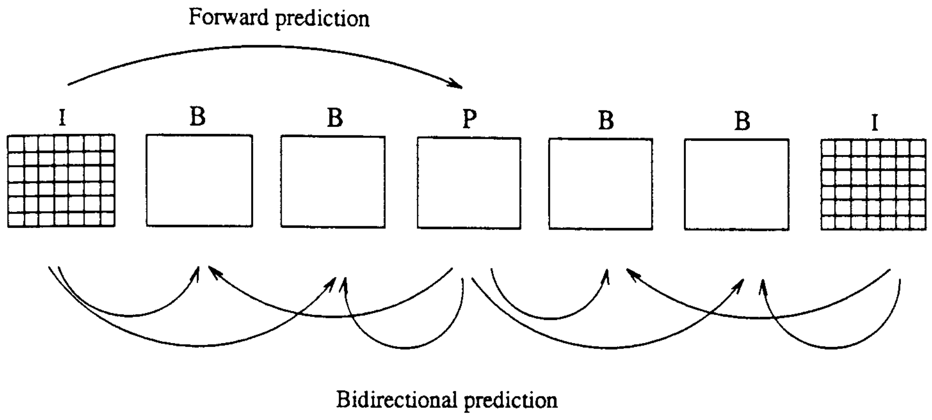


Fig. 1. Forward and bidirectional prediction in MPEG.

video-sequence), as the basis for the video information model. The basic retrieval operators, the OMV-language specification, and some examples are given. Empirical results are outlined in Section 5, and Section 6 presents some concluding remarks.

## 2. RELATED WORK

The research presented in this article builds on the existing results in two areas: dynamic computer vision and digital video modeling.

A current trend in computational vision is influenced by the idea that motion analysis does not depend on complex-object descriptions. Our work follows this trend and is based on the recent publications that are in agreement with this idea in computational vision. The idea of object/event recognition regardless of the existence of object representations can be traced back to the early 70's when Johansson [1976] introduced his experiments with *moving-light displays*. The idea was to attach lights to the joints of a human subject dressed in dark-colored clothing and observe the motion of lights against a dark background. The audience not only could recognize the object (human being) but could also describe the motion and the events taking place. Goddard [1992] investigated the high-level representations and computational processes required for the recognition of human motion based on moving-light displays. The idea is that recognition of any motion involves indexing into stored models of the movement. These stored models, called scenarios, are represented based on coordinated sequences of discrete motion events. The structures and the algorithms are articulated in the language of structured connectionist models. Allmen [1991] introduced a computational framework for intermediate-level and high-level motion analysis based on spatiotemporal surface flow and spatiotemporal flow curves. Spatiotemporal surfaces are projections of contours over time. Thus, these surfaces are direct representations of object motion.

In the dynamic computer vision literature there are general models for object motion estimation and representation, as well as domain-restricted models. A general architecture for the analysis of moving objects is proposed by Kubota et al. [1993]. The process of motion analysis is divided into three stages: moving-object candidate detection, object tracking, and final motion analysis. The experiments are conducted using human motion. Another approach to interpretation of the movements of articulated bodies in image sequences is presented by Rohr [1994]. The human body is represented by a three-dimensional model consisting of cylinders. This approach uses the modeling of the movement from medical motion studies. Koller et al. [1993] discuss an approach to tracking vehicles in road traffic scenes. The motion of the vehicle contour is described using an affine motion model with a translation and a change in scale. A vehicle contour is represented by closed cubic splines. We make use of the research results in all these domain-specific motion analysis projects. Our model combines the general area of motion analysis with individual frame (image) analysis.

In case of video modeling, the video footage usually is first segmented into shots. Segmentation is an important step for detection of cut points which can be used for further analysis. Each video shot can be represented by one or more key frames. Features such as color, shape, and texture could be extracted from the key frames. An approach for automatic video indexing and full video search is introduced by Nagasaka and Tanaka [1992]. This video-indexing method relies on automatic cut detection and selection of first frames within a shot for content representation. Otsuji and Tonomura [1993] propose a video cut detection method. Their projection detection filter is based on finding the biggest difference in consecutive-frame histogram differences over a period of time. A model-driven approach to digital video segmentation is proposed by Hampapur et al. [1994]. The paper deals with extracting features that correspond to cuts, spatial edits, and chromatic edits. The authors present an extensive formal treatment of shot boundary identification based on models of video edit effects. In our work, we rely on these methods for the initial stages of video processing, since we need to identify shot boundaries to be able to extract meaningful information within a shot.

One representation scheme of segmented video footage uses key frames [Arman et al. 1994]. The video segments can also be processed for extraction of synthetic images, or layered representational images, to represent closely the meaning of the segments. A methodology for extracting a representative image, *salient video stills*, from a sequence of images is introduced by Teodosio and Bender [1993]. The method involves determining the optical flow between successive frames, applying affine transformations calculated from the flow-warping transforms, such as rotation, translation, etc., and applying a weighted median filter to the high-resolution image data resulting in the final image. A similar method for synthesizing panoramic overviews from a sequence of frames is implemented by Teodosio and Mills [1993].

Swanberg et al. [1993] introduced a method for identifying desired objects, shots, and episodes prior to insertion in video databases. During the insertion process, the data are first analyzed with image-processing routines to identify the key features of the data. In this model, episodes are represented using finite automata. Only video clips with inherently well defined structure can be represented. The model exploits the spatial structure of the video data without analyzing object motion. Zhang et al. [1994] presented an evaluation and a study of knowledge-guided parsing algorithms. The method has been implemented for parsing of television news, since video content parsing is possible when one has an a priori model of a video's structure.

Another system, implemented by Little et al. [1993], supports content-based retrieval and playback. They define a specific schema composed of movie, scene, and actor relations with a fixed set of attributes. Their system requires manual feature extraction. It then fits these features into the schema. Querying involves the attributes of movie, scene, and actor. Once a movie is selected, a user can browse from scene to scene beginning with the initial selection. Weiss [1994] presented an algebraic approach to content-based access to video. Video presentations are composed of video segments using a *video algebra*. The algebra contains methods for temporally and spatially combining video segments, as well as methods for navigation and querying. Media Streams is a visual language that enables users to create multilayered iconic annotations of video content [Davis 1993]. The objects denoted by icons are organized into hierarchies. The icons are used to annotate the video streams in a *Media Time Line*. The Media Time Line is the core browser and viewer of Media Streams. It enables users to visualize video at multiple time scales simultaneously, in order to read and write multilayered, iconic annotations, and it provides one consistent interface for annotation, browsing, query, and editing of video and audio data.

The work presented here follows from a number of efforts listed above. Specifically, we use low- and intermediate-level motion analysis methods similar to those offered by Allmen [1991] and others. Our object recognition ideas have been influenced by the work of Jain and his students [Gupta et al. 1991a; 1991b], Grosky [Grosky and Mehrotra 1989], and the research in image databases. Several lines of research such as those in Little et al. [1993], Swanberg et al. [1993], Zhang et al. [1994], and Weiss [1994] provided many useful ideas for the modeling aspects of our investigations. An early report of our work was presented in Dimitrova and Golshani [1994].

## 3. MOTION RECOVERY IN DIGITAL VIDEO

In this section we describe in detail each level of the motion analysis pipeline. At the low-level motion analysis we start with a domain of motion vectors. During intermediate-level motion analysis we extract motion trajectories that are made of motion vectors. Each trajectory can be thought of as an n-tuple of motion vectors. This trajectory representation is a basis for various other trajectory representations. At the high-level motion analysis we associate an activity to a set of trajectories of an object using domain knowledge rules.

### 3.1 Low-Level Motion Extraction: Single Macroblock Tracing

In MPEG, to encode a macroblock in a predicted or a bidirectional frame, we first need to find the best matching macroblock in the reference frames, then find the amount of $x$ and $y$ translation (i.e., the motion vector), and finally calculate the error component [Patel et al. 1993]. The motion vector is obtained by minimizing a cost function that measures the mismatch between a block and each predictor candidate. Each bidirectional and predicted frame is an abundant source of motion information. In fact, each of these frames might be considered a crude interpolation of the optical flow. Thus, the extraction of the motion vectors of a single macroblock through a sequence of frames is similar to low-level motion analysis.

Tracing a macroblock can continue until the end of the video sequence if we do not impose a stopping criterion. We have a choice: to stop after a certain number of frames, stop after the object (macroblock) has come to rest, stop if the block comes to a certain position in the frame, stop if the macroblock gets out of the scene, or stop if the macroblock is occluded.

The algorithm for tracing the motion of a single macroblock through one frame pattern for MPEG encoding is given in Figure 2. In Dimitrova [1995], we describe object motion tracing for video databases in more detail. The algorithm takes the forward and backward motion vectors that belong to a particular macroblock and computes the macroblock's trajectory. The algorithm computes the macroblock's position in a B-frame by averaging the positions obtained from: (1) the previous block coordinates and forward motion vectors and (2) next (predicted) block coordinates and the backward motion vector. The position of a macroblock in a P-frame is computed using only block coordinates and forward motion vectors. If during the tracing procedure the initial macroblock moves completes out of its position, then we have to extract motion vectors for the new macroblock position, which implies that we are continuing by tracing the macroblock whose position coincides with the $(x, y)$ coordinates of the initial macroblock. In the rest of this article, we will use $\tau$ to indicate the set of all possible motion vectors.

3.1.1 *Trajectory Description.* Various motion retrieval procedures have specific requirements for retrieving desired objects. These requirements depend on the characteristics of the retrieval which may be flexible to strict. The choice of trajectory representation may dictate the manner in which retrieval is conducted. Given a set of motion vectors for a macroblock, a number of mechanisms exist for trajectory representation. Below we present a sample list:

(1) *Point Representation:* A trajectory in this case is a set of points represented by the absolute or relative frame coordinates of the position of the object, say

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where $(x_i, y_i)$ is derived by projecting $(x, y, i)$ onto the image plane. $(x, y, i)$ denotes the position of an object, i.e., $(x, y)$, at time instant $i$.

(2) *Curve Representation:* A parametric B-spline curve $P(u)$ can be computed that passes through each of the trajectory points $(x_i, y_i)$ (see Farin [1990] for a detailed discussion). The first step involves generating a parameteri-

```
Given: frames F =  Fi i = 0,...,n;
motion vectors V = (fmx(i),fmy(i)),(bmx(i),bmy(i)) i= 1,n
initial block coordinates bx, by
 Initialize $R = \emptyset$,
  for i=1,..., n
    if F(i) $\neq$ I then
      if F(i) == P then
         if previousType == I
            cx = bx - fmx(i)/2;
            cy = by - fmy(i)/2;
            nextblockx = cx; nextblocky = cy;
         if previousType == P
            givenx = futurex;
            giveny = futurey;
            futurex = futurex - fmx(i)/2;
            futurey = futurey - fmy(i)/2;
      if F(i) == B then
         cx=((givenx-fmx(i)/2)+(futurex-bmx(i)/2))/2;
         cy=((giveny-fmy(i)/2)+(futurey-bmy(i)/2))/2;
      if block(bx,by) $\cap$ block(cx,cy) == $\emptyset$ then
         extract(mx(i),my(i)) for (cx,cy)
      R = R $\cup$ {(mx(i),my(i))}
         if F(i) is the last in a group of B frames before a P frame
            cx = futurex;
            cy = futurey;
      if block(bx,by) $\cap$ block(cx,cy) == $\emptyset$ then
         extract(mx(i),my(i)) for (cx,cy)
      R = R $\cup$ {(mx(i),my(i))}
    if F(i) == I then
       (bx,by) $\leftarrow$ bestMatch(bx,by) in I
    if stopping criteria == true, then
          return R;
 endfor
```

Fig. 2. Algorithm for tracing the motion of a macroblock.

zation or *knot sequence* $u_1 \leq u_2 \leq \cdots \leq u_n$. A commonly used approach employs cumulative chord lengths defined by the points $(x_i, y_i)$. The next step involves setting up and solving a tridiagonal linear system of equations whose unknowns are the control points $d_i$ of the B-splines $N_i(u)$. The linear system depends on the $x_i$, $y_i$, and $u_i$ values. This linear system can be efficiently solved in $\mathcal{O}(n)$ time using standard techniques for tridiagonal matrices. The B-spline curve has the form:

$$P(u) = \sum d_i N_i(u),$$

and it satisfies the following:

(a) $P(u_i) = (x_i, y_i)$;

(b) $P(u)$ is a piecewise cubic polynomial, i.e., for $u_i \leq u \leq u_{i+1}$, $P(u)$ is a polynomial of degree less or equal to three; and
(c) the first and the second derivatives of $P(u)$ are continuous.

(3) *Chain Code Representation:* We develop a piecewise linear approximation to the trajectory using a set of orientation primitives. Given a set of discrete trajectory orientation primitives, we use a zig-zag line representation of the trajectory to generate the code. Another way of viewing this approach is derived from a neighborhood matrix with each neighbor coded to correspond to the primitives in the figure [Schalkoff 1989].
(4) *Differential Chain Code Representation:* Each segment is coded relative to the next line segment using the direction (left or right) and the length. For example, we can have a code for: right shorter–1, right equal–2, right longer–3, left shorter–4, left equal–5, left longer–6 [Schalkoff 1989]. This scheme is useful for approximate matching of object trajectories. It is a rotation-, scaling-, and translation-invariant scheme.

Figure 3 illustrates these methods used for the representation of an arbitrary movement. Figure 3(a) is an exact coordinate representation; 3(b) is a B-spline curve representation. Figure 3(c) represents the chain-coding process, and 3(d) shows the differential chain code representation of the trajectory.

Note that in the coordinate representation and B-spline and chain code representation schemes we have a way of representing zero motion, i.e., when the motion vector is a null vector. If the macroblock does not move over a certain number of frames, the point will be repeated. In the B-spline representation, the knot (i.e., the control point) will have a multiplicity greater than one. In the chain code representation, the zero motion is represented by the code "0." So, in all these representations the trajectory is not only a spatial representation of the object's motion (the path) but also a temporal characterization of the motion. By keeping track of the zero motion we are able to describe stationary objects as well.

The diversity of the trajectory representations makes the querying process more flexible. The actual method of representation does not have a significant impact on the querying process as long as modeling, representation, and querying are all done in the same fashion.

3.1.2 *Trajectory-Matching Functions.* Applications such as automated surveillance may require retrieval of either video sequences or objects contained in these sequences based on the object trajectories. For example, queries of the type "retrieve objects that have a motion trajectory whose point of origination is the main gallery door and terminate at the Juan Miro's picture on the opposite wall" may help in the identification of the person who damaged the picture.

Matching functions used for motion retrieval depend on the method employed for trajectory representation, as described below.

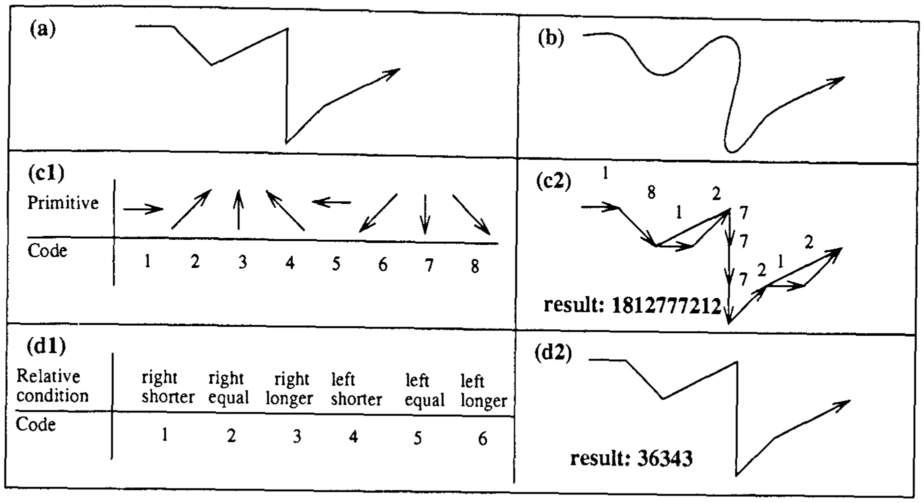


Fig. 3. Alternatives for object motion representation: (a) motion trajectory; (b) B-spline curve representation; (c1) chain-coding scheme; (c2) chain code representing the trajectory; (d1) differential chain-coding scheme; (d2) resulting differential chain code.

— Exact matching function that uses absolute frame coordinates (least-square minimization problem). This matching function has two variations:

(1) exact start position and exact trajectory match
(2) any start position and exact trajectory match.

— Exact matching function that uses relative coordinates. This function is used when the initial position of the object is not important.

— Curve comparison based on the curve-fitting approach used for interpolated trajectory representation.

— Approximate matching that uses chain code:

(1) exact start position and inexact trajectory match
(2) any start position and inexact trajectory match.

The chain code matching translates the problem of trajectory matching into a pattern-matching problem.

— Qualitative matching that uses differential chain code.

The result in each case is a similarity factor between the input trajectory and a target trajectory in the set of object trajectories.

## 3.2 Intermediate-Level Motion Analysis

A macroblock trajectory is the spatiotemporal representation of the macroblock's motion. These trajectories are further used for extracting object motion. This process is different for rigid and nonrigid bodies. A rigid object consists of one solid part to which motion trajectory is associated. If the object consists of several parts which themselves represent rigid objects with independent movements, then, such a nonrigid object is represented as a set of rigid objects with their respective trajectories. At the highest level of motion analysis, we associate "activities" with the object trajectory representations.

Rigid-object motion is represented by a single trajectory. The trajectory is one common representation of the trajectories of all the component macroblocks. Finding the most-representative trajectory is not a simple task. In the simplest case we can take the trajectory of the object centroid as the reference object trajectory. A more-complicated case occurs if we decide to create a common trajectory by processing all of the macroblock trajectories or by examining only a subset of all macroblock trajectories.

Mean averaging of all trajectories of the macroblocks of the object is an alternative to choosing the object centroid's trajectory. The averaging of the trajectories in the exact form is pointwise averaging of the trajectories at each frame.

The following two assumptions make the object motion recovery feasible:

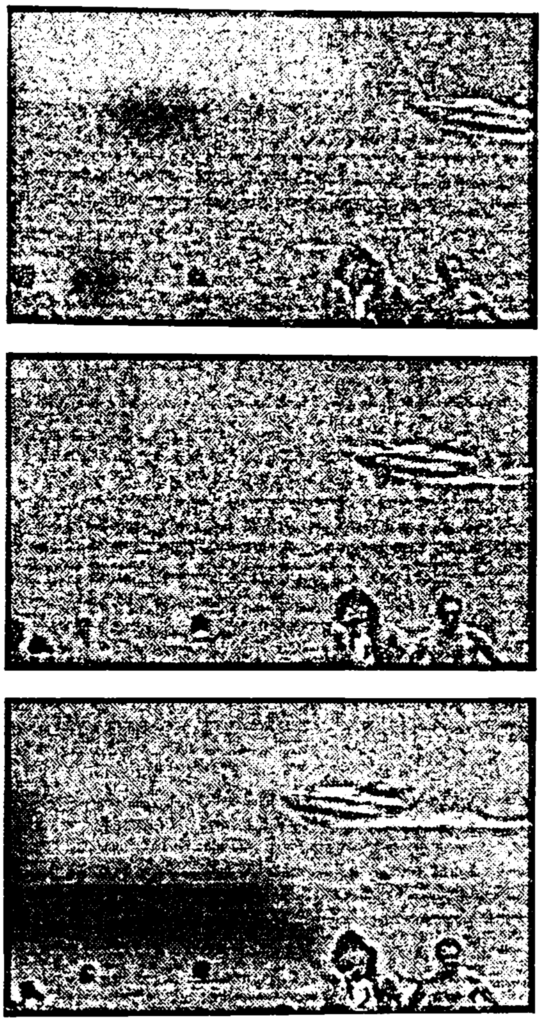
(1) *Integrity of Objects:* We assume objects are rigid or consist of rigid parts connected to each other. We do not consider situations in which objects disintegrate. This assumption is important because we only use object trajectory representation.
(2) *Motion Continuity:* Each macroblock under consideration has continuous motion. This assumption is important for the trajectory representation, since every trajectory segment represents continuation of the previous trajectory segment.

Averaging trajectories is used for determining a representation of a nonrigid body motion. For nonrigid objects, we must determine the number of trajectory clusters and their locations. Each cluster corresponds to a single coherent motion that represents a moving part (i.e., a rigid object). We use a hierarchical clustering algorithm (due to Duda and Hart [1973]) for determining the number of rigid object parts. Initially, the algorithm begins with clusters that contain only one trajectory each. At each subsequent step, we attempt to merge those neighboring clusters that have a similar trajectory. Individual trajectories, in this case, will be averaged to compute a trajectory for the extended cluster.

An example of a traced object through 20 encoded frames using the IBBPBBBP frame pattern is given in Figure 4. Figure 4(a) contains first, middle, and last frames of a video sequence capturing a water skiing scene. Figure 4(b) contains the motion trace for the moving yacht. The axes in Figure 4(b) correspond to the $x$ and $y$ axes of the video frames where the (0, 0) coordinate is at the top left corner.

Figure 5 shows six out of the 60 frames of the "Walfky" video sequence used for our next experiment. The object being traced is a small toy which performs very uneven motion. Figure 6(a) shows how the tracing of a macroblock progresses when every frame in the sequence is used. The frame pattern IBBBPBBB is used for video encoding when macroblock trajectories are extracted in Figure 6. This experiment shows that the macroblock tracing

(a)



(b)

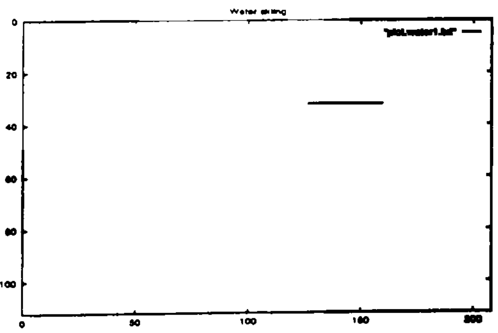


Fig. 4. Tracing a moving yacht. (a) first, middle, and last frames in video sequences; (b) motion trace of the yacht.

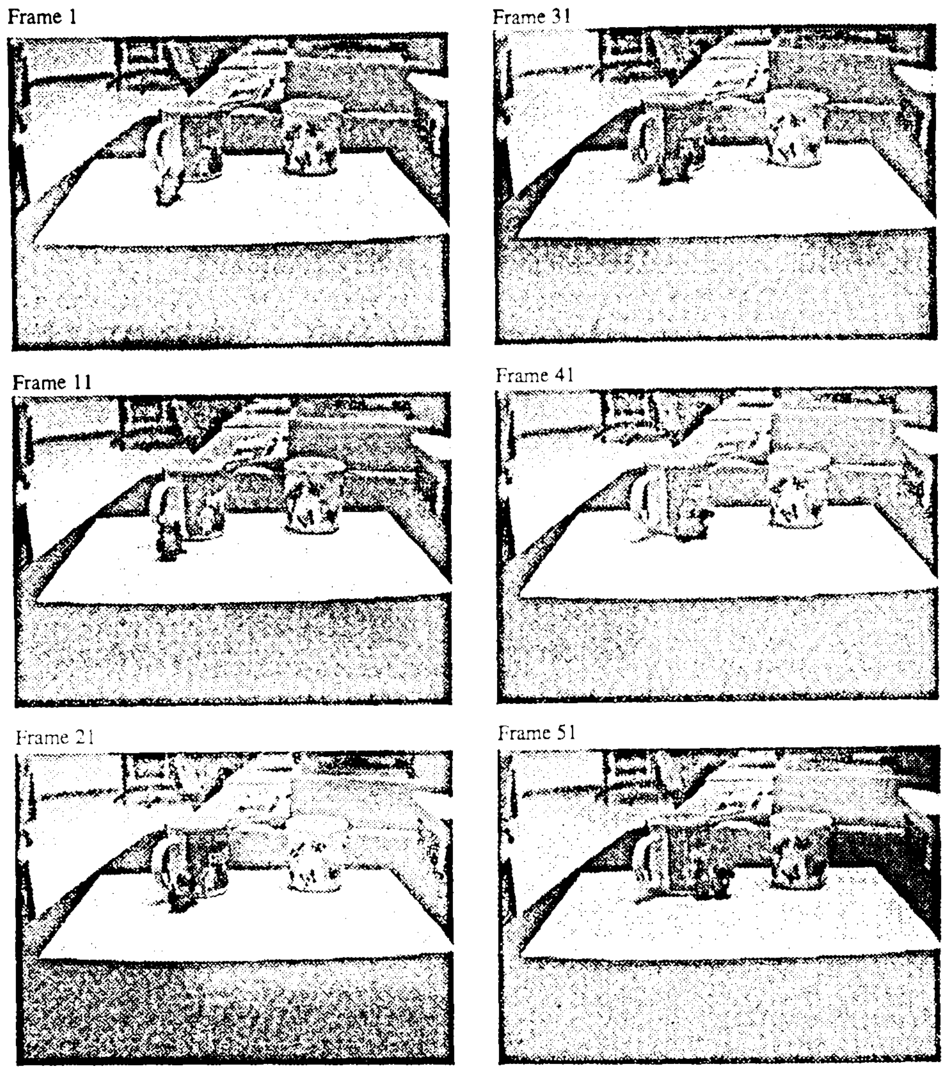


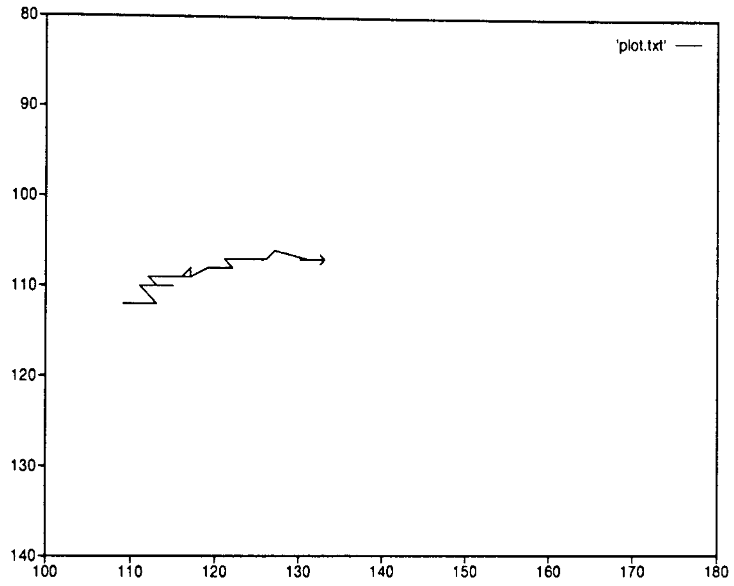
Fig. 5. Snapshots from a video sequence.

is possible when the objects exhibit jerky motion. As expected the trajectory is not only curved but also has the properties of a zig-zag line. In this case the macroblock with coordinates (14, 14) is traced. In terms of absolute frame coordinates, these coordinates correspond to (112, 112). Figure 6(b) shows tracing of the same video sequence in the case when every other frame is used for encoding and tracing.

We use the notation $T$ to indicate the set of object trajectories. Each member of $T$ is a sequence[1] whose range is the set of all motion vectors $\tau$, i.e., $\forall t \in T(t : \mathcal{N} \rightarrow \tau)$, where $\mathcal{N}$ is the set of natural numbers. In other words

[1] A sequence is simply a function whose domain is the natural numbers.

(a) Trace of the macroblock (14,14) using all frames



(b) Trace of the macroblock (14,14) using every other frame

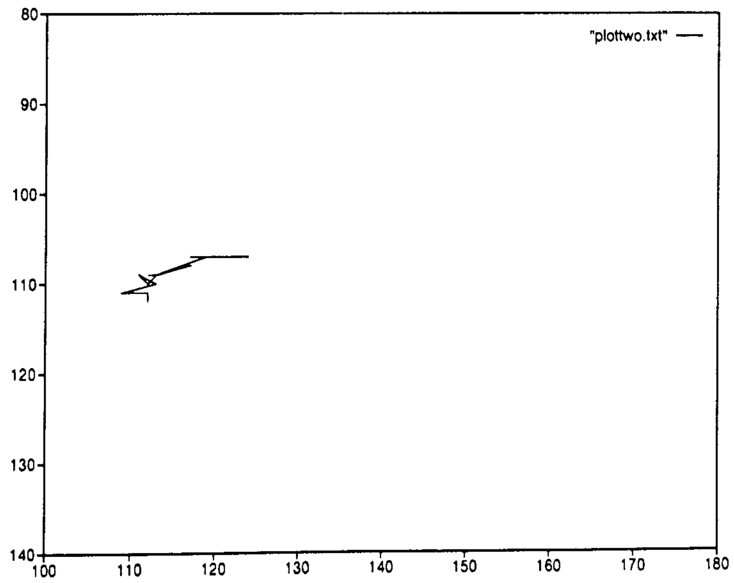


Fig. 6. Traced trajectories in the Walfky video sequence. (a) all frames; (b) only every other frame is used.

each object trajectory is a sequence of motion vectors identifying macroblock displacement for the components of the object. As discussed previously, the actual appearance of the members of $T$ depends on the choice of the representation scheme.

## 3.3 High-Level Motion Analysis

At the highest level of motion analysis, we associate domain-dependent "activities" with the object trajectory representations. An activity can be recognized by the system based on a predefined set of procedures, or it can be designated by the user. We realize that recognizing activities is one of the most-difficult tasks in any vision system. Such undertaking requires information on:

(1) Relative positioning between rigid subparts
(2) Relative timing of the parts movements
(3) Actual and perceived interaction of object parts.

The two main problems in recovering high-level motion representation are (1) the fact that multiple sequences are occurring simultaneously (for example, arm movements and leg movements in human motion) and in a coordinated fashion and (2) tempo changes are global (in the case of the human body, the changes apply to all four limbs and occur slowly).

An activity involves both spatial and temporal representations of the objects of interest. We must identify the object components (shape and other features) and their respective trajectories (as we did in the previous section) at the intermediate-level motion analysis and then assemble activities. The temporal information is needed for discrimination of activities of the same type, for example, strolling, walking, hurrying, etc. After assembling object activities, based on additional knowledge, we can infer event information.

We use $\mathscr{A}$ to symbolize the set of activities. We assume the existence of a knowledge base $\mathscr{K}$ whose contents include all the necessary rules, constraints, and the procedures for deriving activities from lower-level descriptions.

Each member $a$ of $\mathscr{A}$ is a "composition" of $t_1, t_2, \ldots, t_n$, where for every $1 < i < n$ we have:

— $t_i \in T$ and
— $t_i$ satisfies every constraint in $\mathscr{C}_a$, where $\mathscr{C}_a \in \mathscr{K}$ represents the constraints governing the activity $a$.

High-level event representation and manipulation call for the use of either temporal Petri nets, an event-based approach to temporal objects, or other event representation and manipulation schemes.

## 3.4 Spatiotemporal Hierarchical Representation

We use a semantic multiresolution hierarchy for spatiotemporal representation (Figure 7) because it helps video analysis at various resolution levels, with coarser resolutions used for high-level event/scenario descriptions. The

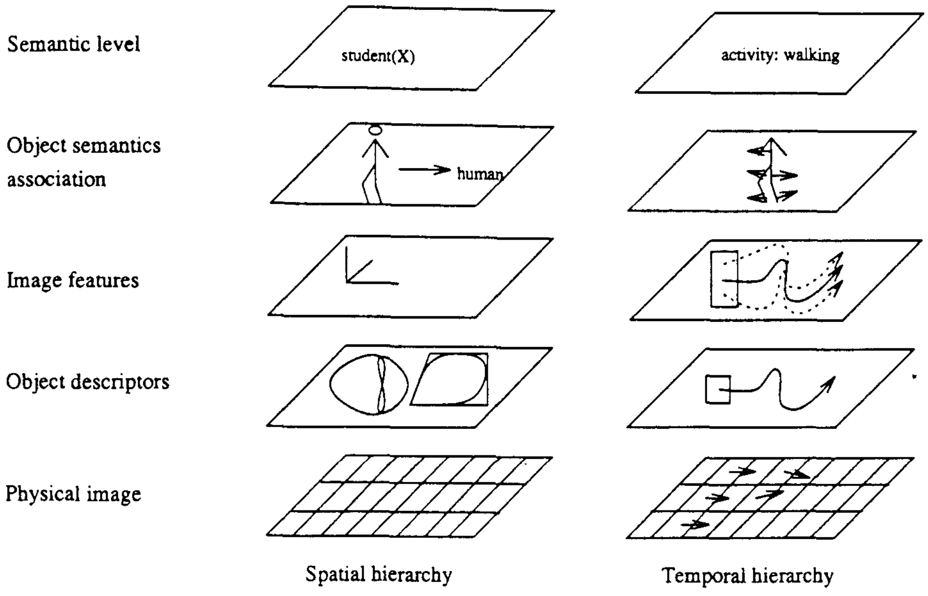


Fig. 7. Multiresolution hierarchy for spatial and temporal video representation.

advantage of multiresolution representation is that it offers a mechanism to make the trade-off between the competing demands of fine spatial/temporal resolution and low computational complexity. The idea of representational hierarchy for still images has been utilized by several image data models [Grosky and Mehrotra 1989; Gupta et al. 1991a; 1991b].

At successive time intervals, a frame is inserted at the base of the spatial hierarchy, and the features are computed for the next levels. The motion features are computed starting at the frame, and the temporal part of the hierarchy is filled with the appropriate motion descriptions. Motion analysis starts with the motion vector recovery (bottom of the temporal hierarchy, Figure 7). At the next level, individual macroblock trajectories are traced. At the intermediate level, rigid-body motion is recovered, followed by nonrigid-motion recovery. Finally, at the highest level of motion analysis, description of activities is derived from previously computed motion features (top of the temporal hierarchy, Figure 7).

The temporal part of the hierarchy can be used for various kinds of motion retrieval ranging from full-object trajectory-based matching to single-macroblock trajectory matching. Since we provide exact and inexact trajectory representation, our retrieval functions can take inputs in terms of precise spatial coordinates, orientation coordinates, and qualitative descriptions.

## 4. INFORMATION FILTERING AND DIGITAL VIDEO

The motion-tracing and representation scheme introduced in previous sections serves as a basis for the classification and retrieval of video sequences. Video sequences may be retrieved using the temporal (motion) part of the hierarchy or the combination of spatial and the temporal representations. The idea of this representation is that we can compute the spatial and temporal features independently of each other. We emphasize that temporal features coupled with spatial features are important in discriminating and classifying video sequences.

Like other knowledge representation cases, we do not attempt to have a universal system that can recognize and distinguish all possible objects. Such general-purpose (i.e., domain independent) representations have been shown to be too complex for present technologies. Thus, we assume that the domain of interest is known a priori and that the video classification system will be confined to working on only those objects. Consider a domain D, called the "scope," containing all objects of interest. Formally, the elements of D are defined as object-oriented structures with potentially complex internal components. Similar to any object-oriented representation, the user can identify the objects of D by their attributes, such as object ID, image descriptions, name, and shape (or convex hull), or a combination of these. Thus, the user may provide any available information on any of the attributes of desired object (for example, object ID, or shape together with a partial description), and the system will attempt to identify the intended object. Although we do not make any assumptions on how the elements of D and their attributes are represented, we offer the following example as an indication of a typical structure.

*Example* 4.1. A walking human may be represented as a moving object $a_1 = (o_1, m_1, v_1)$ where

$$\begin{aligned} o_1 &= (category : human, convexHull : o_6 : skeleton : o_7,\\ &\qquad parts : \{head : o_2, torso : o_3\}),\\ m_1 &= (trajectory : 2467332, activity : walking), \text{ and}\\ v_1 &= (v\# : 234, firstFrame : 45, lastFrame : 485). \end{aligned}$$

Similarly, the head and torso also have their spatial and motion descriptions.

In a database containing only "still" images, a correspondence table of the form (O, I)—where O stands for the object, and I stands for the image—will suffice. In a video database, we have the added parameter of temporal changes. Although the motion of each object can be modeled as an attribute of the object, say, "dog, big, brown, running," it is more appropriate to separate objects and their motions as two different parameters. Note that if motion is considered as just another attribute of the object, then in case the same object appears more than once, each time with a different motion, we would need multiple, different entries into the database. For example, there would be multiple entries for the big brown dog: running right, running left, running in circles, and jumping.

Video sequences are identified by objects present in the scene and their respective motion. The goal of the motion analysis is to extract activity and event representation. An index entry of an activity in a video sequence has the following form:

$$(O, M, V) = (objectRep, motionRep, vid)$$

— *objectRep:* an object represented by its extracted features (convex hull, object skeleton, centroid, texture, set of macroblocks covering the object, etc.; see left side of Figure 7). An object representation might include a set of object representations of the constituting parts (e.g., objects that represent a human figure include head, torso, arms, and legs object representations).

— *motionRep:* an object trajectory specified by objectRep, velocity, trajectory curvature, torsion, and activity description (see right side of Figure 7).

— *vid:* identity of the video subsequence to which an object belongs to: *vid* consists of (*viSeqId*, *firstFrame*, *lastFrame*).

(1) *viSeqId* is a video sequence identity which is unique for a sequence across the whole video database.
(2) *firstFrame:* the first frame in which the specified object appears.
(3) *lastFrame:* the last frame in which the specified object appears.

## 4.1 Content-Filtering Operators

The OMV triplet is the basis for the query functions. There are many possibilities for the selection of filters (in this context, query functions.) A sample selection is presented below. These operators may be used in a relational form, mostly in a table lookup mode, or may be embedded into a more-elaborate query language, as presented in Section 4.2. Recall that $\mathscr{P}$(A) is used to denote the powerset of the set A, i.e., the set of all subsets of A.

$$V\_Seq : O \times M \rightarrow \mathscr{P}(V)$$

This function takes any description that can be provided at any level of the spatial hierarchy. The input might be a characterization of the object in terms of its bounding polygon, stick figure, a name, or concept. At this point we need to emphasize that we use the properties of the object-oriented nature of the representation of the objects. For example, the expression

$$V\_Seq(O\_category = pet, (Activity = walking, Trajectory = t_1))$$

translates into "retrieve all the video sequences in which a pet walks and makes a trajectory $t_1$." The answer will include all the objects (animals) that are classified as pets: cats, dogs, fish called Wanda, etc. It is important to note that here we discuss only the formal framework in an informal way and that these functions are implemented within an interactive window-based graphical query interface which we discuss in Section 5.1.

The function

$$Object\_motion : O \times V \rightarrow \mathscr{P}(M)$$

takes any object description and a particular video sequence and returns a set of motion descriptions related to that object. In order to detect which objects performed a particular type of motion, i.e., the agent of the action, we use a function of the following family:

$$Agents : M \times V \rightarrow \mathscr{P}(O).$$

The next function is used to get a detailed description of all the objects and their respective motions in a video sequence:

$$Describe\_Video : V \rightarrow \mathscr{P}(O \times M).$$

If we just want information about the spatial characteristics of objects in a sequence, we use the function

$$Objects : V \rightarrow \mathscr{P}(O).$$

This is equivalent to the "agents" function where the first argument is unimportant. Thus, given a $v_i \in V$, $Objects(v_i) = Agents(any, v_i)$.

The above functions allow for inexactness, and by default, they return results that are approximately similar to the precise answer. To make the operator exact, we use a higher-order operator that converts the query function in the desired manner, in this case, makes it exact. There are several types of these operators, e.g., exact, partial, and similar.

For making the retrieval exact, the symbol ! is placed in front of the query function. For example, "!*Agents*" returns only objects that have exactly the same motion description as the one given in M. Similarly, "!*Object_motion*" returns only motion descriptions of objects whose spatial characteristics (for example, exact bounding polygon, texture) match the spatial characteristics of a given object.

The # symbol placed in front of the query function is used for partial retrieval. Partial retrieval means that any of the motion or temporal characteristics of the given object should match. For example, "#*Agents*" will return all objects that match at least one of the motion descriptors.

## 4.2 The Query Language

The retrieval functions introduced in the previous section are embedded into the framework of a multimedia functional query language called EVA, described in Golshani and Dimitrova [1994]. EVA is the interface to a multimedia database system capable of storage, retrieval, management, analysis, and delivery of objects of various media types, including text, audio, images, and moving pictures. The language deals with the temporal and spatial aspects of multimedia information retrieval and delivery, in addition to the usual capabilities provided by the ordinary database languages. EVA has five groups of operators, namely: operations for querying and updating (i.e., editing) the multimedia information, operations for screen management, temporal operators, operators for specifying rules and constraints, and aggregation (computational) operators. EVA is an extension of a functional query language whose notation is based on that of conventional set theory. Both the original language and its extensions are formally defined in an algebraic framework. EVA is object oriented and supports objects, object classes, attributes and methods of objects, and relationships between objects. It has been ported onto several different platforms.

EVA provides a wide collection of operators that deal with text, graphics, scanned images, audio, and video. In addition, there are numerous type-independent operators such as set operators like union and set membership. One of the general operations is: the set construction operator. Generally, this has the form $\{f(x) \mid P(x)\}$, where $f(x)$ denotes the desired output objects, and $P(x)$ denotes the retrieval predicate which has to be true for those objects.

— *set operation symbols:* isin, isSubsetOf, isTrueSubsetOf, union, intersection, difference, Union, Intersection, noOf.

— *equality operators:* is, isnot.

— *temporal synchronization* (*for all media types*): sim, before, meets, equals, starts, at, finishes.

— *spatial composition* (*applied only to graphics, images, and video*): left, right, bottom, up, showIn, arrange.

— media-dependent operation symbols include

 —*text:* appendPar, cutPar, eqPar, keyword, isKeywordIn, parSim.

 —*graphics:* insPatch, pictureSum, fill, domain, colors, getPatch, getColor, restriction, scale, translate, dot, lineSeg, box, coincident, contains, disjoint, visible, bounded.

 —*images:* shift, zoom, superimpose, overlay, imageSim.

 —*audio:* intensity, extract, audioIns, audioLen, audioSim.

 —*video:* videoLen, pace, videoClip, videoIns.

—*integer operation symbols:* +, −, *, <, >, <=, >=, min, max, ave, sum, prod.

—*string operation symbols:* concat, strLen.

—*logical operation symbols:* and, or, implies, not.

To demonstrate the capabilities of EVA and how queries are constructed, we present a simplified example. The first part of the example will demonstrate the language without the OMV extensions. The video content retrieval extensions will be discussed once the appropriate distinctions are made.

We present the schema of a multimedia database system and then provide a few sample queries. The schema is represented as a graph whose nodes are object classes (in algebraic terms, sorts) and whose arcs are the relationships between object classes (represented as functions). Readers familiar with the algebraic framework would recognize this as a many-sorted algebra.

Illustrated in Figure 8, the schema models a college basketball multimedia database. The *basic types* in this system are String, Integer, Audio, Text, Video, while Player and School are *user-defined data types*. The highlighted portion appearing in dotted lines relates to the extension for video content retrieval described in Section 4.3.

The main difference between these basic and user-defined types is that the former constitute the application-independent constituents of any schema, whereas the user-defined types depend on each individual application. In

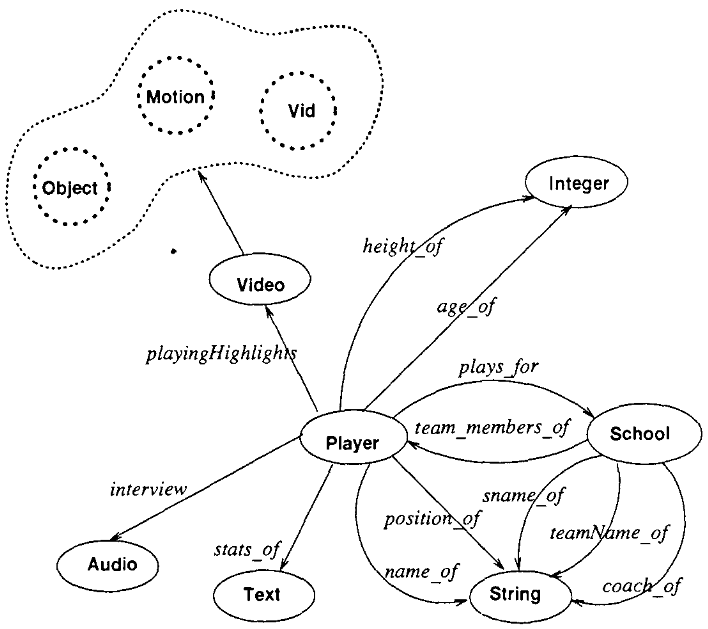


Fig. 8. Basketball schema.

Figure 8, all object types, both basic and user defined, appear in ovals. The attributes of objects and their relationships that are captured by arrows are the following functions:

| | |
|---|---|
| *name_of* | $: Player \rightarrow String$ |
| *position_of* | $: Player \rightarrow String$ |
| *state_of* | $: Player \rightarrow Text$ |
| *interview* | $: Player \rightarrow Audio$ |
| *playingHighlights* | $: Player \rightarrow Video$ |
| *height_of* | $: Player \rightarrow Integer$ |
| *age_of* | $: Player \rightarrow Integer$ |
| *plays_for* | $: Player \rightarrow School$ |
| *teammembers_of* | $: School \rightarrow \mathscr{P}(School)$ |
| *sname_of* | $: School \rightarrow String$ |
| *teamName_of* | $: School \rightarrow String$ |
| *coach_of* | $: School \rightarrow String$ |

Here are some queries on this database.

(1) List all guards who are taller than 190cm.

$$\{name\_of(P) \mid position\_of(P) is \text{“Guard”} \ and \ height\_of(P) > 190\}$$

This is a simple query on the nonmultimedia portion of the database. $P$ is a variable of type *Player*. The result is a list of names of players who satisfy the given conditions.

(2) Play video clips of all centers, and simultaneously display their statistics.

$$\{(name\_of(P), stats\_of(P))\, sim\ playingHighlights\_of(P) | \\ position\_of(P) is \text{“}Center\text{”}\}$$

While variable $P$ ranges over the elements of type *Player*, whenever the condition on position is satisfied, the name, statistics, and the corresponding video clip of the qualified player are displayed. “sim,” standing for “simultaneously,” is one of the synchronization operators that ensure proper semantics for presentations.

(3) Display the statistics of all Phoenix State University guards, and show their highlights before playing their interviews.

$$\{(name\_of(P), stats\_of(P))\, sim(playingHighlights\_of(P) \\ \text{before} interview(P)) | plays\_for(P) is \text{“}PhoenixStateUniversity\text{”} and \\ position\_of\ (P) is \text{“}Guard\text{”}\}$$

The result of this query is that, for every guard of the appropriate school, while their name and statistics are displayed, their video clip is presented first, and then their respective interview is played. The term “before” is another synchronization operator.

### 4.3 Querying Video Contents

Note that in the above queries, we treated *Video*, *Audio*, and *Text* as basic types in a similar manner to the type Integer, i.e., as objects whose contents can be displayed or presented, but no further specific characteristics are known about the contents. Our motion recovery algorithm and specifically the OMV functions enable us to treat *Video* in a different way, as described below.

Grosky's [1994] categorization makes a distinction between the physical basic data types and the conceptual data types. He adopts a generic model to represent *content-independent* and *content-based* properties of multimedia objects. Content-independent properties are related to the physical data object itself (uninterpreted data) as well as synchronization and storage information. Content-based properties refer to relationships between nonmultimedia real-world application entities and multimedia objects. The content-based properties associate semantics to the object at various levels.

A binary object containing the video stream that corresponds to the playing highlights of a particular player is an instance of physical data type. The extracted spatial and motion characteristics are stored in the conceptual data type. The queries on the content of the video data are directed to the conceptual video data type.

The *conceptual video data type* is molded from the spatiotemporal hierarchy presented in Figure 7 using the object-motion-video structures. The OMV retrieval functions augment EVA's retrieval capabilities since they turn the physical object Video into a conceptual one, i.e., an object with its own specific set of properties that can be incorporated into queries for more-precise questions. The extension to the schema which enhances the video type to be a conceptual type appears in the dotted line in Figure 8. Below is a list of operators that augment the retrieval capabilities based on the OMV retrieval functions:

— *Function Composition:* given functions $f: X \rightarrow Y$ and $g: Y \rightarrow Z$, the composition is $f \circ g(x) = g(f(x))$. For *example*, Given an object (any characteristic) in a video sequence $v_1$, retrieve objects in another video sequence $v_2$ which have similar motion.

$$Agents(Object\_motion(o_1, v_1), v_2)$$

— *Temporal Combination Functions:* $f\theta g$ where $\theta \in \{$*before, meets, simultaneously, starts, finishes*$\}$. Although the same syntax is used, these should not be mistaken for the synchronization operators. In this case, no confusion is expected since the context would determine the designation of the operator. An example of usage for this type of operator is the query "retrieve all the sequences in which a tall person is waving while the president walks."

— *Spatial Combination Functions:* $f\xi g$ where $\xi \in \{$*next, behind, inFront, left, right*$\}$.

Using the above combinators and the OMV structure, many new types of queries that refer to the contents of video sequences can be specified. Specifically, we can express queries that refer to the contents of video sequences. Examples include the following:

(1) "Retrieve all the video sequences with the longest successful shots." This query translates into "retrieve all the video sequences for which the length of the trajectory of the ball is maximum."
(2) "Spell out all the details of movements of the players whose height is greater than 200cm." This query is good for analyzing the pattern in which certain players move and achieve the score.
(3) "Find the video sequences in which the player is wearing a blue shirt." The "blue shirt" is inferred using image analysis.

The target language is a visual one that allows for inclusion of spatial properties (sketches) and exact and inexact images. The notation presented in this article is the basis for the visual query interface [Dimitrova 1995; Michael 1994].

## 5. AN ARCHITECTURE FOR VIDEO CLASSIFICATION AND RETRIEVAL

In the previous sections we introduced a model for video classification which exploits motion recovery and representation. In this section, we discuss a general architecture for video database retrieval based on the model. The proposed architecture, as presented in Figure 9, consists of:

— Insertion module

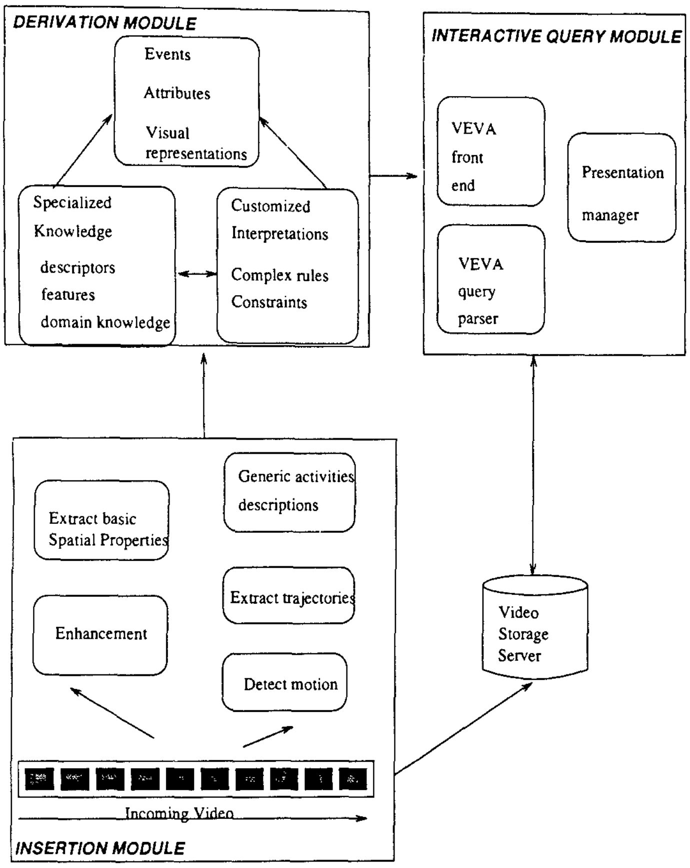


Fig. 9. An architecture for video classification and retrieval.

— Derivation module
— Interactive query module
— Video storage server.

The insertion module is responsible for initial analysis of the incoming video signal. It consists of a suite of operators for image enhancement, operators for the extraction of basic spatial properties, and operators for motion detection and the extraction of motion trajectories. With respect to the spatiotemporal hierarchy, this module is an implementation of the operators between the lowest level of the hierarchy (raw physical data) to the intermediate representation. Currently, the functionalities for spatial analysis are supplied by the Khoros computer vision environment [Rasure et al. 1990]. The extraction of image features, finding regions, and thinning operators are performed by calls to Khoros functions. Although features are automatically extracted, the process of feature selection is manual. For example, we can apply an operator for image segmentation and find the regions in a video frame. However, the selection of regions of importance is decided by the application designer. The automation of this whole process is possible for strictly limited application domains such as industrial monitoring, domain-specific video editing, camera surveillance, and others. The motion detection and tracing operators are also part of the insertion module. The implementation of the motion-tracing algorithm is given in Section 5.2.

The derivation module consists of operators for translation of the extracted features into meaningful descriptions for retrieval. Each application typically defines its own set of meaningful entities and events and has its own interpretation of the same. In our video model and language, the extracted properties are represented by predicates. The derivation module provides the mapping between the visual properties extracted form the video sequences which are geometric by nature and the algebraic representation which is used for querying.

The query module consists of a visual front-end for query composition, a visual query parser, a schema designer, and a presentation manager. The schema designer and the visual front-end are incorporated into the visual query language VEVA [Dimitrova 1995]. The VEVA prototype serves as a testbed for development of new algorithms for video/image segmentation, video parsing, feature selection, and classification. The prototype has been implemented in Tcl/Tk [Osterhout 1994] with the added image and video widgets on top of an existing MPEG encoder [Rowe and Smith 1992].

The video storage server is envisioned to be a disk array serving as a repository of the video sequences. At this point we use a simple file system for storing a limited number of MPEG compressed video sequences.

## 5.1 The Visual Query Language VEVA

Spatial and motion characteristics of objects, derived from images and video sequences respectively, are inherently visual. In this section, we outline the design of a multimedia database language which has well-defined semantics in both character-based and icon-based paradigms.

Defined within the algebraic framework described above, VEVA is a visual query language that provides all the necessary constructs for retrieval and management of multimedia information. The basis for the language is a schema (algebraic signature) which contains entity types (both user-defined and application-independent types) and the associated operators [Golshani and Dimitrova 1994]. By using these operators, the user can visually specify a query for the desired objects in a simple way. VEVA has a formal grammar with which the set of acceptable expressions can be generated. The grammar for the visual language VEVA is given using visual rules in the style of a picture description language which was developed within the syntactic approach to pattern recognition [Schalkoff 1989]. The grammar rules contain nonterminal and terminal icons. The rules are given as graph-rewriting rules where the left-hand side is a nonterminal icon, and the right-hand side is a graph containing nonterminal and terminal icons connected with customized links.

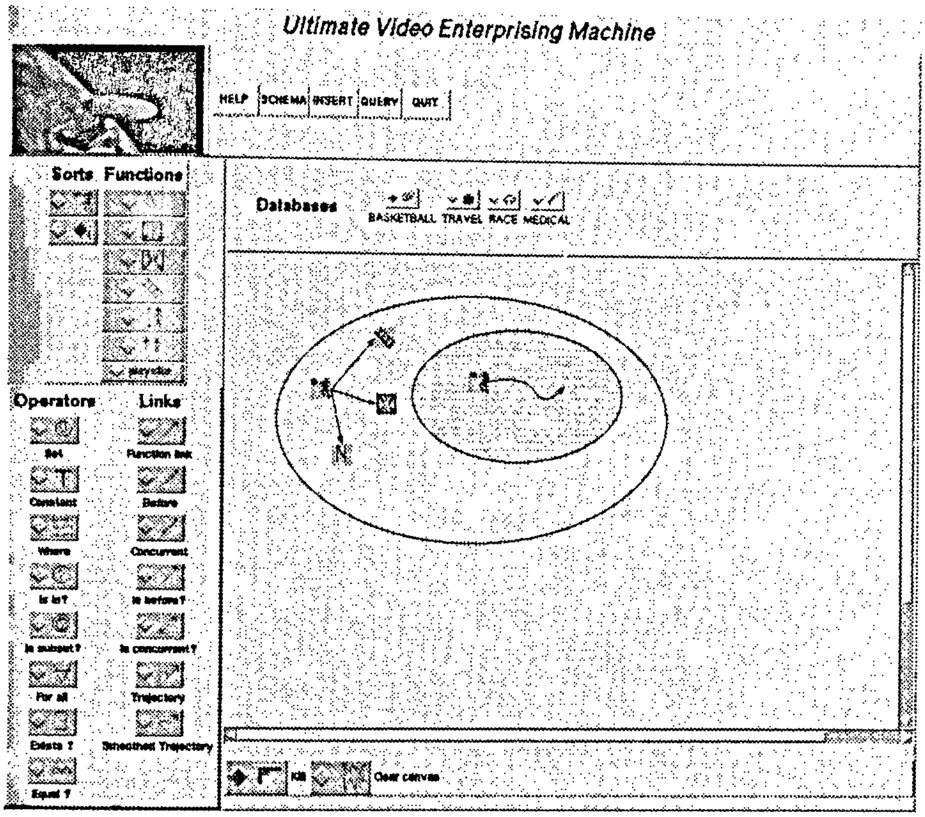


Fig. 10. Visual query involving a trajectory description.

Parsing of visual expressions in VEVA is a process of determining the structure of the workspace. Note that parsing is the first step of the VEVA language processing, because lexical analysis is not necessary. All available icon symbols can be drawn from the given pallette and connected by a set of permissible links. Thus, every expression that is drawn is lexically correct. The execution process begins by parsing the contents of the VEVA workspace. The algorithm finds the top-level set expressions which may contain other set expressions. Translated into visual terms, this algorithm finds the enclosed visual expressions or other iconic elements within a given oval. The algorithm calls the set evaluation procedure recursively for the sets that are contained in it, until there are single sets with simple function-predicate expressions left. The evaluated sets can be connected with temporal links which prescribe the order in which the resulting objects should be presented by the presentation manager. If the evaluated expression contains temporal links, then the parsed execution order is delivered to the presentation manager.

An example query is given in Figure 10. As we stated earlier, VEVA allows for visual queries in which we can specify the path of a moving object. In this example the input trajectory for the player is given as a smoothed trajectory. The visual query given in Figure 10 will select those video sequences from the

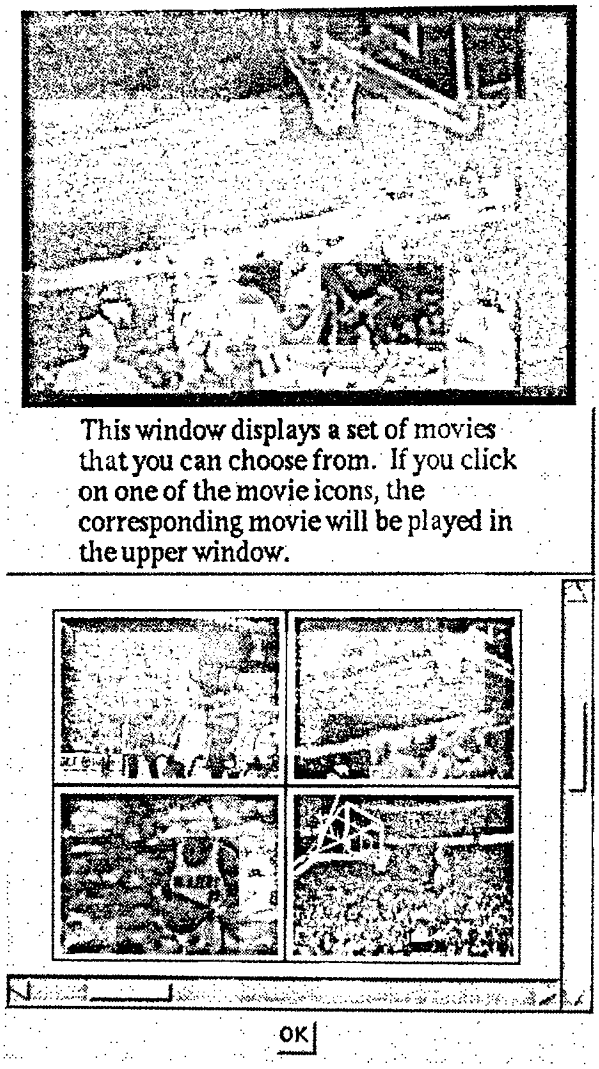

This window displays a set of movies that you can choose from. If you click on one of the movie icons, the corresponding movie will be played in the upper window.

Fig. 11. Results from the visual query. Courtesy of NBA Entertainment.

repository in which the player's trajectory is similar to the one drawn by the user and display the name and the position of the player. The result of the query is shown in Figure 11. The user can browse and play the selected video segments.

Various models have been proposed for temporal synchronization, composition, and presentation in multimedia applications, for example, Buchanan and Zellweger [1993] and Little et al. [1991]. On the other hand, a number of models for content-based access of digital video has been proposed [Arman et al. 1994; Bobick 1993; Rowe et al. 1994; Swanberg et al. 1993; Zhang et al. 1994]. However, a general formal model and a language for content representation, composition, and querying of digital video based on the temporal and the spatial properties of objects found in the video sequences has not been

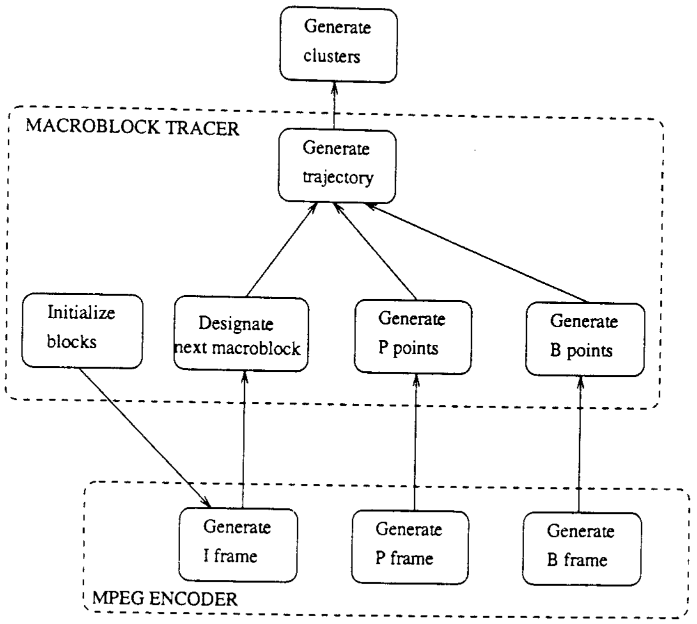


Fig. 12. Macroblock motion extraction.

offered yet. Our video model and language VEVA attempts to unify the presentation aspects as well as content representation aspects of multimedia objects.

## 5.2 Implementation of Macroblock Tracing

The motion-tracing algorithm is a part of the derivation module in our video classification architecture. We have tested our ideas by implementing the motion-tracing and extraction algorithm under Solaris 2.3 using the MPEG encoder produced by the Digital Video research team at the University of California, Berkeley. A functional view of the MPEG-based motion extraction is given in Figure 12. We have introduced functions for extraction of motion vectors during the generation of P- and B-frames. We use the motion-tracing algorithm to compute the macroblock trajectories.

The performance results are shown in Figure 13. We have tested our motion-tracing algorithm by ranging the number of macroblocks being traced from zero to all macroblocks. The input video sequence is the standard table tennis sequence, which consists of 10 frames, each of size 352 by 240 pixels. This sequence is a good performance test case, because it has background and foreground motion. The encoding frame pattern is IBBBPBBBBP. This means that all the input frames are used for video encoding. If only encoding is performed without any motion-tracing algorithm, the total elapsed time is 32.6 seconds (+/− 0.05 seconds). With the motion-tracing algorithm, the time increase is evident with the increase of the number of blocks. Starting

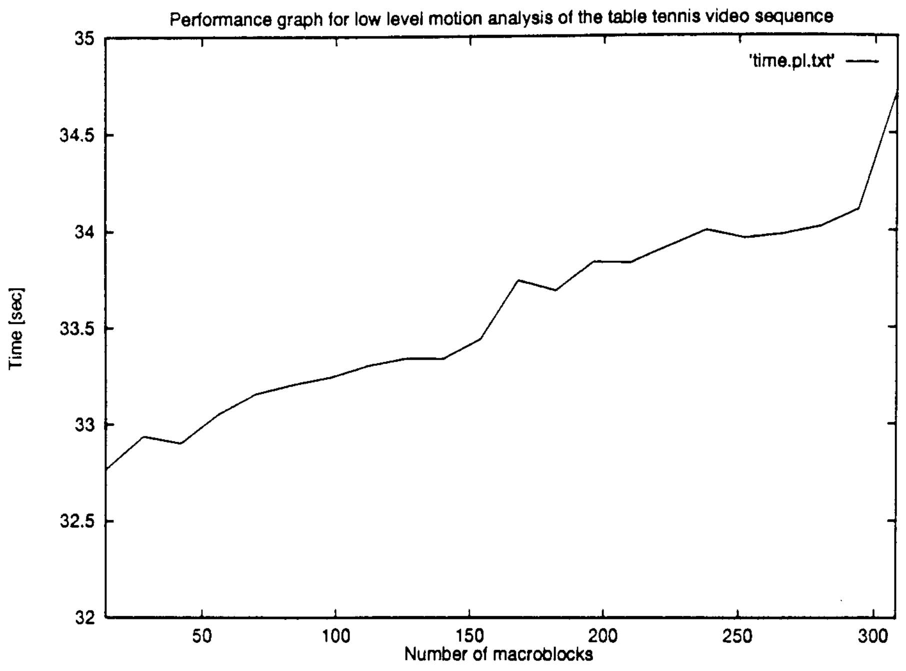


Fig. 13. Performance of object motion tracking.

with one macroblock, we get elapsed time of 32.76 seconds for encoding and tracing which is 0.16 seconds more than the previous case. As shown in Figure 13, when the number of traced macroblocks increases up to 300, the elapsed time goes up to 34.72 seconds. This shows that even if we keep track of the motion of all the macroblocks we have a time increase of 6%.

The gain in MPEG compression is mostly achieved by exploiting temporal redundancy. MPEG avoids coding the same block twice by storing/sending over the displacement vector from the previous image. Thus, the basic assumption is that the frame pattern used for MPEG compression is going to contain P- and B-frames.

Our algorithm for motion tracing would have very limited application if the stream to be encoded is using only I-frames. In that case, there the motion algorithm cannot find any motion vectors to take advantage of. If high quality of encoded video is crucial to the application at hand, then the algorithm has to be rewritten, so that motion estimation is performed using some imaginary frame pattern which would not have any impact on the encoded video stream. Then the motion-tracing algorithm would be performed on the obtained motion estimates. In this case, the motion information that is obtained from the encoder is in the forward vectors of the P-frames only. From the P-frame to the next I-frame we do not have any motion information. We have several choices:

(a) We can make a prediction for the motion vector between the P-frame and the next I-frame. This prediction is a guess that we can use the same motion vector as the vector for the P-frame. This solution does not introduce additional overhead. The problem is that it relies not only on the assumption for the continuity of motion but also assumes that the motion is constant.

(b) We can perform the actual search and compute the motion vector for the blocks from the P- to the next I-frame. This means that we will be adding much more compute cycles than it is necessary for the encoding process.

We also need much more complicated motion models to recover the true motion of the objects in the case of complicated camera motion. For example, when we have the camera focus on a moving object, then the object appears to be stationary. The motion of the object is implied by the macroblock vectors of the background. More-sophisticated relative-motion detection algorithms are needed. This work is part of our ongoing SunSet Multimedia Information System project [Golshani and Dimitrova 1994; Michael 1994].

## 6. CONCLUSIONS

From the point of view of video retrieval, the video technology has not seen much progress from the days when film editors examined each and every frame by hand in order to find the exact place of each cut. In fact, despite the introduction of many video editing systems such as VideoShop and Adobe's Premiere, much of retrieval is done by either time pointers (e.g., the frame counter), visual proxies, or various types of graphical or descriptive pointers. What is clearly missing from the video technology is the ability to locate and retrieve video clips that contain an object with specific characteristics, particularly with respect to movements. Video databases can be useful to many application areas such as education, business, medicine, and more prominently, entertainment. As such, the value of better and more-equipped video systems are becoming clearer. While many aspects of video systems, such as presentation editing tools, have seen significant improvement, our progress on content-based retrieval has not been as forthcoming.

We believe that our attempts to address the above needs must start with a modeling mechanism that allows for the representation of semantic knowledge from both spatial and temporal features of the objects in video sequences. Computing high-level motion description can be done independently of recognizing objects [Allmen 1991]. We elaborate on this property by showing that the recovery of object trajectories can be performed without prior knowledge of objects undergoing motion. The goal is to have both: independent retrieval along the temporal and the spatial hierarchies as well as retrieval of combined features from the spatial and the temporal hierarchies. We treat motion vectors extracted during the motion compensation phase of video encoding as coarse-level optical flow that is further used for intermediate- and high-level motion description. Motion information extraction is then carried out at low level by motion vector detection, at the intermediate level by motion tracing, and the high level by associating an object and a set of trajectories with recognizable activities.

In our object motion representations, we provide various levels of precision of trajectory representation. Retrieval functions based on these representations offer a wide spectrum of approximation in the process of matching. We need to relate the motion at a higher level of abstraction of the object to the detailed motion of parts of objects. Events can be represented in a form that is common in the image-understanding and interpretation area: predicates, temporal networks, etc.

## ACKNOWLEDGMENTS

The authors gratefully acknowledge assistance from the guest editors. We would like to thank Tom Foley for clarifying our ideas on B-spline representation. We would like to thank Guru Banavar for the numerous discussions and insightful comments on the visual language. The authors would also like to thank K. P. Lee for suggesting Tcl/Tk as the implementation platform for the interface.

## REFERENCES

ALLMEN, M. C. 1991. Image sequence description using spatiotemporal flow curves: Toward motion-based recognition. Ph.D. thesis, Univ. of Wisconsin, Madison, Wisc.

ARMAN, F., DEPOMMIER, R. HSU, A., AND CHIU, M.-Y. 1994. Content-based browsing of video sequences. In *Proceedings of ACM Multimedia '94* (San Francisco, Calif.). ACM Press, New York, 97–103.

BOBICK, A. F. 1993. Representational frames in video annotation. In *Proceedings of the 27th Annual Conference on Signals, Systems and Computers*. IEEE Computer Society Press, Los Alamitos, Calif.

BUCHANAN, M. C. AND ZELLWEGER, P. T. 1993. Automatically generating consistent schedules for multimedia documents. *Multimedia Syst. J. 1*, 2 (Sept.).

DAVIS, M. 1993. Media streams: An iconic visual language for video annotation. In *Proceedings of the IEEE Symposium on Visual Languages* (Bergen, Norway). IEEE, New York, 196–202.

DIMITROVA, N. 1995. Content classification and retrieval of digital video based on motion recovery. Ph.D. thesis, Arizona State Univ., Tempe, Ariz.

DIMITROVA, N. AND GOLSHANI, F. 1994. Rx for semantic video database retrieval. In *Proceedings of ACM Multimedia '94* (San Francisco, Calif.). ACM Press, New York, 219–226.

DUDA, R. O. AND HART, P. E. 1973. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York.

FARIN, G. 1990. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, New York.

FURHT, B. 1994. Multimedia systems: An overview. *IEEE Multimedia 1*, 1 (Spring), 47–59.

GODDARD, N. 1992. The perception of articulated motion: Recognizing moving light displays. Ph.D. thesis, Univ. of Rochester, Rochester, N.Y.

GOLSHANI, F. AND DIMITROVA, N. 1994. Retrieval and delivery of information in multimedia database systems. *Inf. Softw. Tech. 36*, 4 (May), 235–242.

GROSKY, W. AND MEHROTRA, R. 1989. Image database management. *IEEE Comput. 22*, 12 (Dec.), 7–8.

GROSKY, W. I. 1994. Multimedia information systems. *IEEE Multimedia 1*, 1 (Spring), 12–24.

GUPTA, A., WEYMOUTH, T., AND JAIN, R. 1991a. Semantic queries in image databases. In *Visual Database Systems II*, E. Knuth and L. Wegner, Eds. Elsevier Science Publishers (North-Holland), Amsterdam, 201–215.

GUPTA, A., WEYMOUTH, T., AND JAIN, R. 1991b. Semantic queries with pictures: The VIMSYS Model. In the *Conference on Very Large Data Bases*. VLDB Endowment, Saratoga, Calif.

HAMPAPUR, A., WEYMOUTH, T., AND JAIN, R. 1994. Digital video segmentation. In *Proceedings of ACM Multimedia '94* (San Francisco, Calif.). ACM Press, New York, 357–364.

HORN, B. K. AND SCHUNCK, B. G. 1981. Determining optical flow. *Artif. Intell. 17*, 1–3, 185–203.

JOHANSSON, G. 1976. Spatio-temporal differentiation and integration in visual motion perception. *Psychol. Res. 38*, 4, 379–393.

KOLLER, D., WEBER, J., AND MALIK, J. 1993. Robust multiple car tracking with oclusion reasoning. Tech. Rep. CSD-93-780, EECS Dept., Univ. of California, Berkeley, Calif. Nov.

KUBOTA, H., OKAMOTO, Y., MIZOGUCHI, H., AND KUNO, Y. 1993. Vision processor system for moving-object analysis. *Mach. Vis. Appl. 7*, 1, 37–43.

LEGALL, D. 1991. MPEG: A video compression standard for multimedia applications. *Commun. ACM 34*, 4 (Apr.), 46–58.

LITTLE, T., AHANGER, G., FOLZ, R., GIBBON, J., REEVE, F., SCHELLENG, D., AND VENKATESH, D. 1993. A digital on-demand video service supporting content-based queries. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York, 427–436.

LITTLE, T., GHAFOOR, A., CHEN, C., CHANG, C., AND BERRA, P. 1991. Multimedia synchronization. *Data Eng. 14*, 3, 26–35.

MATTISON, P. E. 1994. *Practical Digital Video with Programming Examples in C.* John Wiley and Sons, New York.

MICHAEL, N. 1994. VEENA—a visual query language. M.S. thesis, Arizona State Univ., Tempe, Ariz.

NAGASAKA, A. AND TANAKA, Y. 1992. Automatic video indexing and full-video search for object appearances. In *Visual Database Systems II*, E. Knuth and L. Wegner, Eds. Elsevier Science Publishers (North-Holland), Amsterdam, 113–127.

OSTERHOUT, J. K. 1994. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, Mass.

OTSUJI, K. AND TONOMURA, Y. 1993. Projection detection filter for video cut detection. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York, 251–257.

PATEL, K., SMITH, B. C., AND ROWE, L. A. 1993. Performance of a software MPEG video decoder. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York, 75–82.

RASURE, J., ARGIRO, D., SAUER, T., AND WILLIAMS, C. 1990. Visual language and software development environment for image processing. *Int. J. Imaging Syst. Tech. 2*, 2, 183–199.

ROHR, K. 1994. Towards model-based recognition of human movements in image sequences. *CVGIP: Image Understanding 59*, 1 (Jan.), 94–115.

ROWE, L. A. AND SMITH, B. C. 1992. A continuous media player. In the *3rd International Workshop on Network and OS Support for Digital Audio and Video.* Springer-Verlag, Berlin, 334–344.

ROWE, L. A., BORECZKY, J. S., AND EADS, C. A. 1994. Indexes for user access to large video databases. In *Proceedings of SPIE IS and Symposium on Storage and Retrieval for Image and Video Databases* (San Jose, Calif.). SPIE.

SCHALKOFF, R. J. 1989. *Digital Image Processing and Computer Vision.* John Wiley and Sons, New York.

SWANBERG, D., SHU, C.-F., AND JAIN, R. 1993. Knowledge guided parsing in video databases. In *Image and Video Processing Conference; Symposium on Electronic Imaging: Science and Technology.* Vol. 1908. IS & T/SPIE, 13–24.

TEODOSIO, L. AND BENDER, W. 1993. Sallient video stills: Content and context preserved. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York.

TEODOSIO, L. AND MILLS, M. 1993. Panoramic overviews for navigating real-world scenes. In *Proceedings of ACM Multimedia '93* (Anaheim, Calif.). ACM Press, New York.

WEISS, R. 1994. Content-based access to algebraic video. Tech. Rep., Massachusetts Inst. of Technology, Cambridge, Mass.

ZHANG, H., GONG, Y., SMOLIAR, S., AND TAN, S. Y. 1994. Automatic parsing of news video. In *Proceedings of the International Conference on Multimedia Computing and Systems* (Boston, Mass.). IEEE Computer Society Press, Los Alamitos, Calif., 45–54.

Received August 1994; revised February 1995; accepted July 1995

**Barton** MIT Libraries' Catalog

MIT Libraries

**Search Full Catalog:**
- Basic
- Advanced

**Search only for:**
- Conferences
- E-resources
- Journals
- MIT Theses
- Reserves
- more...

- Your Account
- Help with Your Account
- Your Bookshelf
- Previous Searches

Ask Us! Other Catalogs Help

## Full Record

**Permalink for this record:** http://library.mit.edu/item/000395517

**Results List** | **Add to Bookshelf** | **Save/Email**

Choose format: **Standard** | **Citation** | **MARC tags**

**Record 6 out of 8**

Prev record Next record

```
FMT    SE
LDR    02202cas  2200517 a 45r0
003    MCM
005    20010608151522.0
008    890907c19899999nyuqr1p      0   a0eng d
010    |a    89646863  |z sn 89034098
0220   |a 1046-8188 |y 0734-2047
030    |a ATISET
035    |a (SFX)954925591438
035    |a MITb10395517
035    |a (OCoLC)20309308
035    |a (OCoLC)20309247
037    |b Association for Computing Machinery, 11 West 42nd St., New York, NY 10036
040    |a ECO |c ECO |d MYG |d OCL |d NSD |d DLC |d NST |d CAS |d EYM |d NST |d HUL |d NST |d IUL |d NST |d WAU |d NST |d MYG
042    |a lc |a nsdp
05000  |a HF5548.2 |b .A335
08200  |a 651.8 |2 20
099    |a No Call #
2100   |a ACM trans. inf. sys.
222 0  |a ACM transactions on information systems
24500  |a ACM transactions on information systems : |b a publication of the Association for Computing Machinery.
24610  |a Transactions on information systems
24623  |a Association for Computing Machinery transactions on information systems
260    |a New York, NY : |b The Association, |c c1989-
300    |a v. : |b ill. ; |c 26 cm.
310    |a Quarterly
3620   |a Vol. 7, no. 1 (Jan. 1989)-
4901   |a ACM series on computing methodologies
500    |a Title from cover.
5101   |a Computer & control abstracts |x 0036-8113 |b Jan. 1989-
5101   |a Electrical & electronics abstracts |x 0036-8105 |b Jan. 1989-
5101   |a Physics abstracts |x 0036-8091 |b Jan. 1989-
5102   |a Chemical abstracts |x 0009-2258 |b 1989
530    |a Also available via the World Wide Web.
650 0  |a Electronic data processing |v Periodicals.
650 0  |a Information storage and retrieval systems |v Periodicals.
650 0  |a Information retrieval |v Periodicals.
7102   |a Association for Computing Machinery.
78000  |t ACM transactions on office information systems |x 0734-2047 |w (OCoLC)8707220 |w (DLC)  83642576
830 0  |a ACM series on computing methodologies.
85641  |u http://portal.acm.org/tois/
CAT    |a CONV |b 00 |c 20010620 |l MIT01 |h 1704
CAT    |a EDWARDSJ |b 00 |c 20031211 |l MIT01 |h 1620
CAT    |a WPOWERS |b 00 |c 20050606 |l MIT01 |h 1705
CAT    |a WPOWERS |b 00 |c 20050606 |l MIT01 |h 1717
CAT    |a EDWARDSJ |b 00 |c 20050721 |l MIT01 |h 1525
CAT    |a EDWARDSJ |b 00 |c 20071218 |l MIT01 |h 1519
CAT    |a lti0904 |b 00 |c 20090523 |l MIT01 |h 1747
CAT    |a MARCit |b 00 |c 20130823 |l MIT01 |h 1102
CAT    |a BATCH-UPD |b 00 |c 20141027 |l MIT01 |h 1601
95640  |u http://owens.mit.edu/sfx%5Flocal?rft.object%5Fid=954925591438&url%5Fver=Z39.88-2004&ctx%5Fver=Z39.88-2004&ctx%5Fenc=info:ofi/enc:UTF-
       8&rfr%5Fid=info:sid/sfxit.com:opac%5F856&url%5Fctx%5Ffmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore%5Fdate%5Fthreshold=1&svc%5Fval%5Ffmt=info:ofi/fmt:kev:mtx:sch
049    |a MYGE
910    |a wj990218/4
936    |a Vol. 7, no. 4 (Oct. 1989) LIC
946    |m ACM
PST7   |0 Z30 |1 000395517000140 |b LSA |c OCC |o ISSUE |d 15 |y 00000 |f N |r MIT60-001515054 |n 7 |h HF.A184 |2 Hfcl |a MCM |3 Serial |4 Library Storage Annex |5 Off Car
       Collection |6 OCC 60 |p Avail
LDR       ny   22     3n 4500
008    0106234p    0   0001uu   0031211
```

Ex. I Page 1 of 2

| | |
|---|---|
| **004** | 000395517 |
| **85281** | \|a MCM \|b NET \|c EJ \|h **See URL(s) \|z MIT Access Only |
| **LDR** | ny 22 3n 4500 |
| **008** | 0106234p 8 1001uueng0080523 |
| **004** | 000395517 |
| **85271** | \|a MCM \|b LSA \|c OCC \|h HF.A184 \|2 Hfcl |
| **866** | \|b Library Storage Annex \|c Off Campus Collection \|h HF.A184 \|a v.7(1989)-v.26(2008) |
| **001** | 000395517 |
| **SFX91** | \|s 0-0-0-3-9-5-5-1-7 \|l MIT01 \|9 001 \|z MARCit~~~ \|p Avail \|f 002 \|a v.7:no.1 (1989:Jan.)- |
| **CNTR1** | \|a (SFX)954925591438 |
| **URL** | \|u http://portal.acm.org/tois/ \|9 001 |
| **U564** | \|u http://owens.mit.edu/sfx%5Flocal?rft.object%5Fid=954925591438&url%5Fver=Z39.88-2004&ctx%5Fver=Z39.88-2004&ctx%5Fenc=info:ofi/enc:UTF-8&rfr%5Fid=info:sid/sfxit.com:opac%5F856&url%5Fctx%5Ffmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore%5Fdate%5Fthreshold=1&svc%5Fval%5Ffmt=info:ofi/fmt:kev:mtx:sch \|9 001 |
| **U564** | \|u http://portal.acm.org/tois/ \|9 002 |
| **LU564** | \|u http://portal.acm.org/tois/ \|f 002 |
| **SMHL** | \|b Internet Resource \|c Electronic Journal \|h **See URL(s) \|a v.7:no.1 (1989:Jan.)- \|9 001 |
| **SYS** | 000395517 |

Prev record Next record

## Basic Search of Full Catalog

Search type:
Keyword
Title begins with...
Title Keyword
Author (last name first)
Author Keyword
Call Number begins with...
----- Scroll down for more choices -----

Search for:

Search

MITLibraries

Barton Questions: Ask Us! | Contact Us
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA

-- Quick Links --

PROCEEDINGS
SPIE—The International Society for Optical Engineering

25th AIPR Workshop

# Emerging Applications of Computer Vision

**David Schaefer**
**Elmer F. Williams**
*Chairs/Editors*

**16–18 October 1996**
**Washington, D.C.**

*Sponsored by*
SPIE—The International Society for Optical Engineering
AIPR Executive Committee

*Published by*
SPIE—The International Society for Optical Engineering

SPIE
P
PROCEEDINGS SERIES

**Volume 2962**

TA1632
A15
1996

The papers appearing in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors or by SPIE.

Please use the following format to cite material from this book:
Author(s), "Title of paper," in *Emerging Applications of Computer Vision,* David Schaefer, Elmer F. Williams, Editors, Proc. SPIE 2962, page numbers (1997).

ISSN 0277-786X
ISBN 0-8194-2366-1

Published by
**SPIE—The International Society for Optical Engineering**
P.O. Box 10, Bellingham, Washington 98227-0010 USA
Telephone 360/676-3290 (Pacific Time) • Fax 360/647-1445

Printed in the United States of America.

# Autonomous Video Surveillance

Bruce E. Flinchbaugh and Thomas J. Olson

Texas Instruments Corporate Research Laboratories
P.O. Box 655303, M/S 8374, Dallas, TX 75265

## ABSTRACT

This presentation highlights needs for autonomous video surveillance in the context of physical security for office buildings and surrounding areas. Physical security is described from an operational perspective, defining the principal responsibilities and concerns of a physical security system. Capabilities and limitations of current video surveillance technology are described, followed by examples of how computer vision techniques are being used and advanced for autonomous video surveillance systems.

Keywords: video surveillance, scene monitoring, computer vision, security

## 1. Physical Security Systems and Operations

Major activities of physical security for office buildings today are: access control, intrusion detection, guard patrols, CCTV surveillance, alarm monitoring, response dispatching, and investigations. Autonomous video surveillance technology will ultimately improve productivity and effectiveness in all of these activities. The primary responsibilities of physical security are described below.

### 1.1 Access Control

The most common approach to physical security is to control who may enter a building or an area at the perimeter. By restricting access to trusted individuals (e.g., employees), many opportunities for security breaches are eliminated. Door locks and keys provide much of this security, and many buildings deploy guards at building entrances to control who enters. The primary automatic access control technology in use today is provided by electronic badge reader systems. In this approach, an electronically readable badge is issued to each person with access privileges, and the person may use the badge like a key to open doors where guards are not stationed. A drawback of keys and electronic badges is that they may be used by unauthorized individuals to gain access. For tighter security in access control, a variety of biometric access control technologies are available to measure physical characteristics of people: voice verifcation, retina scanners, fingerprint scanners, hand scanners, face recognition, and body weight measurements. The primary limitation of access control technology is that it does not defeat security breaches by insiders.

### 1.2 Intrusion Detection

A second line of defense against physical security breaches is to detect situations where people gain unauthorized access to a facility. This may occur at regular access control points (e.g., doors) or at other places (e.g., windows and fences). For example, an unauthorized person might enter by "piggy backing" on access by authorized person (i.e., by slipping through a door before it closes), or by any number of physical "breaking and entering" approaches. Various technologies are available to detect intrusions. The most commonly used devices are door switches and infrared motion detectors, like those found in many home security systems. Infrared motion detectors operate by detecting rapid heat changes in an area that are caused when a person enters the area. Other intrusion detection techniques include sound detectors, glass-break detectors, light beams, and various other electromagnetic change detectors. Although an intrusion detector may be used to signal alarms, it does not provide a description of the specific situation that causes an alarm, thereby requiring a subsequent follow-up action to respond to the alarm or investigate the incident later.

## 1.3 Guard Patrols

In addition to guards who monitor entrances for access control, a common security practice is for guards to periodically patrol sites to visually confirm that the premises are secure and to identify and report adverse conditions (e.g., missing property and damage from vandalism). Nearly all of the technology that supports these visual surveillance activities in practice today involves mechanical and electronic devices that the guards use to prove that they visited specific areas and to report observations. However, some advanced robotic security systems attempt to accomplish various guard patrol responsibilities. For example, mobile robots equipped with sonic sensors for navigation and detection have been installed for experimental security applications to patrol buildings, but this approach faces many problems to overcome before it can compete effectively with guards and more-reliable video surveillance approaches.

## 1.4 CCTV Surveillance

Closed circuit television (CCTV) camera networks that supply video data to security system centers are often used to support physical security operations in buildings and surrounding areas. In a small system, a few cameras may be cabled to TV monitors for remote viewing by guards or other security personel. Another common practice is the use of time-lapse video tape recorders to record video data from one or more cameras. Some time-lapse recorders provide a multiplexed recording capability so that several cameras may be recorded on a single tape. In large CCTV security systems, many cameras are cabled to an array of TV monitors and video tape recorders, to support live observations as well as after-the-fact investigations using recorded video data.

CCTV cameras may be mounted with a fixed field of view, or mounted on a pan and tilt mechanism that can be remotely controlled by an observer to view a wider area. Video cameras may also be mounted on a small platform that moves along a track, enabling a single camera to scan a much wider area (e.g., by moving along a long wall in a large parking garage).

The primary autonomous video surveillance systems available today are known as "video motion detectors" or "VMDs". In principle, VMDs can be programmed for various tasks such as intrusion detection and to signal alarms for fairly complex situations. For example, some VMDs can be programmed to signal an alarm when something moves across the field of view from left to right, while not signaling an alarm when something moves from right to left. However, in practice available VMDs produce too many false alarms in typical environments [6], and they require substantial operator training and experience to program the system.

## 1.5 Alarm Monitoring and Response Dispatching

Large physical security systems generally have a centralized control center where security staff monitor alarms, dispatch guards to respond to incidents, and maintain a record of the incidents and their resolution. In some cases, control center operators actively monitor remote cameras to detect incidents and engage in wide-ranging visual surveillance tasks. Although people are very good at these tasks, long periods of routine activity or inactivity in a scene make this monitoring job tedious. It is also impractical for one person to reliably and simultaneously monitor hundreds of cameras. Thus control center operators typically spend most of their time monitoring and processing alarms from autonomous devices.

### 1.6 Investigations

Although the primary goal of physical security is to prevent security breaches, or to detect and respond in time to minimize the impact, a rapid and accurate capability to investigate incidents after the fact is also a key activity of physical security. Unlike retail store security (where a frequent threat is shoplifting by customers) office building security often faces the challenge of determining which insider has stolen something after it is determined that a theft has occurred. In this matter, differences between physical security and information security begin to blur. For example, evidence for theft of trade secrets might be visually observable (as when an employee opens a file drawer and reads or copies a document) even though the document may be physically replaced in the drawer within a few minutes. For cases where a large material property item has been stolen, time-lapse video recordings of building entrances and exits provide a useful record of events for investigators to search for suspects. However, the limited video surveillance tool for investigators in this regard remains a video tape player with a fast-forward button, and automatic video surveillance systems to assist investigators in detecting theft of intellectual property are beyond the state of the art.

## 2. Computer Vision in Autonomous Video Surveillance Systems

Texas Instruments has demonstrated a variety of autonomous video surveillance capabilities involving computer vision. In this section we summarize several TI capabilities for surveillance in office building environments.

Our approach to video surveillance has been to exploit data that is readily computed from incoming video streams by using domain-specific constraints and contextual information to interpret the data. The video processing is primarily a matter of detecting areas of dynamic change via thresholded image differences and forming connected components for the subsequent analysis. In addition to devising new methods for providing more accurate and reliable information about moving entities and their surroundings, we are focusing on computer vision for interpreting scene events involving complex spatio-temporal conditions and interactions.

### 2.1 People Tracking and Position Mapping

The most basic function of a surveillance system is to provide situational awareness, i.e., to inform security personnel of what is going on in the monitored area. TI has demonstrated an end-to-end real-time video surveillance system that uses a visual memory paradigm to integrate information from multiple cameras and to provide situational awareness. The visual memory is an object-oriented database that supports a variety of spatio-temporal queries [1, 3, 4]. The system detects and tracks people at ten frames per second, applies a ground plane constraint to estimate their 3-D positions, and records their positions and other information in the visual memory. A graphical interface allows users to construct security-related queries and view the results.

Figure 1 shows the system being used to monitor activity in a hallway. To detect people, the system differences live video frames with a background reference image to estimate regions of change. These regions are grouped into collections that are consistent with the size and shape of a person walking in the field of view. This process is repeated until all regions are either interpreted as part of a person, or dismissed if sizes and shapes of neighboring regions are inconsistent with such an interpretation. The user interface presents video surveillance observations from the visual memory in an interactive map display. Users interact with a map of the monitored region, and can pan, scroll and zoom to focus on a region of interest. Queries enable the display of both current positions of objects of interest and historical positions (i.e., paths) of particular objects. Users can also specify alarms to be generated when someone enters a particular region.

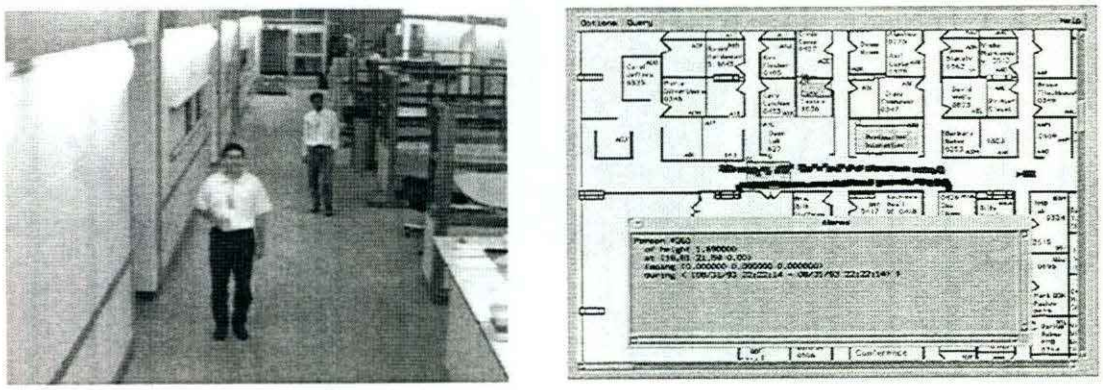


FIGURE 1. TI people tracking and position mapping system. The system tracks moving objects in video from security cameras, as at left. Square icons in the map-based interface at right represent location histories of humans at left. Pop-up window gives a detailed view of one observation record. Shaded rectangles in map are alarm regions.

## 2.2 Detection of People Carrying Boxes

After a theft has been reported in an area monitored by time-lapse video recorders, security staff typically review video tapes for the period during which the theft occurred. Tapes captured at building access points often reveal how the stolen material left the facility, or yield a short list of people who may have removed the material. However, the process of reviewing the tapes can be extremely tedious, especially when the period of interest spans many days.

In these experiments, TI demonstrated a system for screening security video tapes to identify segments that may contain images of people carrying boxes. This task was suggested by experts in physical security operations, who often look for boxes as part of theft investigation. The algorithm uses perceptual grouping techniques to identify collections of lines that may be projections of rectangular solids. If it finds a high-quality grouping of an appropriate size, it flags that video frame for inspection by a human and proceeds to the next frame. This relieves the investigator of the need to examine and interpret every frame.

The box detection algorithm was tested on 500 frames taken from time-lapse video sequences showing humans passing through a revolving door. Figure 2 shows a typical frame. The system exhibited a 93% detection rate at a false alarm rate of 13%. Details of the experiments and algorithm are given in a forthcoming paper [5].

## 2.3 Event Description for Video Indexing Using Motion Graphs

Future autonomous video surveillance systems will need to be able to classify motions and interactions of objects into events that are meaningful and important to security staff. TI has developed an Automatic Video Indexing (AVI) system ([2], figure 3), which performs event recognition on surveillance video tapes. As in the case of the box detection work described above, the immediate goal of this project was to assist human investigators in finding relevant segments of security video recordings; however, the event recogntion algorithm is general and can be applied to many video understanding tasks.

In the AVI system, object motions and interactions are described by a directed acyclic graph called a motion graph. Each node of the graph is an observation of an object, which is simply a region of change detected by image differencing. In the motion graph, each object is tracked and is linked to its predecessor and successor in time. Forks and joins in the graph represent complex interactions. For example, if a person enters a scene, puts down an objects and leaves, the graph will contain a chain of nodes representing the person, with a fork node whose successors are the continuation of the person track and a chain of observations of a stationary object.

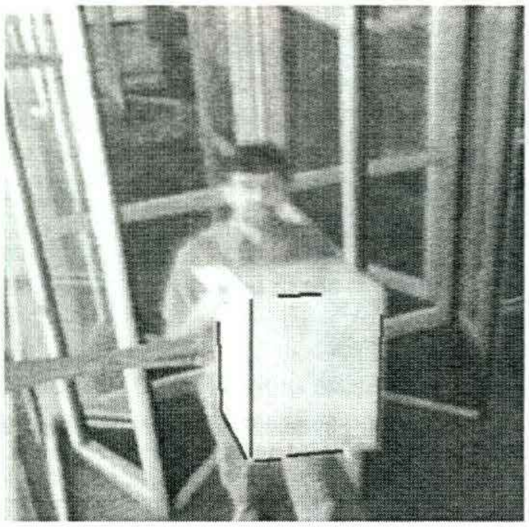


FIGURE 2. Box Detection in a security video image. Lines overlaid on the figure were identified by the detection algorithm as providing evidence for a box. In testing on 500 frames, the algorithm achieved a 93% detection rate at a false alarm rate of 13%.

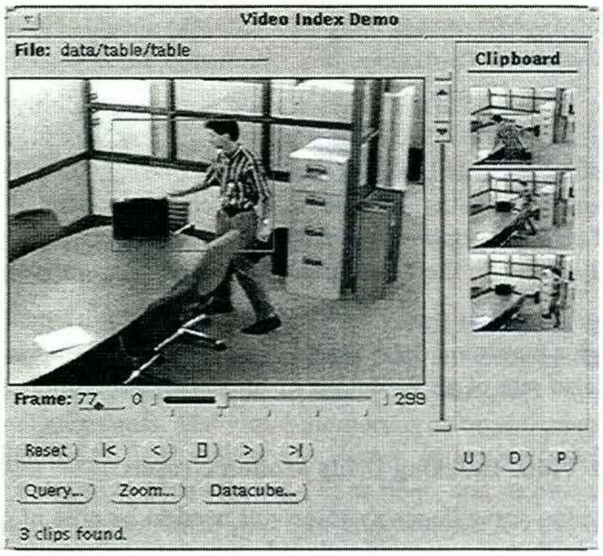


FIGURE 3. The Automatic Video Indexing system detects significant events in security videotapes. Above, the system identifies an instance of removal of an object from the scene, defined as disappearance of a stationary object (the briefcase) while in contact with a moving object (the human). An overlaid box highlights the objects involved in the removal event. Other removals from other points in the videotape are shown on the clipboard at right.

Figure 4 provides an example of how the motion graph is constructed and interpreted. Vertical lines represent frames, here compressed to 1D to simplify the diagram. Each node of the graph consists of an observation of an object at a particular place in a particular frame. Links between objects constitute hypotheses about identity, constructed by tracking objects over time. Nodes with two or more successors correspond to places where the tracking algorithm determined that an object split into multiple objects. Similarly, nodes with multiple predecessors result from the merging of multiple objects. Events are defined

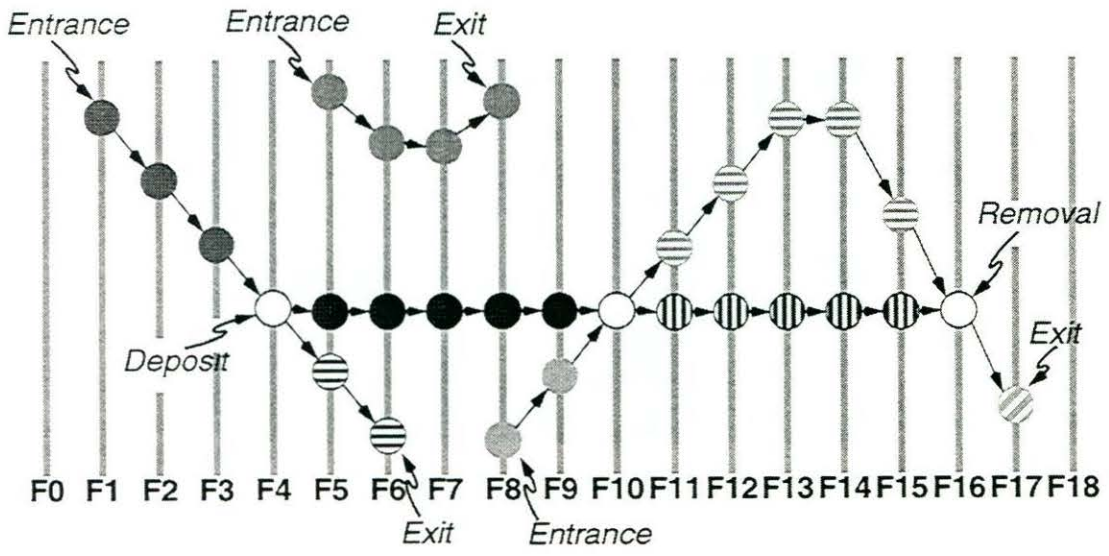

Entrance
Entrance
Exit
Removal
Deposit
Exit
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16 F17 F18
Exit
Entrance

FIGURE 4. Automatic Video Indexing motion graph example. The motion graph captures the motions and interactions of objects in the scene. Predicates on the graph correspond to event classes such as Entrance, Exit, Removal, et cetera.

by characteristic structures in the graph, as shown in the figure. For example, a chain of observations beginning near the edge of the image constitutes an *Entrance* event, a moving object that splits into a moving object and a stationary one constitutes a *Deposit* event, and so on.

The AVI video indexing capability allows extremely rapid access to significant events in long time-lapse security videos. The algorithms were demonstrated live at the 1996 Image Understanding Workshop and have been used with both visible and infrared video data.

## 3. Advanced Proof-of-Concept Demonstrations

In autonomous video surveillance research, TI is integrating the capabilities described in the preceding section to produce a series of proof-of-concept demonstrations. Together these demonstrations illustrate how physical security operations of the future will use autonomous video surveillance systems. Key challenges are computing event graphs on-line at ten frames per second, and devising event graph analysis methods to exploit contextual information in useful office surveillance scenarios.

This research is focussed on three overall surveillance scenarios, for monitoring hallway, office, and building perimeter areas. In each area, a camera provides live video data of scenes in the field of view, while the AVS system monitors the video to analyze events and signal alarms.

### 3.1 Hallway Surveillance

In this scenario, suggested previously in Figure 1, the autonomous video surveillance system detects and tracks people as they walk in office building hallways. Alarms are interactively defined for conditions such as when someone loiters in a specified area or enters a particular office. This visual assessment provides information to augment other security system data, such as biometric access control information at building entrance points.

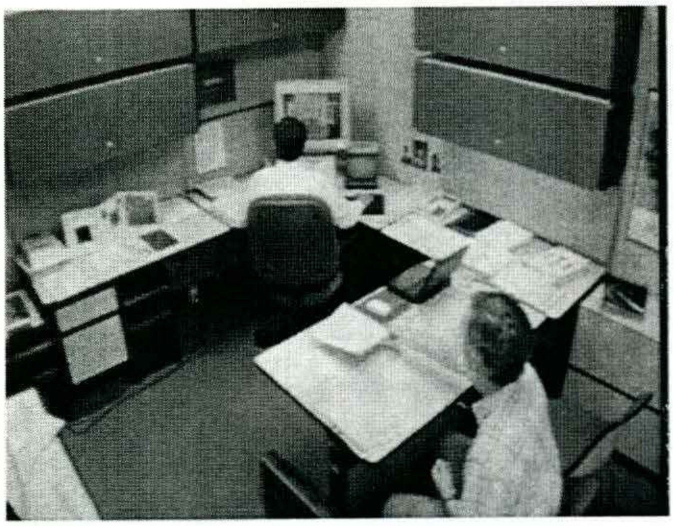


FIGURE 5. Office Surveillance Scenario. Video event recognition can be used to detect unauthorized access to documents, file drawers, computers, et cetera.

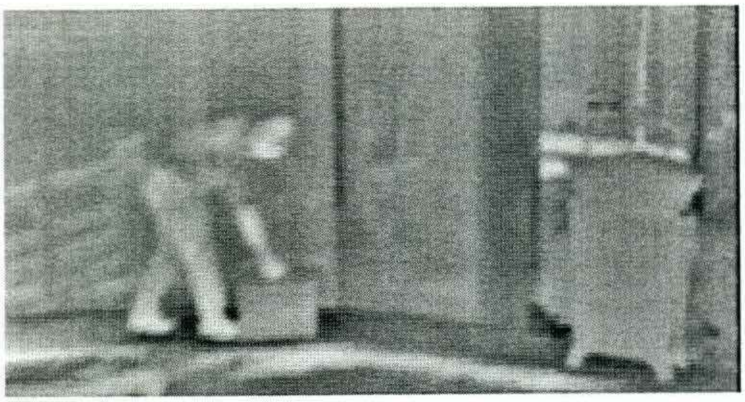


FIGURE 6. Perimeter Surveillance scenario. The AVI event recognition algorithms can be used with low-cost infrared cameras to detect security threats in total darkness.

### 3.2 Office Surveillance

Inside individual offices, the autonomous video surveillance system will maintain a situational awareness record of events and signal alarms for a variety of specified conditions. For example, an alarm may be specified for events in which someone enters the office and places a briefcase on the desk, but not if the person leaves a document on the desk. Using contextual information such as time of day and access control identification, the system will report other alarm conditions that are functions of the number of people in the room and what they do. A representative office surveillance scene is shown in Figure 5.

### 3.3 Perimeter Surveillance

For perimeter monitoring scenarios, a TI NightSight infrared camera will provide video data for the surveillance system to monitor areas outside the building at night. For example, the system could monitor a building entrance and signal an alarm if someone walks by and leaves a box outside the door (e.g., as illustrated in Figure 6), but not if someone loiters without leaving a box.

The ability to reliably discriminate significant and insignificant events is important to reduce false alarms for physical security applications, and poses many challenges for computer vision research to advance the state of the art.

## 4. Acknowledgements

The autonomous video surveillance capabilities described in this paper represent the research of many members of the technical staff at TI over the past five years. Among them the primary contributors are Tom Bannon, Frank Brill, Jon Courtney, Chris Kellogg, and Kashi Rao.

## 5. Bibliography

[1] Bannon, T., "Visual Memory Prototype Demonstration", TI Internal Technical Report CSL-ITR-93-01-25, Jan. 1993.

[2] Courtney, J. "Automatic video indexing via object motion analysis", TI Internal Technical Report, to appear in Pattern Recognition.

[3] Flinchbaugh, B., and T. Bannon, "Autonomous Scene Monitoring System", Proc. 10th Annual Joint Government-Industry Security Technology Symposium, American Defense Preparedness Association, June 1994.

[4] Kellogg, C., "Visual Memory", TI Internal Technical Report, also MIT Media Lab MS thesis, 1993.

[5] Rao, K., and P. Sarwal, "A computer vision system to detect 3-D rectangular objects", TI Internal Technical Report. Accepted for IEEE Workshop on Applications of Computer Vision (WACV 96).

[6] Vigil, J., "An evaluation of exterior video motion detection systems. Volume 1: Intrusion detection tests. Volume 2: Nuisance alarm results", Sandia Reports SAND92-0108/1, SAND92-0108/2, Sandia National Laboratories, 1992.

LIBRARY OF CONGRESS
ONLINE CATALOG

Search Navigation ▾ ‹ 9 of 11 ›

BOOK

# Emerging applications of computer vision : 25th AIPR Workshop, 16-18 ...

Full Record **MARC Tags**

| | | |
|---|---|---|
| 000 | 01814cam a2200409 a 4500 | |
| 001 | 1638062 | |
| 005 | 20160811120013.0 | |
| 008 | 961121s1997 waua b 101 0 eng d | |
| 906 | __ | \|**a** 7 \|**b** cbc \|**c** copycat \|**d** 2 \|**e** opcn \|**f** 19 \|**g** y-gencatlg |
| 925 | 0_ | \|**a** acquire \|**b** 1 shelf copy \|**x** policy default |
| 955 | __ | \|**a** pb20 10-21-97 to cat.; jf00 10-23-97; jf09 to sub 10-23-97; jf08 10-24-97 to SL; je25 11-12-97 to ddc;aa05 11-14-97 |
| 010 | __ | \|**a** 96071104 |
| 020 | __ | \|**a** 0819423661 |
| 035 | __ | \|**9** (DLC) 96071104 |
| 035 | __ | \|**a** (OCoLC)36514905 |
| 040 | __ | \|**a** MoKL \|**c** MoKL \|**d** DLC |
| 042 | __ | \|**a** lccopycat |
| 050 | 04 | \|**a** TA1634 \|**b** .A37 1996a |
| 082 | 00 | \|**a** 006.3 \|**2** 21 |
| 111 | 2_ | \|**a** AIPR Workshop \|**n** (25th : \|**d** 1996 : \|**c** Washington, D.C.) |
| 245 | 10 | \|**a** Emerging applications of computer vision : \|**b** 25th AIPR Workshop, 16-18 October 1996, Washington, D.C. / \|**c** David Schaefer, Elmer F. Williams, chairs/editors ; sponsored by SPIE--the International Society for Optical Engineering, AIPR Executive Committee. |
| 260 | __ | \|**a** Bellingham, Wash. : \|**b** SPIE, \|**c** c1997. |
| 300 | __ | \|**a** x, 302 p. : \|**b** ill. ; \|**c** 28 cm. |
| 490 | 1_ | \|**a** Proceedings / SPIE--the International Society for Optical Engineering ; \|**v** v. 2962 |
| 504 | __ | \|**a** Includes bibliographical references and index. |
| 650 | _0 | \|**a** Computer vision \|**x** Congresses. |
| 650 | _0 | \|**a** Image processing \|**x** Databases \|**x** Congresses. |
| 650 | _0 | \|**a** Database management \|**x** Congresses. |
| 650 | _0 | \|**a** Optical storage devices \|**x** Congresses. |
| 700 | 1_ | \|**a** Schaefer, David. |
| 700 | 1_ | \|**a** Williams, Elmer F. |
| 710 | 2_ | \|**a** Society of Photo-optical Instrumentation Engineers. |
| 710 | 2_ | \|**a** AIPR Executive Committee. |
| 830 | _0 | \|**a** Proceedings of SPIE--the International Society for Optical Engineering ; \|**v** v. 2962. |
| 920 | __ | \|**a** ** LC HAS REQ'D # OF SHELF COPIES ** |
| 922 | __ | \|**a** co |
| 991 | __ | \|**b** c-GenColl \|**h** TA1634 \|**i** .A37 1996a \|**t** Copy 1 \|**w** BOOKS |

Request this Item LC Find It

## Where to Request

| | |
|---|---|
| CALL NUMBER | TA1634 .A37 1996a FT MEADE<br>Copy 2 |

Ex. K Page 1 of 1

Barton MIT Libraries' Catalog

MIT Libraries

**Search Full Catalog:**
- Basic
- Advanced

**Search only for:**
- Conferences
- E-resources
- Journals
- MIT Theses
- Reserves
- more...

- Your Account
- Help with Your Account
- Your Bookshelf
- Previous Searches

Ask Us! Other Catalogs Help

## Full Record

**Permalink for this record:** http://library.mit.edu/item/000819773

**Results List** | **Add to Bookshelf** | **Save/Email**

Choose format: **Standard** | **Citation** | **MARC tags**

**Record 1 out of 1**

| | |
|---|---|
| **FMT** | BK |
| **LDR** | 01492nam 2200325Ia 45r0 |
| **003** | MCM |
| **005** | 20010609092045.0 |
| **006** | m d |
| **007** | cr un- |
| **008** | 970310s1997 waua b 101 0 eng d |
| **020** | \|a 0819423661 |
| **035** | \|a MITb10819773 |
| **035** | \|a (OCoLC)36514905 |
| **040** | \|a LHL \|c LHL \|d MYG |
| **090** | \|a TA1634 \|b .A37 1996 |
| **1112** | \|a AIPR Workshop \|n (25th : \|d 1996 : \|c Washington, D.C.) |
| **24510** | \|a Emerging applications of computer vision : \|b 25th AIPR Workshop, 16-18 October 1996, Washington, D.C. / \|c David Schaefer, Elmer F. Williams, chairs/editors ; sponsored by SPIE--the International Society for Optical Engineering, AIPR Executive Committee ; published by SPIE--the International Society for Optical Engineering. |
| **260** | \|a Bellingham, Washington : \|b SPIE, \|c c1997. |
| **300** | \|a x, 302 p. : \|b ill. ; \|c 28 cm. |
| **4901** | \|a Proceedings / SPIE--the International Society for Optical Engineering ; \|v v. 2962 |
| **530** | \|a Also available online via the World Wide Web; access restricted to licensed sites/users. |
| **504** | \|a Includes bibliographical references and author index. |
| **650 0** | \|a Image processing \|x Databases \|v Congresses. |
| **650 0** | \|a Image processing \|x Digital techniques \|v Congresses. |
| **650 0** | \|a Database management \|v Congresses. |
| **650 0** | \|a Optical storage devices \|v Congresses. |
| **7001** | \|a Schaefer, David. |
| **7001** | \|a Williams, Elmer F. |
| **7102** | \|a Society of Photo-optical Instrumentation Engineers. |
| **7102** | \|a AIPR Executive Committee. |
| **85641** | \|u http://spiedigitallibrary.org/proceedings/resource/2/psisdg/2962/1 |
| **830 0** | \|a Proceedings of SPIE--the International Society for Optical Engineering ; \|v v. 2962. |
| **CAT** | \|a CONV \|b 00 \|c 20010620 \|l MIT01 \|h 1548 |
| **CAT** | \|a spieurl \|b 00 \|c 20060331 \|l MIT01 \|h 1607 |
| **CAT** | \|a lti0904 \|b 00 \|c 20090523 \|l MIT01 \|h 2259 |
| **CAT** | \|a EDWARDSJ \|b 00 \|c 20110408 \|l MIT01 \|h 1056 |
| **95641** | \|u http://libraries.mit.edu/get/spie |
| **049** | \|a MYGG |
| **910** | \|a tn970627 |
| **PST0** | \|0 Z30 \|1 000819773000010 \|b LSA \|c OCC \|o BOOK \|d 15 \|y 00000 \|f N \|r MIT60-000781407 \|n 0 \|h TA1634.A37 1996 \|a MCM \|3 Book \|4 Library Storage Annex \|5 Off Campus Collection \|6 OCC 60 \|p Avail |

```
LDR      nx  a22     1i 4500
008    1104082u    8   4001uueng0000000
8528   |b NET |z MIT Access Only |h **See URL(s)
004    000819773
LDR      nx   22     zn 4500
008    0106230u    0   4   uueng1
004    000819773
8520   |a MCM |b LSA |c OCC |h TA1634.A37 1996
001    000819773
SFX41  |s 0-0-0-8-1-9-7-7-3 |l MIT01 |9 001 |z button~~~ |p Avail |f 001 |a Click button for available online volumes
URL    |u http://spiedigitallibrary.org/proceedings/resource/2/psisdg/2962/1 |9 001
U564   |u http://spiedigitallibrary.org/proceedings/resource/2/psisdg/2962/1 |9 001
LU564  |u http://spiedigitallibrary.org/proceedings/resource/2/psisdg/2962/1 |f 001
SYS    000819773
```

## Basic Search of Full Catalog

Search type:
Keyword
Title begins with...
Title Keyword
Author (last name first)
Author Keyword
Call Number begins with...
----- Scroll down for more choices -----

Search for:

Search

Barton Questions: Ask Us! | Contact Us
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA

-- Quick Links --