

---


Electronic Theses and Dissertations, 2004-2019

---

2013

## Scene Understanding For Real Time Processing Of Queries Over Big Data Streaming Video

Alexander Aved  
*University of Central Florida*

 Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)  
Find similar works at: <https://stars.library.ucf.edu/etd>  
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact [STARS@ucf.edu](mailto:STARS@ucf.edu).

---

### STARS Citation

Aved, Alexander, "Scene Understanding For Real Time Processing Of Queries Over Big Data Streaming Video" (2013). *Electronic Theses and Dissertations, 2004-2019*. 2511.  
<https://stars.library.ucf.edu/etd/2511>

# SCENE UNDERSTANDING FOR REAL TIME PROCESSING OF QUERIES OVER BIG DATA STREAMING VIDEO

by

ALEXANDER J. AVED  
B.A., Anderson University, 1999  
M.S., Ball State University, 2001

A dissertation submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy  
in the Department of Electrical Engineering and Computer Science  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Spring Term  
2013

Major Professor: Kien A. Hua

© 2013 Alexander J. Aved

## ABSTRACT

With heightened security concerns across the globe and the increasing need to monitor, preserve and protect infrastructure and public spaces to ensure proper operation, quality assurance and safety, numerous video cameras have been deployed. Accordingly, they also need to be monitored effectively and efficiently. However, relying on human operators to constantly monitor all the video streams is not scalable or cost effective. Humans can become subjective, fatigued, even exhibit bias and it is difficult to maintain high levels of vigilance when capturing, searching and recognizing events that occur infrequently or in isolation.

These limitations are addressed in the *Live Video Database Management System (LVDBMS)*, a framework for managing and processing live motion imagery data. It enables rapid development of video surveillance software much like traditional database applications are developed today. Such developed video stream processing applications and ad hoc queries are able to “reuse” advanced image processing techniques that have been developed. This results in lower software development and maintenance costs. Furthermore, the LVDBMS can be intensively tested to ensure consistent quality across all associated video database applications. Its intrinsic privacy framework facilitates a formalized approach to the specification and enforcement of verifiable privacy policies. This is an important step towards enabling a general privacy certification for video surveillance systems by leveraging a standardized privacy specification language.

With the potential to impact many important fields ranging from security and assembly line monitoring to wildlife studies and the environment, the broader impact of this work is clear. The privacy framework protects the general public from abusive use of surveillance technology;

success in addressing the “trust” issue will enable many new surveillance-related applications. Although this research focuses on video surveillance, the proposed framework has the potential to support many video-based analytical applications.

## **ACKNOWLEDGMENTS**

I am grateful to my advisor, Dr. Kien A. Hua, for his dedication, commitment and encouragement. He always found time to discuss ideas, research plans and provide helpful advice. Furthermore I would like to thank my committee members; Dr. Foroosh, Dr. Ni and Dr. Zou.

I think my wife Jessica for her support over the years and allowing me to pursue my dreams and research endeavors, as well as my family and friends, and colleagues in the Data Systems Lab and the Office of Research & Commercialization, for their unwavering support.

I am grateful to the University of Central Florida for providing the resources and facilities for me to conduct my research, and for the helpful CECS and EECS faculty and staff who have provided invaluable help and advice throughout my years as a student.

# TABLE OF CONTENTS

LIST OF FIGURES .....	xi
LIST OF TABLES .....	xv
CHAPTER 1: INTRODUCTION .....	1
Motivations.....	1
Overview of Dissertation .....	2
CHAPTER 2: MULTIMEDIA DATABASES AND LIVE VIDEO COMPUTING .....	3
Introduction .....	4
Fundamental Concepts and Components of Multimedia Database Systems .....	5
Multimedia Data Representation – Data and Information.....	8
Digital Sampling and Reconstruction.....	10
Data Representation and Features .....	10
Feature Extraction.....	13
Background Subtraction .....	15
Segmentation of Image Regions.....	19
Tracking Objects within a Single Camera.....	21
Distributed Object Tracking with Multiple Cameras .....	22
Supervised and Unsupervised Learning .....	24
Clustering.....	26

A Brief Review of Multiple-Instance Learning.....	27
Shot Boundary Detection and Representative Image Selection .....	30
Multimedia Data Representation for Indexing .....	33
Multimedia Indexing Storage and Retrieval.....	35
The Semantic Gap .....	41
Content-based Image Retrieval .....	42
Research Video Database Management Systems.....	45
Introduction to Live Video Computing and Big Data.....	46
Basic Premises of Live Video Computing .....	46
Live Video Computing is Big Data .....	51
Summary .....	52
CHAPTER 3: INTRODUCTION TO THE LVDBMS .....	54
LVDBMS Architecture .....	56
LVDBMS Data Model .....	60
Introduction to the LVQL Query Language.....	63
Summary .....	66
CHAPTER 4: AN INFORMATICS-BASED APPROACH TO OBJECT TRACKING .....	68
Introduction .....	68
Previous Multi-Camera Object Tracking Work.....	69



Cross-Camera Object Tracking in the LVDBMS .....	71
LVDBMS Cross-Camera Tracking Implementation.....	73
Performance Evaluation .....	78
Evaluation Scenario Setup.....	80
Evaluation Based Upon Relative Quality .....	81
Performance of Cross-Camera Tracking .....	82
Inclusion Distance Threshold .....	87
Conclusions and Comments .....	88
Summary .....	89
CHAPTER 5: MANAGING LIVE VIDEO DATA WITH PRIVACY PROTECTION .....	91
Introduction .....	91
Background .....	94
Privacy Filter Framework Objectives .....	97
Scope and Assumptions of Privacy Preservation and the LVDBMS .....	98
Overview of Privacy Framework .....	99
Defining Privacy Filters with the Privacy Specification Language .....	102
Combinations of Privacy Filters .....	106
Formal Specification of the Privacy Filter Model .....	106
Performance Evaluation .....	107

Privacy Filter Effectiveness.....	109
Object Tracking Effectiveness.....	110
Holistic Demonstration of a Privacy Filter.....	114
Summary .....	117
CHAPTER 6: EFFICIENT QUERY PROCESSING.....	118
Introduction .....	119
Background .....	120
Query Processing.....	121
Overview .....	121
The Query Parser and Translator.....	122
The Query Optimizer.....	124
LVDBMS Query Optimization .....	127
LVDBMS Query Execution Environment .....	130
Cost Estimation.....	131
Experimental Study .....	133
Summary .....	137
CHAPTER 7: LVDBMS PROTOTYPE .....	139
Introduction .....	139
Prototype System Architecture.....	139

Camera Layer .....	139
Spatial Processing Layer .....	141
Stream Processing Layer .....	143
Client Layer .....	144
Experimental Study .....	145
Query Processing Performance .....	145
Summary .....	147
CHAPTER 8: CONCLUSIONS .....	150
Summary of Contributions .....	150
Future Work .....	151
LIST OF REFERENCES .....	153

## LIST OF FIGURES

Figure 1. A typical database architecture (left) vs. a multimedia database (right). ....	3
Figure 2. Multimedia data (images) represented as points in a 2-dimensional feature space. ....	7
Figure 3. An object (yellow car) whose pixels are segmented from the image background. ....	7
Figure 4. Complete image (left), and magnified view of extracted foreground pixels (right). ....	16
Figure 5. A traffic scene (left), and the corresponding foreground mask (right).....	18
Figure 6. Typical processing steps implemented in a background subtraction algorithm.....	20
Figure 7. Image segmentation example, based upon graph-based representation (Felzenszwalb & Huttenlocher, 2004) © 2004 Kluwer Academic Publishers. ....	20
Figure 8. Another graph cut segmentation image segmentation example (Boykov & Funka-Lea, 2006) © 2006 Springer Science + Business Media, LLC.....	21
Figure 9. Cameras with overlapping (left) and non-overlapping (right) fields of view. ....	24
Figure 10. Camera network calibration utilizing a robot with a pattern (Rekleitis et al., 2006). © 2006 Elsevier B.V.....	24
Figure 11. Essential clustering steps; information flow.....	26
Figure 12. Hierarchical structure of a video depicting scenes, shots and frames. ....	31
Figure 13. Example of an R-Tree spatial index. ....	38
Figure 14. Representative architecture of a typical CBIR system. ....	44
Figure 15. Siloed approach to camera/stream processing application development. ....	48
Figure 16. Applications built as information silos utilizing middleware and application-specific data adapters.....	48
Figure 17. LVC approach showing common image processor and query interface. ....	49

Figure 18. LVC stream data model contrasted with relational record and disk data model.....	50
Figure 19. LVC exhibits the three popular attributes of big data; variety, velocity and volume. ....	51
Figure 20. Logical 4-tier architecture of the LVDBMS prototype and major components of the framework encapsulated in each tier.....	57
Figure 21. LVDBMS prototype, illustrating query, subqueries and results. ....	57
Figure 22. Example images of the LVDBMS; the appearances of some objects have been redacted. ....	58
Figure 23. Depiction of the data flow in the LVDBMS, from frames to query result.....	62
Figure 24. A simplification of the LVQL grammar, including privacy <i>view definition language</i> (VDL) productions. ....	65
Figure 25. Illustration depicting dynamic and static objects. ....	65
Figure 26. Illustration of an object moving about a scene and corresponding instances, in a feature space.....	67
Figure 27. (left) An object moves about in a straight path, and (right) an object moves along a random path. ....	70
Figure 28. Illustration of feature vectors calculated based upon an object's appearance in every $n^{\text{th}}$ frame. ....	72
Figure 29. Metadata structures implemented in the LVDBMS to facilitate cross-camera tracking and queries. ....	75
Figure 30. Object metadata structure showing two objects, <i>1.12</i> and <i>0.37</i> , which have been merged.....	75
Figure 31. Sample frames from CAVIAR dataset; <i>walk3</i> (left) and <i>OneShopOneWait2Cor</i> (right). ....	79

Figure 32. Sample video frames from two cameras, one in a room and the second outside the room ( <i>hallway sequence</i> ). .....	79
Figure 33. Relative Quality for <i>OneShopOneWait2cor</i> ; (a) varying $\beta$ with fixed $\alpha=0.5$ , (b) an enlarged view of a portion of (a), and (c) varying $\alpha$ with fixed $\beta=1.2$ . .....	85
Figure 34. <i>Normalized distance</i> (ND) and <i>closest point distance</i> (CPD) plotted over time.....	86
Figure 35. Tracking results for <i>hallway sequence</i> videos; where $\alpha=1$ and various $\beta$ . .....	87
Figure 36. Illustration of instances corresponding to two image sequences with bags of cardinality 4 and $k=3$ .....	90
Figure 37. Example illustrating the LVDBMS deployed in a traffic management center. ....	93
Figure 38. Comparison of cascading relational database views (left) vs. cascading privacy filters (right). .....	100
Figure 39. Video stream illustrating a privacy filter with a Gaussian blurred MBR.....	100
Figure 40. Query targets: object <i>D121</i> satisfies the query condition, <i>D102</i> does not. ....	103
Figure 41. The PSL extension of LVQL; colored text illustrates user-supplied values. ....	105
Figure 42. Privacy filter examples from the first (left) and third (right) video series. ....	109
Figure 43. Cross-camera query evaluation accuracy for video sequence #44.....	112
Figure 44. Cross-camera query evaluation accuracy for video sequence #46.....	112
Figure 45. Holistic privacy filter example showing a camera, a view with associated privacy filter and two video consumers. ....	115
Figure 46. Imagery as observed by the TMC operator. ....	116
Figure 47. Video stream as observed through the view; note the blur effect. ....	116
Figure 48. Query lifecycle; from the inception of a query to results delivered to the issuer.....	124
Figure 49. Query transformation steps, from inception to execution. ....	125

Figure 50. Transformations undergone by a query; from query to subquery to results.....	128
Figure 51. Before and after illustrations of a subquery merged into a query group. ....	129
Figure 52. Example depicting four query groups in the LVDBMS query execution engine. ....	132
Figure 53. Example video frame showing a busy road and a static object (the blue rectangle).134	
Figure 54. Evaluation 1: queries utilized for testing the query optimizer. ....	134
Figure 55. Evaluation 2: complex queries for optimization evaluations. ....	134
Figure 56. The lower three LVDBMS tiers, showing major components. ....	138
Figure 57. Camera adapter, simulating a video stream from a pre-recorded video.....	141
Figure 58. The camera server service, which runs in the spatial processing layer. ....	142
Figure 59. The query processing service, which runs in the stream processing layer.....	143
Figure 60. The LVDBMS user client, showing a connection to a camera server and three video streams and two views. ....	144
Figure 61. Evaluation costs for a selection of queries in milliseconds; plotted at one-second intervals.....	148
Figure 62. Screen capture of Visual Studio 2012 IDE depicting a dependency graph of LVDBMS assemblies. ....	149
Figure 63. Illustration of potential future works pertaining to LVC and the LVDBMS. ....	152

## LIST OF TABLES

Table 1: Modalities of unstructured data by dimensionality .....	11
Table 2: Comparison of LVC and traditional DBMS concepts .....	51
Table 3: Comparison classes of LVC objects .....	61
Table 4. CAVIAR video <i>OneShopOneWait2cor</i> .....	83
Table 5. System parameters for <i>Walkers</i> evaluations .....	84
Table 6. Walkers–closest point distance .....	84
Table 7. Walkers–normalized distance .....	84
Table 8. Normalized distance for <i>hallway sequence</i> videos .....	87
Table 9. Privacy filter attribute <i>target</i> .....	104
Table 10. Privacy filter attribute <i>temporal scope</i> .....	104
Table 11. Privacy filter attribute <i>object scope</i> .....	104
Table 12. Privacy filters corresponding to scenario in Figure 45 .....	113
Table 13. Query accuracy evaluation results .....	114
Table 14. Evaluation costs of various operators and operands .....	132
Table 15. Evaluation 1, performance counters without optimization .....	136
Table 16. Evaluation 1, performance counters with optimization .....	136
Table 17. Evaluation 1, cost efficiencies gained.....	137
Table 18. Evaluation 2, cost efficiencies gained.....	137
Table 19. Average query evaluation in milliseconds of CPU time, by video.....	148



# CHAPTER 1: INTRODUCTION

## Motivations

Many factors contribute to the proliferation of camera networks, including decreased hardware costs and heightened concerns regarding security and safety. In order to be positioned to quickly react to events observed in these live video streams, they must be actively monitored. However, due to the volume and velocity of live data produced by the multitude of cameras, it is difficult, if not impossible, to review all the video streams without some degree of automation. When considering such a solution for such an environment, two factors are of primary importance, (1) the application of intelligent algorithms to sift through the multitudes of video streams to discern which scenes are of potential interest, and (2) usability; the ease of use, efficiency and satisfaction with which operators, in a variety of roles, can interact with and use the system in order to meet objectives.

Video streams consume significant network bandwidth, and network capacity bottlenecks need to be mitigated. Therefore, applicable algorithms (and their implementation) must be distributed in nature in order to scale with the number of video streams and not overload portions of the network.

With the combination of technological advances such as better algorithms that can recognize actions occurring in a scene, face recognition and tracking, inexpensive and abundant data storage, and expanding areas of surveillance coverage, mean that more of our lives will be observed, analyzed and digitally preserved. Consequently, privacy is important. Camera network implementers must ensure that socially accepted standards (which can vary from culture to culture) are upheld, and that data captured for one purpose (such as ensuring security or safety) is

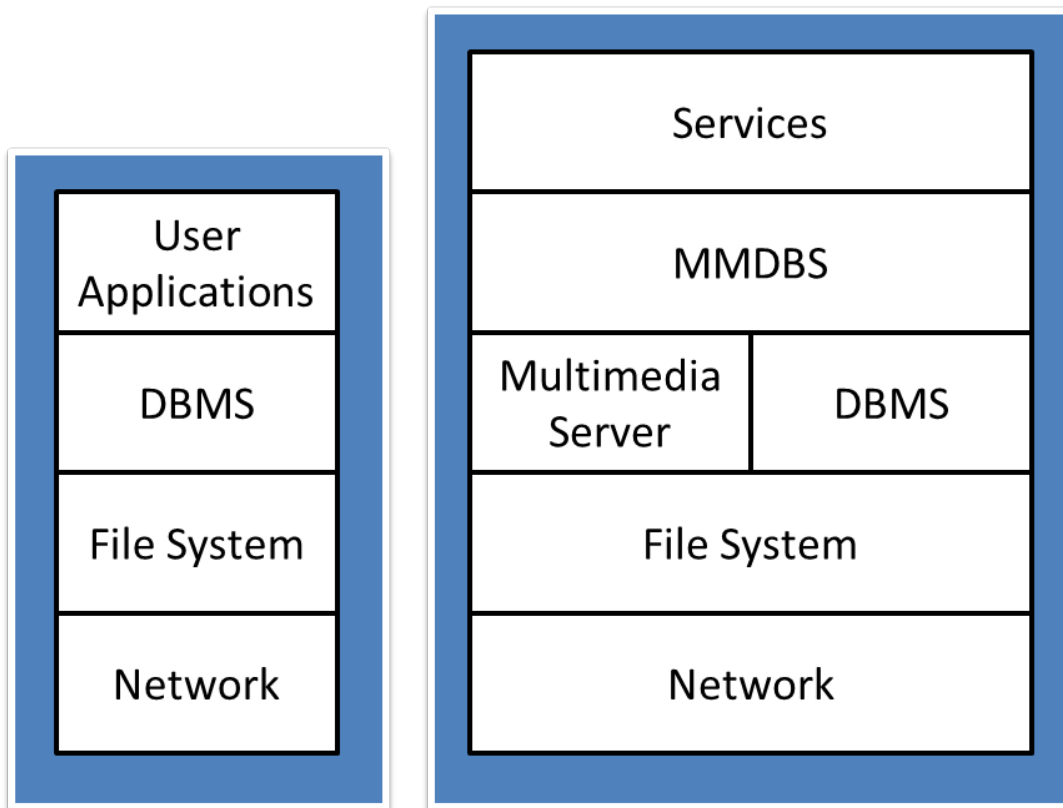
not later used for another. A viable solution that is socially acceptable must consider various privacy aspects, and have facilities to define and implement privacy policies so that people can be confident that their stored appearance will be utilized consistently and ethically; both when the data is captured and in the future.

### Overview of Dissertation

This dissertation is primarily organized into six major content areas. Firstly, a review of multimedia database systems and enabling concepts are presented. Secondly, the LVDBMS, a *Live Video Computing (LVC)* prototype database is introduced along with its data model and query language. Thirdly, an approach to modeling objects and tracking them across video streams is presented. A technique based upon bipartite graph matching is presented. Fourthly, a context-aware privacy preserving framework is presented, which protects the privacy of individuals observed in video streams. Fifthly, a run-time query optimizer and query execution environment is presented. Efficient query processing is a key element to effective resource utilization. Sixthly, the LVDBMS prototype is presented, including both implementation and performance details. In the remainder of this manuscript the terms stream, video stream and camera are used interchangeably.

## CHAPTER 2: MULTIMEDIA DATABASES AND LIVE VIDEO COMPUTING

This chapter discusses fundamental components that comprise *Multimedia Database Systems (MMDBS)*. Later sections build upon this foundation, extending it to the LVC platform. The LVC platform combines hardware, networking infrastructure and a software framework into a solution which allows operators to interact with numerous live video feeds in real time.



**Figure 1. A typical database architecture (left) vs. a multimedia database (right).**

## Introduction

Advances in imaging sensors, networking, processing, data storage and algorithms all contribute to ever increasing quantities of multimedia information. This encompasses audio and video, recorded and live, as well as still imagery and text. Multimedia information processing includes the generation, representation, storage and retrieval, processing, communication and presentation of multimedia content, as well as that of related metadata (Nwosu, Thuraisingham, & Berra, 1996). In order to manage these massive quantities of data, MMDBS have been created. A *Database Management System (DBMS)* is software that allows users to store and use data in an abstract way, without having to consider how the data is physically stored and managed on a storage volume (Ullman, 1981). MMDBS are DBMS that facilitate the storage and retrieval of multimedia data, to include the transmission, indexing, querying and manipulation of said data.

Figure 1 compares a traditional DBMS with a MMDBS. A typical DBMS implementation, Figure 1 (left), supports business applications by persisting application state, resolving queries, and facilitating transactions to mitigate concurrency errors. Figure 1 (right) illustrates a MMDBS, which can utilize a traditional DBMS to manage metadata and indices, but also encompasses additional technologies and services not typically present in DBMSs which include: video on demand, document management and imaging, spatial data, specialized query languages, face recognition and relevance feedback, to name a few. Because multimedia content, and video in particular, can be quite large and its communication bandwidth intensive, MMDBS are often paired with specialized communication frameworks, such as the HeRO protocol discussed in (Tantaoui, Hua, & Do, 2004), in order to provide content delivery to a multitude of concurrent users without overwhelming the physical communication medium.

## Fundamental Concepts and Components of Multimedia Database Systems

Early database systems were designed solely to efficiently manage the storage, retrieval and querying of alphanumeric data (Date, 1977). Advances in microprocessors, storage device capacity and imaging sensors led to vast quantities of pictorial data such as medical images, satellite imagery, and topographical maps, to name a few. With this data arose the problem of how to manage it in an efficient and structured way. Early approaches were based upon metadata that was generated manually by adding textual annotations to digital content (Blaser, 1979; N. Chang & Fu, 1980; N. S. Chang & Fu, 1980; S. K. Chang & Kunil, 1981; S. K. Chang, Yan, Dimitroff, & Arndt, 1988). As computer technology advanced, so did the content: pictures and discrete objects were replaced by video: sequences of still images temporarily correlated with audio. Also changing was the acquisition of metadata. It could be generated as the content was created (recording date and time stamps), generated automatically during the editing phase (pertaining to compositional information as the content is spliced together), or added post-production (bibliographic information, etc). A different post-production method is based upon extracting representative features automatically, called feature extraction. This approach is applicable to both the visual and audio components of video.

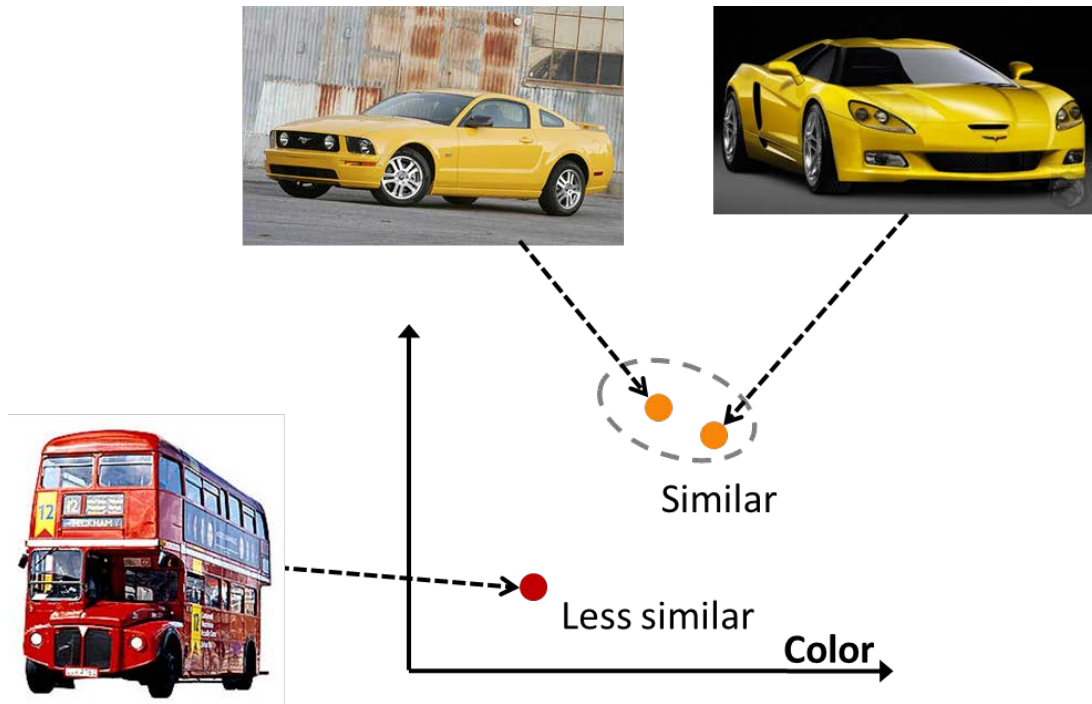
A MMDBS must be architected with the flexible to support a variety of applications. One such application is *Video on Demand (VOD)*. In VOD, users can browse a large selection of videos (for example, in a hotel setting these could be entertaining movies, in a corporate setting they could be training videos). The collection of videos are typically stored on and distributed from a centralized server. Browsing and search is conducted based upon the metadata associated with the videos, and the videos are consumed as a single unit. That is, the user will watch the

content from start to finish, though they may pause or fast forward or perform other similar linear operations (Barbieri, Mekenkamp, Ceccarelli, & Nesvadba, 2001).

In addition to VOD, interactive video is another application of MMDBS, serving video scenes in a non-linear fashion. Examples of such a service include educational videos, where a sequence of short clips pertaining to a particular topic may be combined, reordered, and viewed together as one video clip. Other applications include stock-shot (Turner, 1990) and browsing, indexing and searching video archives (Dimitrova et al., 2002).

MMDBS frameworks typical consist of three primary components, or phases (Z. Zhang & Zhang, 2008). The first entails representing the raw multimedia data as a point in an abstract,  $n$ -dimensional space termed a feature space, where  $n$  is the number of features that describes the data item. The process of representing the data as a point in the feature space is called feature extraction. Similar items should be grouped together (e.g., see Figure 2), thus, the quality of the feature selection and extraction methods affect the grouping and compactness of the data points. The compactness of the data in the feature space can have ramifications pertaining to the effectiveness of retrieval (e.g., *k-nearest neighbor* ( $k$ -NN) and classification (e.g., the application of support vector machines).

The second component of the framework is knowledge representation. A feature represents a measurable property of the multimedia data item (e.g., the number of red pixels in an image), and are typically represented as numeric data, though they can be a string or also a graph representation. Numeric features are usually chosen, as they can be operated upon mathematically. Discriminative features should be chosen, and the effectiveness by which the multimedia data may be represented by the selected features will have a significant impact on the performance of the MMDBS.



**Figure 2. Multimedia data (images) represented as points in a 2-dimensional feature space.**



**Figure 3. An object (yellow car) whose pixels are segmented from the image background.**

The third framework component performs some type of analysis or retrieval on the multimedia data that is represented in the feature space, for example, categorization (applying class labels or keywords), retrieval ( $k$ -NN), data mining, etc.

### *Multimedia Data Representation – Data and Information*

In its raw form multimedia data is suitable for human consumption (e.g., to watch a movie or listen to music), to be archived for historical or legal purposes, etc. This data consists of images (e.g., JPEG or BMP format), audio (MP3, WAV, etc.), video (MPEG2), etc. Multimedia content that is stored in a similar format is referred to as data, because it is not retained in a format that is optimized for processing by an implementation of an algorithm (e.g.,  $k$ -NN similarity search, data mining). For example, consider the car shown in Figure 3. The initial image is in JPEG format and amenable to human consumption (a person can observe the image and recognize it as depicting a vehicle; more specifically a yellow sports car). However, from the perspective of a machine, it is a sequence of bits. Multimedia data can be processed and have information extracted from it. The image itself is data, and the data that has been extracted through some process and can be associated with some higher-level concept is termed information (for example, the label “car”). Consider again the automobiles shown in Figure 2; the images themselves are considered data, and feature vectors can be considered information. The information contained in the feature vectors represents specific aspects of the images and is suitable for further automated processing. A feature vector can also be considered to represent a point in a multi-dimensional Euclidean space which is called the feature space.

In this work image frames are denoted by a capital  $F$ . For example, the first frame of a video is denoted  $F_1$ , and the  $i^{th}$  as  $F_i$ . Images are composed of pixels, which may be referenced within the image by their 2D coordinates  $(x,y)$ , for example, the pixel  $\mathbf{x} = (x,y) \in \mathcal{R}^2$  or,



$$\mathbf{x} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \quad (1)$$

Images themselves (the terms image and frame are used interchangeably in this work) are represented as a matrix of pixel values. For simplicity we confine pixels to be represented as 8-bit integers, which range from 0 to 255, and signify the intensity of the (color) channel. Additionally, we work with two types of images; color images and grayscale images. Color images are represented as a 3-dimensional matrix, where the first two dimensions are the width and height, and the third dimension is the color component; for the RGB color space, indices in the third dimension correspond to the R, G and B color planes. Grayscale images (also known as black and white) are represented as a 2-dimensional matrix of pixel values, where a value of 0 corresponds to black and 255 is white. Often color images are converted to grayscale because the 2D matrices are simpler to work with and manipulate, and for many tasks (such as finding the difference between frames) the results, when performing the tasks on grayscale as opposed to color, is acceptable. A simple formula for converting from the RGB color space to the grayscale color space is as follows:

$$\mathbf{G} = 0.30\mathbf{I}_R + 0.59\mathbf{I}_G + 0.11\mathbf{I}_B \quad (2)$$

Where  $\mathbf{G}$  is the corresponding grayscale image. We note that there are many different weightings that may be utilized in the conversion and additionally, there are many different color spaces in addition to RGB which have different properties. The suitability of which color space to use varies with the application. For additional details pertaining to image representations and color space, the reader is referred to a computer vision text, for example, the work of (Szeliski, 2010).

### *Digital Sampling and Reconstruction*

In order to be communicated digitally and processed by a computer, analog signals must first be digitized. Digital sampling is the process of converting a continuous signal (e.g. in terms of space or time) into a sequence of discrete numeric values. More specifically, there are two processes associated with digital sampling: sampling and reconstruction. In the sampling process a continuous signal is converted into a sequence of measurements by periodically measuring the value of the continuous signal. This period  $P$  is called the sampling interval and can be measured in terms of time or space (depending upon what it is that is being sampled). The measurements themselves are referred to as samples, and the reciprocal of the period  $1/P$  is the sampling frequency. When the sampling period is measured in terms of seconds, then the sampling frequency is referred to as hertz.

The transformation of a series of discrete samples into a continuous signal is called reconstruction. The portions of the continuous signal that are not represented in the discrete samplings may be reconstructed through the process of interpolation. For further details pertaining to sampling theorems the reader is referred to (Jerri, 1977).

### *Data Representation and Features*

Data can be classified as *structured* or *unstructured*. Structured data is organized in accordance with a data model (Hoberman, 2005). Some examples of structured data include tabular data stored in a relational database or in an XML file. Data in this class is identifiable (by both humans and computers) due to its structure. Unstructured data is not inherently organized by an identifiable structure. Examples of unstructured data include audio (e.g. MP3 files), images and video.

Unstructured data can be categorized by its inherent dimensionality. The simplest type of unstructured data consists of alphabet characters, and the more complex is video. Table 1 provides a list of different types of unstructured data. The data listed as “Continuous” in the state column consists of data that is related temporally, and one or more of these classifications (or types) of data may be combined and still be considered multimedia (Grimes & Potel, 1991).

**Table 1: Modalities of unstructured data by dimensionality**

Data dimensionality	Example of data	State
0	Characters, text	Discrete
1	Audio, output from sensor	Continuous
2	Image, graphics	Discrete
3	Video, animations	Continuous

As previously stated, features represent a measurable property of a type of data that can be observed. Typically more than one feature is extracted to represent an item of multimedia data, and taken together these features form a vector which can correspond to a point in a multidimensional Euclidean feature space. The process of identifying and calculating features from multimedia data is called feature extraction.

There are different types of features; and some features are applicable only to certain modalities of data. Three types of features are described here: *geometric*, *statistical* and *meta*. *Geometric* features apply to specific objects that have been identified within a unit of multimedia data (such as a frame of video). Before objects can have features calculated for them, a previous processing step must have been executed to identify the objects contained in the data item. An example of a geometric feature is a moment. In image processing, a moment is a weighted average of the intensities of the pixels that represent the appearance of an object. Features that can be derived from the moment include area (the number of pixels that contribute to the object’s representation) and also the centroid (or, the coordinates of the center of the object) (Hu, 1962).

Another simple geometric feature is the shape number (Bribiesca & Guzman, 1980). The shape number represents the contour of a shape, and is a sequence of that describe the directions of line segments that one would encounter when tracing the shape of an object, having started from some particular boundary point. For details about shape and image processing the reader is referred to a computer vision text, for example (Nixon & Aguado, 2012).

A *statistical* feature is another type of feature that can describe an image. Statistical features are generally applied to the image as a whole. A histogram is an example of a statistical feature that can represent a property of an image, for example, the intensities of the pixel values that represent the appearance of the image. Consider, for example, a grayscale image, which is a two-dimensional image whose pixels represent shades of gray with intensities ranging from 255 (white) to 0 (black). A histogram representing a particular grayscale image could have 256 bins, one for each possible pixel intensity, and the value of each bin would be the number of pixels contained in the image with that particular value. To make the histogram more compact, the bins can be generalized to represent non-overlapping ranges of pixel values. Other features that could fit into the statistical category are edges (e.g. the number of pixels that represent edges in the image, as outputted by some edge detection algorithm (Harris & Stephens, 1988)), and interesting points within the image (Lowe, 1999).

*Meta* features are another class of features that can describe data. Meta features apply to the data as a whole. For example, for an audio recording of music a meta feature could be the name of the artist who recorded the work. For an image, a meta feature could be the focal length of the lens used to capture the image, or the model of camera. For video, frame rate, aspect ratio, language, producer, etc. are all examples of meta features.

As indicated in Table 1, the term multimedia encompasses a number of different modalities of data. In the remainder of this work the modalities of data that are of primary consideration primarily are video, and the images (i.e. frames) extracted from the video. It is important to also note that the data (and metadata) generation techniques considered in this work are those that are primarily automated. For example, some algorithms for image segmentation require a human to provide “seed” parameters, but we would still consider such a technique to be automated; as opposed to a technique in which a human observes some data and performs some manual transformation such as determining relevant labels to associate with said data. This includes user correction (such as correcting a metadata value that is incorrect) or applying (i.e. associating) context with an object (e.g. marking whether or not a video sequence contains a representation of a particular person), other than for purposes such as determining a ground truth baseline.

### *Feature Extraction*

A significant amount of data is required to represent (or store) images and videos. It is common to implement an algorithm that takes image (or video) data as input and performs some service, such as finding similar items in a larger collection. In order for such an algorithm to execute this kind of a search, the entire collection of data would need to be loaded into the primary memory of the computer and operated upon by the CPU. For example if the search were conducted over video data that had an audio track multiplexed with the video data (all within the same file), then the audio data may need to be read in order to extract the video contents of the file. Furthermore, if the search was for a particular object observed within the video, then many parts of the video itself would need to be processed (e.g. all the video frames). Therefore a

significant quantity of irrelevant data must be loaded and processed, even though much of it is not needed.

A solution is to reduce the quantity of data that must be processed, such that only data that is likely to be relevant to the objective function is loaded and processed, and data that is likely to be irrelevant does not consume processing resources. A relevant data reduction technique that can do this is called feature extraction. Feature extraction is a process that reduces the dimensionality of a more “verbose” data format (such as an image or video) by performing some sort of transformation algorithm to arrive at a more concise representation that still describes the original item (or some aspect of the original item) with some sufficient (representative) level of accuracy. That is, feature extraction is a technique to reduce the quantity of data (or, the dimensionality) required to represent some target item. Oftentimes the particular features that are extracted are associated with some application (or domain) specific application. By applying domain specific knowledge to the feature extraction process, features that are more relevant (or expressive with respect to a particular algorithm or class of algorithms) may be selected.

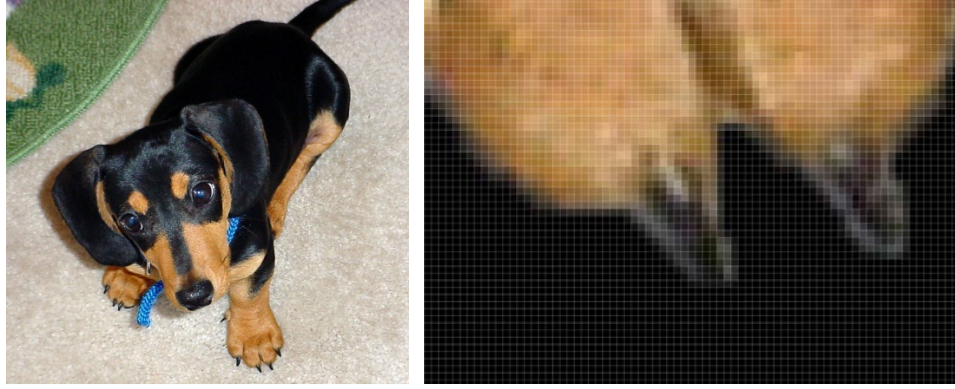
An additional reason for reducing the amount of data that must be processed is that some machine learning and data mining algorithms are less effective when the data input is of a high dimension. For example, the accuracy of such algorithms may degrade as the dimensionality of the data increases. This can be the result of what is commonly referred to as the curse of dimensionality (R. E. Bellman, 1986). The curse of dimensionality refers to artifacts that can occur when analyzing data in a high-dimensional space that would not occur in a lower dimensional space. For example data points represented in a higher-dimension that is intrinsic to the data will appear sparse, and similar items will lie farther apart, potentially reducing the

accuracy of techniques whose effectiveness is affected by the closeness of data in the feature space, such as nearest neighbor retrieval.

In addition to feature extraction, other common techniques for data dimensionality reduction include *principal component analysis (PCA)* (Jackson, 1991; Jolliffe, 2005) and *factor analysis (FA)* (Mardia, Kent, & Bibby, 1980) to name a few. The reader is referred to the work of Fodor for a survey of additional and related dimensionality reduction techniques (Fodor, 2002).

### *Background Subtraction*

Background subtraction is the process of identifying objects (or portions thereof) of interest in an image, from the rest of the image. The output from the background segmentation process is a mask image of binary values that indicates which pixels (in the corresponding image) represent the foreground object (or said another way, the pixels which are detected to not represent the scene background). For example given a view of a parking lot, a security officer might be interested in monitoring which vehicles have recently left or arrived. As an image is composed of a series of pixels, the task of background subtraction is that of determining which pixels are meaningful; that is, determining which pixels are part of an object of interest and which are not. As an example, consider the animal shown in the left image of Figure 4 (left). Figure 4 (right) shows an enlarged view of the animal's foot; the non-black pixels contribute to the animal's appearance, and the black pixels contribute to the scene background. In this section we consider the task of background subtraction where the camera is fixed.



**Figure 4. Complete image (left), and magnified view of extracted foreground pixels (right).**

In order to determine which pixels contribute to the appearance of object of interest and which do not, a model of the scene must be constructed; or learned. Note that in this usage the term background is ill-defined as its meaning can vary depending upon the context and application. However, in this case we consider the background to be pixels whose brightness changes slowly, or with some periodic motion (such as a tree swaying in the wind). (For example, the brightness of an outdoor scene will slowly change as the sun changes position in the sky.)

Frame differencing is the simplest case of background subtraction, in which the foreground pixels of a scene can be determined by taking two images (and converting them to grey scale images to simplify handling the separate color channels) and subtracting (or, finding the absolute difference) between the pixels in the images. Pixels that are beyond some threshold can be considered to be part of an object of interest (e.g. something that moved and caused the pixels to change illumination values).

$$|F_{i+1} - F_i| > t \quad (3)$$



Frame differencing can be improved upon by computing the average pixel value from the last  $n$  frames, and slowly updating the background model over time to account for slow changes to the illumination of the scene. To account for this, an adaptive background model can be maintained by calculating the running average of the background  $B$  over time:

$$B_{i+1} = \alpha F_i + (1 - \alpha)B_i \quad (4)$$

Where  $B_i$  the current background model,  $F_i$  the current frame of video and  $\alpha$  is the learning rate (for example,  $\alpha=0.05$ ) (Cucchiara, Grana, Piccardi, & Prati, 2003; Lo & Velastin, 2001).

The background models just discussed model each pixel independently from its neighbors and base the color model on each pixel's recent history, such as the weighted average of the previous  $n$  frames. These don't take into account complex scenes with moving objects, like branches moving in the wind, moving water or clouds passing overhead. Background subtraction methods that improves upon these base the value of background pixels on a *probably distribution function (PDF)* that follows a Gaussian distribution (Wren, Azarbajani, Darrell, & Pentland, 1997), or a *Mixture of Gaussians (MOG)* (Stauffer & Grimson, 1999). The downside of MOG is that it does not adapt well to fast-changing backgrounds like waves, or to cases where more than a few Gaussians might be required. The Codebook (K. Kim, Chalidabhongse, Harwood, & Davis, 2005) background segmentation model takes into consideration periodic background variations over a long period of time. In order to conserve the amount of memory required to implement the algorithm, a codebook is constructed by associating with each pixel one or more codewords which can be thought of as clusters of colors at each pixel (e.g. each pixel may be associated with one or more codewords), and the clusters may not necessarily correspond to a Gaussian distribution or any other parametric distribution. That is, Codebook

still encodes the background representation on a pixel-by-pixel basis. Classification of a pixel as background or foreground is done by comparing a pixel's value to the corresponding codewords; if its color distribution is sufficiently close to one of the codewords and its brightness is within a range of the corresponding codeword, the pixel is considered to be part of the background, else it is classified as a foreground pixel. For additional information pertaining to background subtraction methods the reader is referred to the works of (Piccardi, 2004) and (Cheung & Kamath, 2004).

It should also be noted that the pixels in the foreground mask might not always represent the object completely; that is, there may be some error due to noise. For example, as can be observed in Figure 5, in some situations the pixels that represent the appearance of the object can match the color of the background. In such cases the object might appear as two objects (as can be observed in Figure 5 in the object labeled “5”), or as a cluster of “loosely connected” points. This type of error can be mitigated by introducing a post-processing step to reduce noise in the binary foreground mask image, or also group together nearby disconnected components that could correspond to the same object (Parks & Fels, 2008).



**Figure 5. A traffic scene (left), and the corresponding foreground mask (right).**

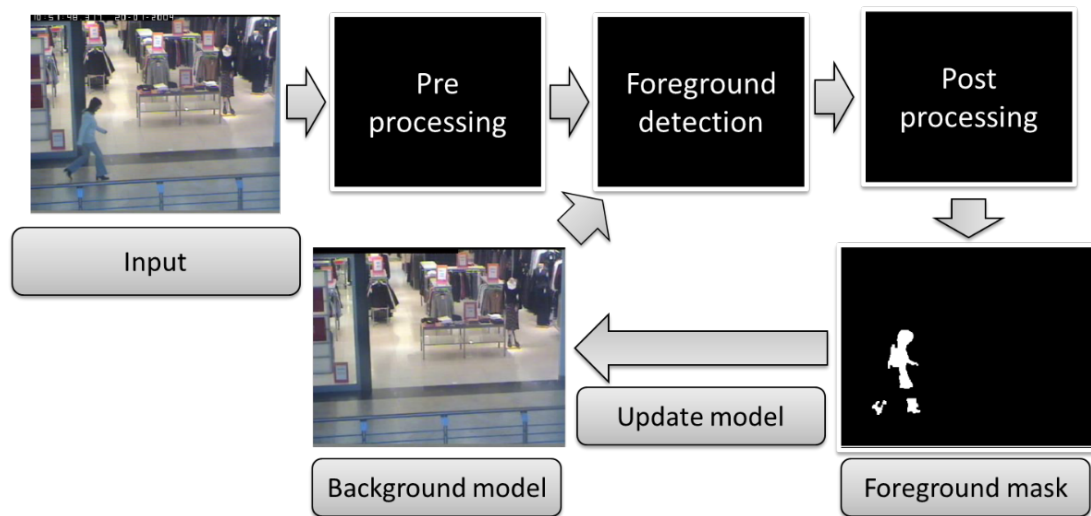
The typical steps of a background subtraction algorithm are illustrated in Figure 6. As indicated in Figure 6, major steps include a preprocessing step (such as smoothing the image to reduce noise from the image capture process), utilizing the background model to detect the foreground, performing post-processing (such as running a connected component algorithm or ignoring pixels indicated in the mask that are not connected to a larger grouping of pixels), and finally updating the background model in preparation for processing the next frame of input.

### *Segmentation of Image Regions*

Image segmentation refers to the process of grouping the pixels that compose an image into multiple salient regions. The pixels that are grouped together in a particular region are related; for example they form a part of an object or correspond to a surface, have a similar appearance, etc. While image segmentation is an ill-posed problem, it is a widely researched topic in computer vision. Researchers have taken a number of approaches to solve this problem and there are many algorithms available. The Watershed algorithm (Beucher & Lantuejoul, 1979) is a popular image segmentation algorithm, which was first introduced in 1979 and now has many variants. Other segmentation methods are based on snakes (Kass, Witkin, & Terzopoulos, 1988) and active contours (Xu, Yezzi Jr, & Prince, 2000). More recent approaches aim for some type of global optimization, for example consistency within a region or dissimilarity between regions (Cremers, Rousson, & Deriche, 2007).

Figure 7 shows an example of image segmentation; the pixels representing the athletes are grouped into a number of different regions. In this example the authors follow a graph-based approach to image segmentation (Felzenszwalb & Huttenlocher, 2004). Figure 8 provides another image segmentation example, again based upon a graph cut technique. In this example the user provided seeds (i.e. hints) to the image segmentation algorithm. For additional details

pertaining to image segmentation algorithms and their applications, the reader is referred to (Fu & Mui, 1981; Pal & Pal, 1993; B. Peng, Zhang, & Zhang, 2013).



**Figure 6. Typical processing steps implemented in a background subtraction algorithm.**



**Figure 7. Image segmentation example, based upon graph-based representation (Felzenszwalb & Huttenlocher, 2004) © 2004 Kluwer Academic Publishers.**



**Figure 8. Another graph cut segmentation image segmentation example (Boykov & Funkalea, 2006) © 2006 Springer Science + Business Media, LLC.**

### *Tracking Objects within a Single Camera*

The problem of object tracking can be defined as the task of following an object as it moves about within the view of a camera. Though it can be defined simply, object tracking in general is a difficult problem due to noise in the images and due to the capture device, the loss of information as a 3D world is projected into a 2D image, illumination changes in the scene, real-time processing requirements, selecting objects to track, etc. Sometimes simplifying constraints can be imposed onto the problem to make it more tractable; these include assuming smooth object motion and velocity, prior knowledge about the shape or size of the objects to be tracked, and assumptions pertaining to physical constraints (e.g. assuming an object will not move through a wall or fence). The objects to be tracked must first be segmented from the background. Once objects are identified, they can change shape and appearance as they move (for example, if the lighting in the scene is not uniform; the shape of people changes as their legs and arms move

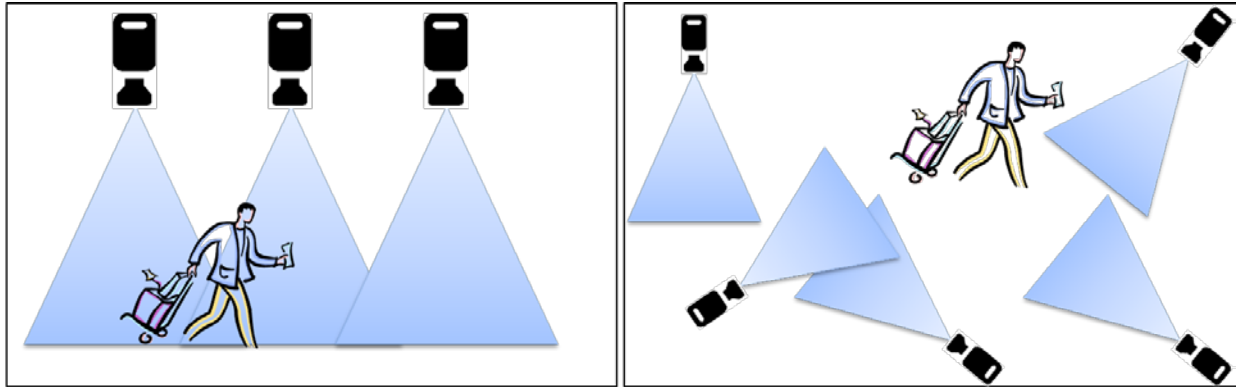
about as they walk). Objects in the foreground can obscure the view if the objects, for example the pole of a street light or a car driving by. The tracks of the objects themselves can change abruptly, or the camera can move, or the size of the object can appear change as its proximity to the lens of the camera changes, to name just a few of the scenarios that may be encountered. To help simplify the problem, many of the tracking algorithms make the assumption that the track of an observed object takes will not change sharply or that their brightness will remain constant. The suitability of which tracking algorithm to use depends on many factors including the selection image features, motion, shape and appearance, to name a few. Features used for tracking include color, edge (object boundaries), optical flow (e.g. motion) and texture.

Object tracking is important for a number of applications such as surveillance, human computer interaction, medical imaging and intelligent transportation systems (e.g. traffic control) to name a few. Many of these algorithms have sufficiently good performance to be usable for real-time object tracking, for example (Berclaz, Fleuret, Turetken, & Fua, 2011) reports to have  $O(KN \log N)$  performance, and (Pirsiavash, Ramanan, & Fowlkes, 2011) reports approximately  $O(NK)$  performance when tuned. For a comprehensive review of object tracking algorithms the reader is referred to the work of (Yilmaz, Javed, & Shah, 2006).

### *Distributed Object Tracking with Multiple Cameras*

Tracking objects with a distributed camera system is a difficult task. Many distributed tracking algorithms assume a scenario with a centralized computer. However, with sensor networks that scale to contain hundreds of cameras, the centralized approach is not tractable due to the CPU, memory and network capacity requirements required to route video data to a single sink and to then concurrently process the video streams.

Two sub-problems encountered with distributed tracking relate to configuration parameters and topology estimation. The cameras on the network (and corresponding processing nodes) should be able to come to a consensus pertaining to how objects should be represented and configuration of global network parameters (i.e. calibration), without requiring a centralized node to make the decision. Pertaining to topology, there are two problem scenarios; overlapping and non-overlapping fields of view, which are illustrated in Figure 9. In the overlapping scenario two neighboring cameras will observe the same scene because some portion of their views overlap. In this case the cameras' topology can be represented by a graph, in which two cameras are neighbors (they have an undirected edge connecting them) if their observed scenes overlap. The cameras can estimate their parameters (e.g. their spatial relationships) by observing the same objects at the same time. In the non-overlapping case there are a number of works that attempt to estimate the topology of the network. For example, in (Zou, Bhanu, Song, & Roy-Chowdhury, 2007) the topology of the camera network is estimated by tracking people using face recognition. In (Javed, Rasheed, Shafique, & Shah, 2003) the topology of the camera network is estimated with a training phase. Objects are tracked based upon their appearances and spatio-temporal movement, for example by making the assumption that objects will continue on a fixed trajectory and that structural constraints such as walls are fixed or that object movement is confined to roads or tracks. Or alternatively, Rekleitis et al. proposed a method to calibrate a camera network by utilizing a robot that moves a calibration pattern (Figure 10) through the fields of views of the cameras (Rekleitis & Dudek, 2005; Rekleitis, Meger, & Dudek, 2006). For a more comprehensive review of distributed computer vision algorithms, which includes a review of distributed calibration and tracking algorithms, the reader is referred to (Radke, 2010).



**Figure 9. Cameras with overlapping (left) and non-overlapping (right) fields of view.**



**Figure 10. Camera network calibration utilizing a robot with a pattern (Rekleitis et al., 2006). © 2006 Elsevier B.V.**

### *Supervised and Unsupervised Learning*

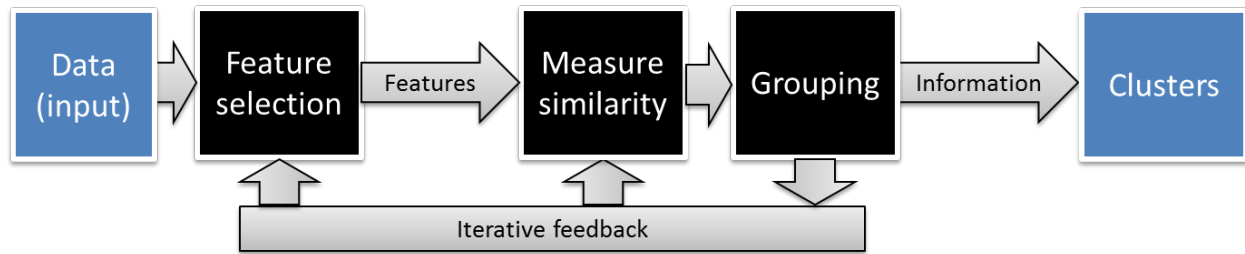
Machine learning refers to a class of algorithms that analyze data, adapt and learn from it. The data can come from a database or a sensor such as a camera. The data is typically associated with labels; or classifications. It is generally done in two phases, where the first phase is a training phase that identifies known properties from the training data. In the training phase an initial dataset is analyzed to extract relationships (and corresponding probability distributions)



imbedded within the data. In the second phase the learned relationship information is utilized to classify (i.e. label) new data. Often a complete knowledge of the hidden variables and distributions cannot be learned from the training data, and thus assumptions (or simplifications) must be made when classifying unknown data. This section provides brief reviews of supervised and unsupervised learning algorithms.

The goal of supervised machine learning is to create a classifier that can associate an output classification with some input data. This is done by first examining training data, often in the form  $\{(data\ vector_1, output\ label_1), \dots, (data\ vector_n, output\ label_n)\}$  during the training phase, and applying a function that maps the input data to the output labels. There are a number of algorithms that fall into the category of supervised learning; some popular ones are the naïve Bayes classifier (Rish, 2001), the k-nearest neighbor algorithm (Weinberger, Blitzer, & Saul, 2006) and support vector machines (Cortes & Vapnik, 1995).

Unsupervised learning differs from supervised in that the training data is unlabeled. The task is to uncover the structural relationships among the data. Unsupervised learning algorithms are applicable to scenarios where it is difficult or not cost effective to label the input data (for example, in the case of speech recognition) or when the features (that classification will be based upon) are unknown beforehand. Common unsupervised learning techniques include clustering algorithms and dimensionality reduction techniques (Duda, Hart, & Stork, 1995).



**Figure 11. Essential clustering steps; information flow.**

### *Clustering*

Clustering often deals with unlabeled data, and as such, it falls under the category of unsupervised learning (as opposed to supervised learning, which entails the association of items with labels). More succinctly, clustering is the assignment of objects into groupings based upon some measure of similarity. It is a collection of techniques that are applicable to the large mass of unstructured data such as video, image, and text from webpages, for example, where a predefined model that relates the data does not exist (or can change depending upon the context from which the data is accessed). Clustering may be used to uncover the underlying structure inherent in seemingly unstructured data, for classification (e.g. grouping based upon a similarity measure), and for data summarizing (e.g. hierarchical browsing to allow for the efficient search of image databases) (J. Y. Chen, Bouman, & Dalton, 2000; A. K. Jain, 2010).

The basic task of clustering is, given  $n$  objects, find  $k$  groupings, such that the objects that are grouped together are more similar (based upon some measure of similarity) than objects that are associated with one of the other groupings. The groupings (or clusters) themselves can be described based upon the compactness of the data, its shape, overlap with neighboring clusters, etc. When implementing a clustering algorithm for a particular application, the choice of algorithm and parameters needs to be guided based upon some decisions such as; how to normalize the data, which similarity measure to use, and how to incorporate any domain

knowledge into the clustering process. Figure 11 depicts some typical steps that must be undertaken in a clustering process (A. K. Jain & Dubes, 1988). Given data to cluster (which does not necessarily need to be finite in size; as would be encountered when processing stream-oriented data from sensors), features from the data representation must be identified. The features must then be extracted, and measured for similarity by applying a distance function such as the Euclidean distance (Anderberg, 1973). Based upon the resulting clusters the process can be repeated with updated parameters or data, in order to optimize some aspect of the resulting clusters. For example, if a cluster is too large the corresponding data may need to be grouped into smaller clusters, or if two clusters are too close together with a large number of overlapping data points in the feature space, the clusters may need to be merged. The clustering process can end when improvements to the clustering are below a threshold (based upon some criterion function) or when some maximum number of iterations are reached. The reader is referred to (Kanungo et al., 2002) and its associated references for in-depth discussion and analysis of clustering algorithm stopping criteria; for example convergence based upon local vs. global optimization, etc.

### *A Brief Review of Multiple-Instance Learning*

This section offers a brief review of *multiple-instance learning (MIL)*. Objects observed in video streams are modeled using a multifaceted object model and concepts borrowed from MIL are at the heart of the object tracking technique that is used to facilitate the recognition of objects across video streams in the LVDBMS.

Traditional supervised learning (e.g. (Settles, 2010)) is traditionally conducted in two phases; where first a training phase builds an analytical model based upon training data, and second a classification phase leverages the model to provide insight pertaining to previously-

unseen data; i.e. the classification of new data. In the training phase machine learning algorithms are presented with evidence in the format  $\{<object, result>\}$ . As an example, consider the *Expectation-Maximization (EM)* algorithm (Dietterich, Lathrop, & Lozano-Perez, 1997; Maron & Lozano-Perez, 1998) and Bayes classifier (Rish, 2001). Objects are typically represented by a feature vector, so the mapping provided to the classifier consists of a mapping from a feature vector to a class label which is to be associated with objects having some similarity to the feature vector. In traditional learning problem scenarios objects are represented by a single feature vector; for example an image might have a feature vector composed of 150 components. However, complex situations are encountered in practice such that the learning algorithm has incomplete knowledge about an object (i.e. the training samples). For example, Diettrich provides an example of a locksmith, who is given a set of key rings, where each key ring contains multiple keys. It is the job of the locksmith to determine which key opens a particular door, in a set of doors. However, the locksmith is not given direct access to open the door with the keys (that is, the locksmith cannot implement an algorithm to sequentially attempt to open each lock with each key in order to determine the key to lock mapping). Thus the locksmith must infer the mapping from the evidence (the set of key rings). Thus, the locksmith does not have full knowledge of the training data and the relationship between the evidence and the classification is indeterministic. There are many other domains where similar indeterministic relationships exists; for example in text-based search and image processing and retrieval. A renowned searching example involves a query for the term “nut”, where nut could refer to a shell around an edible kernel; a round threaded object that is fastened to the end of a bolt to hold the bolt securely in place; a person who is not sane, an Egyptian goddess, or a mechanical device used for climbing mountains, among other possible definitions.

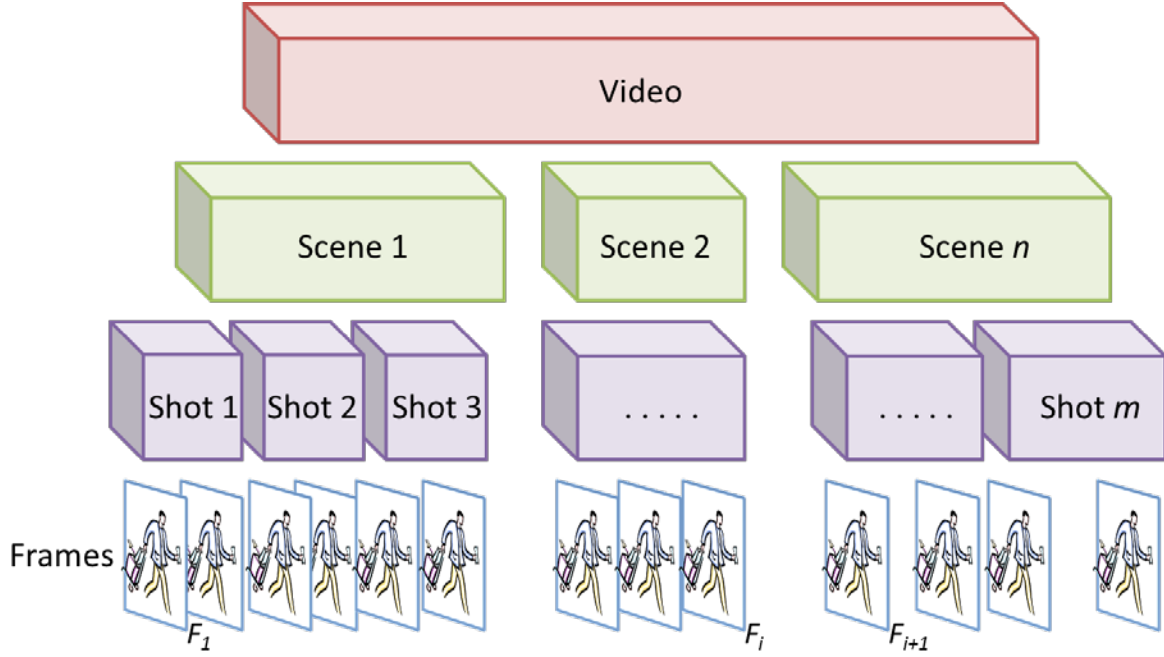
In MIL training data is provided in the form of bags of instances, where a bag is similar to a set but can contain duplicate items (X. Chen, Zhang, Chen, & Chen, 2005; Cheng, Hua, & Yu, 2010). A bag is labeled positively if it contains at least one instance of a particular concept and negatively if it does not contain any instances of the concept. For example, consider an image that is segmented into regions, and each region is then represented by a feature vector. The image in its entirety can be represented by a bag of feature vectors, where each feature vector in the bag represents (i.e. is representative of) one of the segmented regions (Maron & Ratan, 1998). Now, consider the case where a user is querying a dataset of pictures for images of automobiles. A bag is labeled “automobile” if it contains at least one instance of an automobile (more precisely, it contains a feature vector corresponding to a region that depicts an automobile in the image). If the image does not contain any instances of automobiles, it is a negative example of the concept “automobile”.

Correspondingly in MIL, training data consisting of positive and negative examples are provided to the learner as bags of instances. A bag is labeled as positive if it contains an instance corresponding to a particular concept, and negative otherwise. A learning algorithm “learns” concepts (i.e. builds a statistical model thereof) by applying a learning algorithm. The task is to then classify an unknown object by applying a distance function. In the example of images, it is possible (and likely) that different images will be segmented into a different number of regions and thus, their corresponding bags will have differing cardinalities. There are different methods to compare bags with different cardinalities, for example, a normalization method (or factor) can be applied (Gartner, Flach, Kowalczyk, & Smola, 2002). For an in-depth discussion of MIL algorithms and concepts the reader is referred to the work of (Ray & Craven, 2005).

### *Shot Boundary Detection and Representative Image Selection*

Automated shot boundary detection is an essential component of video content analysis; it is the temporal segmentation of a video into a continuous scene. Although such a partitioning could be done manually, given the vast quantities of video that is collected, analyzed and stored today, manual shot boundary identification is not feasible to be done as a manual process. A video can be thought of as a series of scenes. Scenes consist of a logical grouping of one or more shots, where each shot is a contiguous sequence of frames captured by a camera; Figure 12 provides a visual representation of the hierarchical relationship between a video, scenes, shots and frames. Shot boundary detection is useful for a number of applications that pertain to organizing or categorizing shots (and their corresponding videos) for later retrieval and indexing and other offline analysis.

Shots are concatenated together with a transition separating them, where the transition can be abrupt or gradual. An abrupt transition (also termed “cut” or “hard cut”) is a very quick changeover from one shot to the next. For example, a transition that is done in two consecutive frames. A gradual transition blends the shots together with a more gradual spatial or chromatic effects such as a wipe, fade or dissolve, to name just a few of the many different types of transition effects.



**Figure 12. Hierarchical structure of a video depicting scenes, shots and frames.**

Detecting cuts is a difficult problem for computers. A human can interpret the situation depicted in the video and understand semantically what it is being observed. So when a transition occurs (say, a bolt of lightning striking in an outdoor scene) the human can easily make a determination if what was observed was a cut. From the perspective of a computer, what is observed is temporally correlated data from consecutive frames of video. A simple way determining a cut is to compare consecutive frames pixel by pixel (that is, the intensity of the pixels) and if the change is beyond some threshold, to mark the frame as a cut. The below equation computes the difference in pixel intensity values for a frame  $F$  at index  $i$  and pixels at coordinates  $x$  and  $y$  within the frame, for some threshold  $t$ :

$$D_i = \begin{cases} 1 & \text{if } |F_i(x, y) - F_{i+1}(x, y)| > t \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

This equation counts the number of pixels that have changed between two frames and can be used as a metric to determine a cut. Given an image that is of dimension  $M$  by  $N$ , and thus contains  $M*N$  pixels, this can be rewritten as follows:

$$\frac{\sum_{m,n=1}^{M,N} D_i(m,n)}{M*N} * 100 > t \quad (6)$$

This equation computes a ratio of pixel changes beyond an intensity and applies a threshold. The shortcoming to this method is camera movement; a pixel in one particular position, say position (40,50) in one frame, is compared with a pixel in the same position in the next frame. The problem is if the camera is moving, that adjustment is not taken into consideration. This is the problem taken on by (H. J. Zhang, Kankanhalli, & Smoliar, 1993). They pair a histogram comparison technique with motion gradient detection to improve upon the simplistic threshold technique. Other techniques utilize histograms, motion vectors, block matching algorithms and the discrete cosine transformation (Boreczky & Rowe, 1996). Others compute the difference in color histograms, look at the ratios of edges that are detected in the frames, the contrast and standard deviations of pixel intensities to detect hard cuts, fades and dissolves (Feng, Fang, Liu, & Fang, 2005; Ford, Robson, Temple, & Gerlach, 2000; Lee, Yang, & Lee, 2001; Lienhart, 1999; X. Liu & Chen, 2002; D. Zhang, Qi, & Zhang, 2001; Zheng, Yuan, Wang, Lin, & Zhang, 2005). A technique developed by (Oh, Hua, & Liang, 2000) computes the difference in the background of a scene to determine a scene cut, while handling camera motion. The benefit of their technique is that it is less sensitive to predefined threshold values. Other techniques claim that changes to illumination and the motion of the camera (or objects observed in the scene) are contributors to the poor performance of shot detection transitions. In order to mitigate this they utilize clustering and support vector machines and independent component



analysis to produce more accurate shot boundary detections (Camara-Chavez, Precioso, Cord, Phillip-Foliguet, & De A Araujo, 2007; J. Zhou & Zhang, 2005; Y. H. Zhou, Cao, Zhang, & Zhang, 2005, p. -).

### *Multimedia Data Representation for Indexing*

Collections of multimedia information can grow to very large sizes, consuming many gigabytes of storage space. In order to utilize multimedia content it must be retrieved; whether the retrieval is to find a movie based upon its title, or one is looking for images, clips of audio or video segments showing a particular subject or class of objects. As an example consider a table of records in a traditional relational database. Each record in the table can be considered as a point in a multidimensional space (Samet, 1990, 2006). Consider a record for an employee-department relation with the following fields:  $\{employee\_id, department\_id, manager\_id, start\_date, end\_date\}$ . In this case, records in this table correspond with points in a 5-dimensional space, where three of the dimensions refer to, say, integers ( $employee\_id$ ,  $department\_id$  and  $manager\_id$ ) and the other two dimensions are of type date-time (i.e.  $start\_date$ ,  $end\_date$ ). The DBMS manages this collection of these records and stores them in a file on some persistent media. In order to facilitate efficient retrieval of records in this database, indexes can be created. The index itself is simply another table (or, correspondingly, a file created and maintained by the DBMS). For example, an index over the field  $employee\_id$  could contain only  $employee\_id$ 's and the location of associated records in the corresponding employee-department file. By utilizing the index file in order to resolve queries, less data would need to be loaded and processed, since the index file contains primarily  $employee\_id$  data (and not other data fields such as  $manager\_id$ ). To further enable efficient retrieval, an ordering can be imposed upon the records, either in the primary data file or in the index. However, to

accommodate future record operations to the primary data table (e.g. delete, insert, update) it is often more efficient to impose the ordering only on the data in the index files. For numeric fields, the ordering can be based upon numeric value. For character fields, the order can be based upon corresponding ASCII or UNICODE numeric values, or based upon lexicographic order. For other types of data, such as color, the ordering could be based upon the corresponding hexadecimal value (e.g. red is “ff0000”) or the color’s wavelength.

Samet (Samet, 2006) identifies five key questions that should be considered when deciding how to represent a dataset: (1) What is the type of the data; continuous, discrete? (2) The operations that will be performed; e.g. a log file might only have data appended to its end. (3) How should the ordering of the data be applied; should the data in the primary file be ordered, or only the index files? Which attributes should be included in the ordering? (4) Will more data be added or removed? Will additional attributes be added in the future? And, (5) is the quantity data sufficiently small such that all of it will fit into the primary memory of the computer hosting the database, or will disk-resident data access algorithms need to be utilized. There are many different ways data can be represented, and considering questions such as these can guide the process of designing an implementation.

When considering multimedia for browsing and searching, an index is also required. Some fundamental question are pertaining to multimedia data are *what*, *which* and *how*. At *what* granularity should the item be indexed; as a whole or by frame or a clip of frames? *Which* refers to which items should be indexed; should all pixels shown in each frame of video be represented somewhere in an index, or should only moving objects be stored? Should the time index an object appears or disappears be recorded? *How* to index an item pertains to selecting and extracting features to be indexed. Data indexing, and more specifically multimedia data indexing

is a multifaceted and difficult problem, and as such, there is a significant quantity of research and correspondingly, solutions and indexing algorithms and data structures. Some works that addresses the issues of multimedia indexing holistically are (Bolle, Yeo, & Yeung, 1998; Brunelli, Mich, & Modena, 1999; Snoek & Worring, 2005; Y. Wang, Liu, & Huang, 2000).

To illustrate this, consider the information that can be extracted from a video: the visual component (the visual content represented by pixels in the frames), the auditory information (i.e. audio tracks) and text (text that can be extracted; and metadata pertaining to the video itself such as genre, actors, etc). A multitude of semantic properties of the video can be extracted from the metadata pertaining to its content: the type of video (e.g. education, training, entertainment), the time period the video covers; major actors who appear, and so forth (Boggs, 1996; R. Jain & Hampapur, 1994). To index content that is depicted visually in the video, pattern recognition approaches can be employed; for example, template matching (e.g. Bayes classifier, decision trees, Hidden Markov Models, face and people detection (Belhumeur, Hespanha, & Kriegman, 1996), etc). The reader is referred to (A. K. Jain, Duin, & Mao, 2000) for a comprehensive review of pattern recognition techniques. To index videos, they can be decomposed into a series of semantic shots, and each shot can be individually indexed (Ide, Yamamoto, & Tanaka, 1999; Nagasaka & Tanaka, 1992). Pertaining to audio data, a number of different techniques can be employed, for example sounds can be analyzed to detect musical instruments or talking (Foote, 1997; Wold, Blum, Keislar, & Wheaton, 1996).

### *Multimedia Indexing Storage and Retrieval*

To index multimedia content, first it is decomposed and segmented and features which correspond to points in a multidimensional space are extracted. The next step is to efficiently store and retrieve those points and correspondingly, the associated multimedia content. Some of

the questions raised in the previous section are also relevant to how data will be represented for storage and retrieval. A key question pertains to by what facet of the data the index space should be organized. Consider a 32-bit integer as an example; it has a large (but finite) range of values, ranging from 0 to  $2^{32}-1$  possible values. However, the number of items in the collection (say they are organized about a single integral dimension) may be significantly less than the number of unique values a 32-bit integer can represent. To translate this example into one that is representative of storage structures, the comparison is similar to the distinction between tree-based and trie-based (De La Briandais, 1959) search methods (a trie is an ordered data structure with branching where the nodes represent prefixes and decedents of each node have the parent node as a common prefix. The data that corresponds to one of these structures can be stored on persistent media (e.g. a hard drive, etc). Storage of data on a disk implies that it is organized; logically the data is organized into buckets and physically the buckets are oriented in pages.

Pages (and correspondingly, the buckets containing data points) are stored in files. The simplest way to store a set of points in a file is as an unordered sequential list. The downside is that in order to do an equality search on the file for a particular attribute value, the entire file must be processed. Thus, if there are  $N$  records stored in the file and each file has  $d$  attributes, the processing will be of order  $O(Nd)$ . With this simple organization as a starting point, there are numerous structures (and corresponding algorithms) that facilitate indexed storage and retrieval, one example is the Grid File (Nievergelt, Hinterberger, & Sevcik, 1984).

Another straight-forward technique to organize data in a file is to utilize a hash function. The concept behind a hash function is to utilize a mathematical function to distribute items (i.e. key/value pairs) into buckets which are stored on persistent media in a file (or files, depending upon the implementation). Given a key, the hash function can suggest which bucket to store the

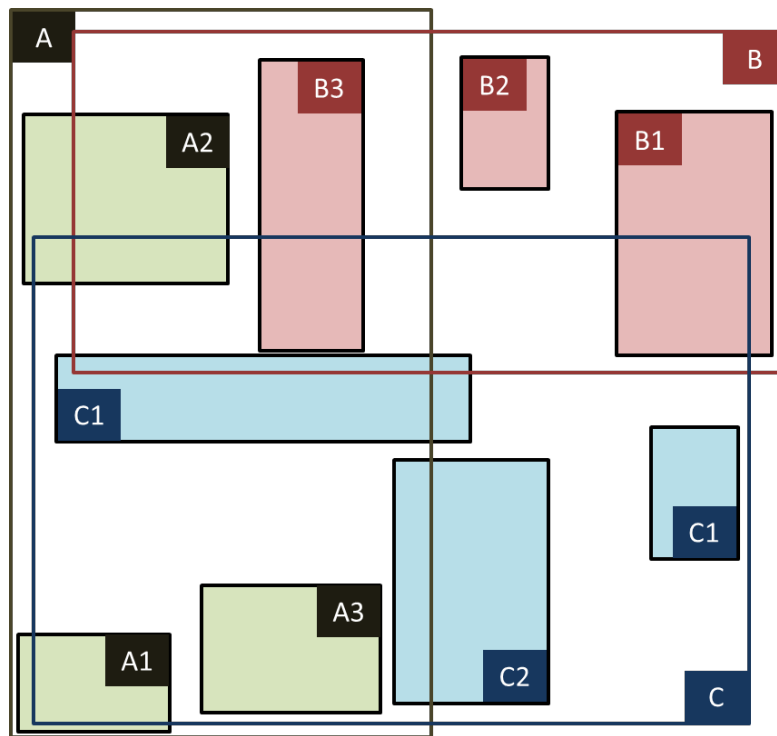
value into. In the case that the bucket is at capacity, there are various algorithms that determine how to manage the overflow (collision resolution, load factor, etc.) (Aho, Hopcroft, & Ullman, 1983; Cormen, Leiserson, Rivest, & Stein, 2001; Pieprzyk & Sadeghiyan, 2001).

When choosing an index structure it is important to consider the type of data that will be stored; for example, strings or numbers, point data, lines (or line equations), rectangles, regions, surfaces, volumes, etc., and the types of queries that will be performed; point queries, range queries, window queries, etc. For point data one can utilize index structures like the Binary Search Trees (Bentley, 1975), B-Tree (Scheuermann & Ouksel, 1982) or B<sup>+</sup>-Tree, etc). When indexing data in multiple dimensions, one can use a 2-dimensional version of the binary search tree called a Point Quadtree (Finkel & Bentley, 1974). To detect the line segments (i.e. intervals) that contain a point one can use a unit-segment tree (Bentley, 1977; Finkel & Bentley, 1974). To detect line segments that overlap one can use an interval tree (Chazelle & Edelsbrunner, 1992).

The R-Tree (Guttman, 1984; Manolopoulos, Nanopoulos, Papadopoulos, & Theodoridis, 2005) is a multidimensional tree structure for indexing spatial objects such as coordinates (i.e. points) and polygons. Some index structures such as B-trees, are not well suited for indexing spatial data due to issues related to ordering multidimensional data. Objects that are close together are grouped within a minimum bounding rectangle at the next higher level in the tree. Thus the granularity of the objects represented increases as one gets lower in the tree. When querying an R-Tree one needs to check if the query region intersects any rectangles at the current level, and if so, then the corresponding rectangles at the next lower level are recursively queried. By resolving queries in this manor, only a minority of the rectangles need to be queried due to how the space is partitioned. Figure 13 illustrates a 2-dimensional R-Tree composed of three

levels. The root level encompasses the entire terrain; and the first level consists of three rectangles namely, “A”, “B” and “C”.

The tree structures discussed thus far are referred to as space-partitioning structures; they are hierarchical data structures that decompose the space into disjoint partitions. (A notable variant of the R-Tree is the R\*-Tree (Beckmann, Kriegel, Schneider, & Seeger, 1990), which is an optimized variant of the R-Tree and performs well under both point and spatial queries with only slightly higher overhead.) A downside is that if they become unbalanced then their implementation suffers in terms of I/O. The SP-GiST index is a space-partitioning index that is designed to be I/O efficient, even in the case where the tree structure is unbalanced (W. G. Aref & Ilyas, 2001).



**Figure 13. Example of an R-Tree spatial index.**

The aforementioned structures are amenable to storing lower-dimensional data structures, applicable to problem domains such as *geographical information systems (GIS)*, *intelligent transportation systems (ITS)*, computational geometry, computer vision, video game programming, etc. However, many applications work with high-dimensional feature vectors (e.g. multimedia databases where the objects are represented by feature vectors). A common class of queries posed over this high-dimensional space is similarity queries; given an example of one object, find similar objects (or, objects exhibiting the same property). Similarity queries can be formulated as point queries (e.g. finding other objects with the same feature, such as a particular color or size), range queries, nearest-neighbor queries, and spatial join queries, to name a few examples. It should be noted that when dealing with data that is represented in higher-dimensional spaces, one encounters the curse of dimensionality (R. Bellman & Kalaba, 1959), which in essence states that as the dimensionality of the data grows, more of it must be examined when resolving point queries. In (K. Beyer, Goldstein, Ramakrishnan, & Shaft, 1999) the nearest-neighbor problem is analyzed in the context of the dimensionality of the data. Their findings are that as the dimensionality of a space approaches infinity that the distance from a point to its nearest neighbor, and the distance from that point to its farthest neighbor, converge to some distances that are within an epsilon of each other.

When working with high-dimensional data, one method of data management is to reduce the dimensionality and utilize one of the hierarchical data structures discussed previously, such as an R-Tree (Guttman, 1984) or R\*-Tree. Alternatively, there exist indices that are not based upon the dimensions (i.e. features) of the objects, but on the distances between them (the interobject distances), e.g. SparseMap (Hristescu & Farach-Colton, 1999), FastMap and MetricMap (J. T. L. Wang et al., 1999). Some types of data cannot be represented by bounding

boxes, for example, the representation of a plane or surface. In this case, these types of objects can be decomposed into a smaller volume, for example, a cube, and the corresponding cubes indexed (or for query purposes, a cube can be queried and then determined which object(s) it corresponds to). The  $R^+$ -Tree index structure can accommodate these types of items, but the downside is that one object can be represented by multiple blocks and can thus potentially lead to duplicated results being reported. To accommodate this, algorithms have been developed that take into account duplicate objects in the search space, for example (W. G. Aref & Samet, 1994; W. Aref & Samet, 1992; Samet, 1995).

When performing searches over highly-dimensional spaces, once the objects are represented in an index structure, the next step is to select a search algorithm that will efficiently resolve queries. Most of the structures discussed thus far are hierarchical in nature and the data they contain is grouped together (say, in minimum bounding rectangles) based upon some type of clustering. A few representative algorithms that utilize such data structures to resolve nearest neighbor queries are (Baeza-Yates, Cunto, Manber, & Wu, 1994; Bern, 1993; Bozkaya & Ozsoyoglu, 1999; Eastman & Zemankova, 1982; Graham, 1972; Kamgar-Parsi & Kanal, 1985; Yianilos, 1993). Since these algorithms generally entail some type of tree traversal (e.g., bounded depth-first search), they can be improved upon if conditions or rules pertaining to branch pruning can be employed (Fukunaga & Narendra, 1975; Skopal, 2004, p. -; Uhlmann, 1991; Weber, Schek, & Blott, 1998).

Finally, another method is to reduce the number of features that must be managed in the index (Hinneburg, Aggarwal, & Keim, 2000). This can be done by analyzing the data and utilizing techniques such as Singular Value Decomposition and Principal Component Analysis (to name a few). In addition to the aforesaid algorithms and structures, there are many more.



For example, (Gionis, Indyk, & Motwani, 1999; Indyk & Motwani, 1998) presents a similarity-search technique for high-dimensional data that utilizes a hashing technique, and other approaches entail mapping the data points into a different representation space, called an embedded space (Linial, London, & Rabinovich, 1995) (one of the reasons to implement such a mapping is that the embedding can improve the precision and recall of searches).

### *The Semantic Gap*

In the realm of video indexing and retrieval the semantic gap refers to the difference in representations when an activity (or object or observation) is represented as data in a computer system. More specifically, it is the lack of a strong correspondence between the low-level representation and the high-level interpretation that would be perceived by a person (Snoek & Smeulders, 2010). The semantic gap, and more specifically the detection (discovery, uncovering, etc.) of semantic information in multimedia is and has been a highly researched area of computer science (Lew, Sebe, Djeraba, & Jain, 2006; Smeulders, Worring, Santini, Gupta, & Jain, 2000). Even with all the research and significant progress that has been made, there is still significant work that is yet to be done. That is, the current state of the art is not where we would like for it to be in terms of the quality of concept detection that has been achieved (Yang & Hauptmann, 2008). Yang and Hauptmann elaborate that mainstream approaches suffer from learning problems pertaining to classifiers that do not perform well outside of the data on which they were trained (that is, they generalize poorly to domains other than the ones on which they were trained). One of the challenges is simply the scope of the problem; the number of concepts that exist are unlimited (Snoek, Worring, Van Gemert, Geusebroek, & Smeulders, 2006).

To state the problem in an alternate fashion, the semantic gap is the disparity between the information that can be extracted from the representation of an item, and the interpretation of

said data, and for humans, the interpretation can be context dependent. For example; the determination of similarity can depend upon domain knowledge, or it could be defined as the difference between the color of pixels in two images (Hatano, 1996). Another factor pertains to how people perceive and interpret similarity (Rosin, 1997; Siddiqi & Kimia, 1995; Treisman, Cavanagh, Fischer, Ramachandran, & von der Heydt, 1990). As Treisman points out, humans are particularly adept at recognizing objects, for example, to recognize the form of a white snowman in the presence of a background of snow. Another issue pertains to the representation (or lack thereof) of spatial relationships in images (for example one visual artifact is parallel lines meeting at the horizon) (S. K. Chang & Hsu, 1992; Lau & King, 1997; Schneiderman & Kanade, 1998; Smith, Self, & Cheeseman, 1990; Tagare, Vos, Jaffe, & Duncan, 1995). In general, human perception can be affected by their knowledge; that is, in terms of their cultural, geometric, categorical, perceptual, physical and literal understanding of an object and its context, and current computer algorithms encounter limitations when dealing with broad concept categories and the modeling of image semantics (Mojsilovic & Rogowitz, 2001; X. S. Zhou & Huang, 2000). From a computer science (that is, from a computational and algorithmic) perspective, one avenue to bridge this gap is being addressed in the field of *content-based image retrieval (CBIR)*.

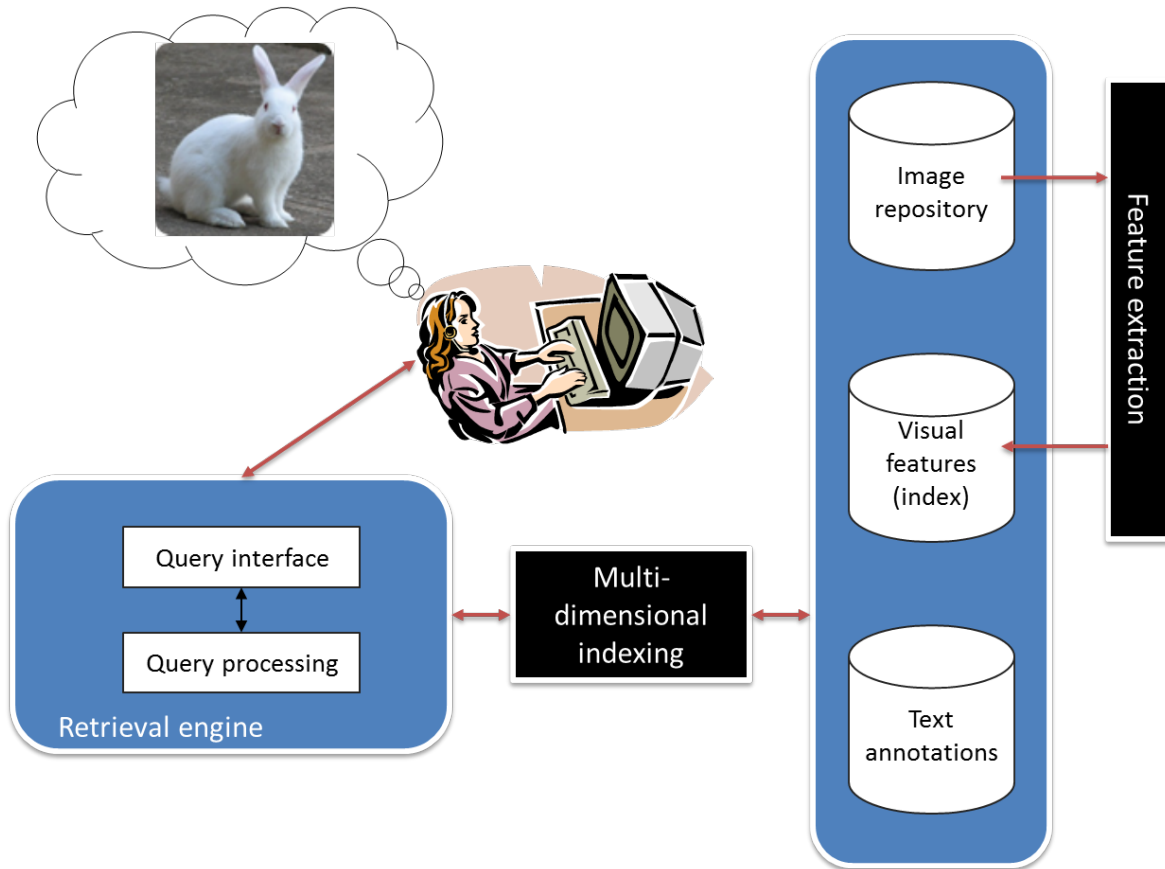
### *Content-based Image Retrieval*

In literature there are many ways in which CBIR described. In particular, it is the application of computer vision techniques to extract information from an image in an automated fashion for the purposes of retrieval. Also referred to as *query by image content (QBIC)* (Flickner et al., 1995), it pertains to the retrieval of images based upon what they visually depict; not by metadata or human-ascribed annotations, whose assignment can vary from person to person, culture to culture, reflect personal biases, etc. In CBIR systems, image data is represented by

features corresponding to its visual appearance; color, texture, shape, edges, etc. Early work in CBIR was done with pictorial databases (Blaser, 1979, 1979; N. Chang & Fu, 1980; N. S. Chang & Fu, 1980).

Present day CBIR systems facilitate retrieval by accommodating a variety of query methods, to include query by example, sketching an image by hand, random browsing, text search (i.e., keyword, speech/voice recognition) and hierarchical navigation by category (S. F. Chang, Eleftheriadis, & McClintock, 1998). Objects in CBIR systems are represented by features associated with their content. As such, feature extraction is an important step inherent to CBIR systems. Features (color, shape, texture, edges, regions, etc.) are extracted and stored in a multidimensional index (feature vectors can range from very few to hundreds of dimensions). Figure 14 provides an example of a system architecture for generic CBIR systems. A user submits an image as a query through a user interface. The query image is parsed and its representative features are extracted. The features from the query image are mapped to a multidimensional query point in the index, and similar images are returned back to the user as the query result.

There are presently many research and commercial CBIR systems; a few representative examples include QBIC (Flickner et al., 1995), Virage (Bach et al., 1996), Photobook (Pentland, Picard, & Sclaroff, 1996) and MARS (Huang, Mehrotra, & Ramchandran, 1997; Mehrotra, Rui, Ortega-Binderberger, & Huang, 1997; Rui, Huang, & Chang, 1999; Rui, Huang, & Mehrotra, 1997) to name a few. Additionally there are many good surveys on CBIR techniques and systems (Y. Liu, Zhang, Lu, & Ma, 2007; Rui et al., 1999; Zhao & Grosky, 2002).



**Figure 14. Representative architecture of a typical CBIR system.**

Although originally applied to images, *content based video retrieval (CBVR)* is another active area of research due to the commoditization of compute and storage capacity (Durkee, 2010). CBVR is semantically similar to CBIR except its domain is that of video, rather than images. Videos are segmented into shots, which may be represented by key frames (Sato, Kanade, Hughes, Smith, & Satoh, 1999), features are extracted and indexed. At that point retrieval is similar to the workflow presented in Figure 14 for CBIR (Geetha & Narayanan, 2008). Of course, video adds the potential to fuse additional data modalities not available in traditional CBIR into the indexing and retrieval process, such as correlation with audio tracks (Foote, 1999; Z. Liu & Huang, 2000; Makhoul et al., 2000).

## Research Video Database Management Systems

A variety of video database management systems have been introduced over the years, spanning from research prototypes such as BilVideo (Catarci, Donderler, Saykol, Ulusoy, & Gudukbay, 2003), VideoText (Jiang, Montesi, & Elmagarmid, 1997) and the *Advanced Video Information System (AVIS)* (Adali, Candan, Chen, Erol, & Subrahmanian, 1996), to name a few (Flickner et al., 1995; Guting et al., 2000). For example, the AVIS system segments video frames into a tree structure in order to represent the relationships between objects. However this system does not provide support for queries that resolve spatial relationships. In addition, many of these systems utilize offline processing to analyze video data to perform pertinent steps like feature extraction. Very few works, for example (Velipasalar, Brown, & Hampapur, 2010), address the real-time processing aspects of multimedia databases and surveillance applications. For a database system to be applicable to the domain of LVC it must support real-time online visual analysis of streaming video data, meaning that feature extraction and any other analysis must be done online and within a reasonably bounded time period. Relevant algorithms and data structures must also be amenable to the nature of working with continuous data streams; meaning that it is not acceptable for a software platform to perform processing for some period of time and then stop when the memory capacity of the host system has been exceeded. Thus, LVC databases have performance (i.e. efficiency) characteristics that must be adhered to in order to facilitate the pre-processing steps necessary for real-time continuous query evaluation. The video database system presented by Velipasalar provides real-time query functionality of high-level events spanning single and multiple cameras, but is lacking in terms of a high-level declarative query language; events are defined in a procedural fashion. Similarly, the KNIGHT system (Javed & Shah, 2008) utilizes a *maximum likelihood (ML)* (Akaike, 1973) framework to track

objects across cameras, but it also does not support a high-level declarative language for expressing events of interest.

### Introduction to Live Video Computing and Big Data

This section introduces basic tenants of LVC and shows that the “big data” label is applicable, due to the nature of streaming video and the real-time processing requirements.

#### *Basic Premises of Live Video Computing*

LVC is the theoretical framework upon which the LVDBMS prototype system is based. Traditional video stream processing applications (e.g. depicted in Figure 15) are designed specifically to solve a particular problem, and may be designed to work with a specific set of cameras or camera hardware. The result of this style of application development are applications that are not capable of operating with each other in a reciprocal fashion to share information and provide additional value and value-add opportunities. For example, in a hospital environment a patient monitoring system would not be able to interact with the hospital video surveillance system, and likewise, the hospital surveillance system may not be able to utilize cameras that are utilized by the patient monitoring system. The result is that additional hardware would have to be purchased in order for the surveillance system to have some capabilities in patient areas where the monitoring application is deployed. If data from applications developed in this style needs to be combined for auditing, reporting or other purposes, additional software (middleware) must be purchased and interfaced with these applications.

However the downside is that this middleware must be installed and configured on a case-by-case bases, and “adapters” for each application must be configured or developed to provide application-specific interfaces to the middleware. The middleware must then perform an

*extract, transform and load (ETL)* process to transform data received from the application-specific adapters into a common data format that is amenable to further processing. The result is additional middleware software that must be purchased and maintained and also staff resources to install, configure, maintain and upgrade, as appropriate Figure 16.

Note that libraries such as OpenCV (Bradski, 2000) and *Intel Performance Primitives (IPP)* (Taylor, 2007) are commonly used by programmers when developing these types of applications, to provide basic data-handling functionality. The OpenCV library provides a comprehensive assortment of image processing and data management routines and data structures, the IPP library provides functions and associated data structures that are specifically tuned to take advantage of features provided by modern multicore processors such as parallel data processing instructions. However these common libraries provide low-level functionality that programmers use as conveniences; and do not generally provide out-of-the-box high-level application functionality. (For example, OpenCV routines could be used to read in frames from a camera, and other routines would need to be called in the proper order with the proper parameters and settings in order to interpret imagery depicted in the frames.)

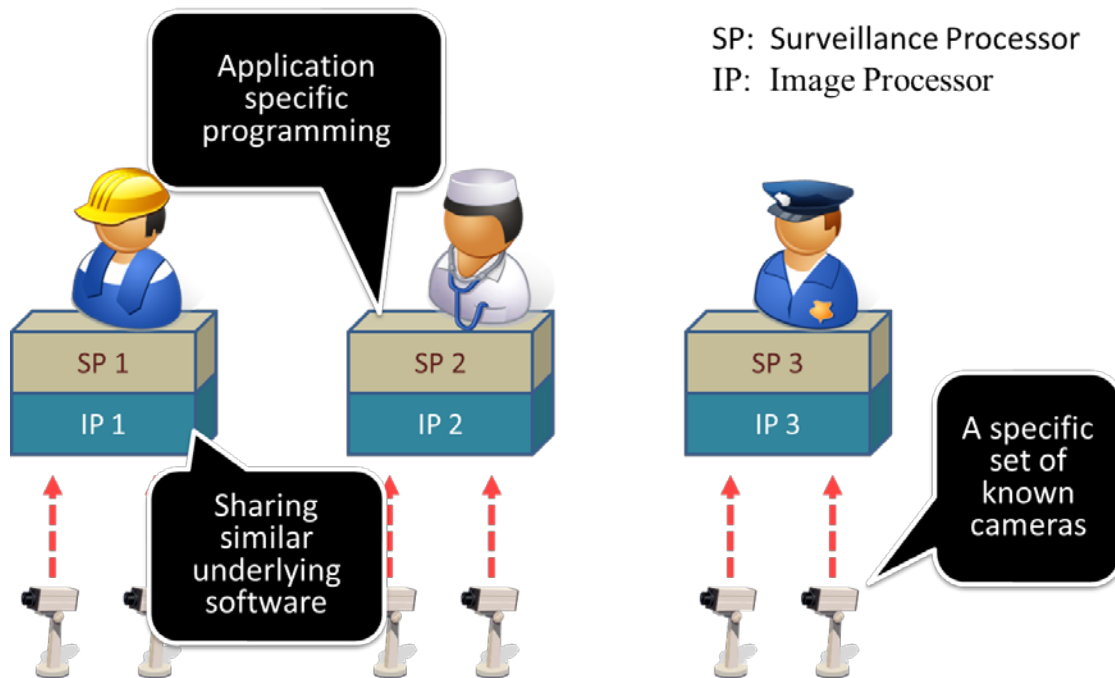


Figure 15. Siloed approach to camera/stream processing application development.

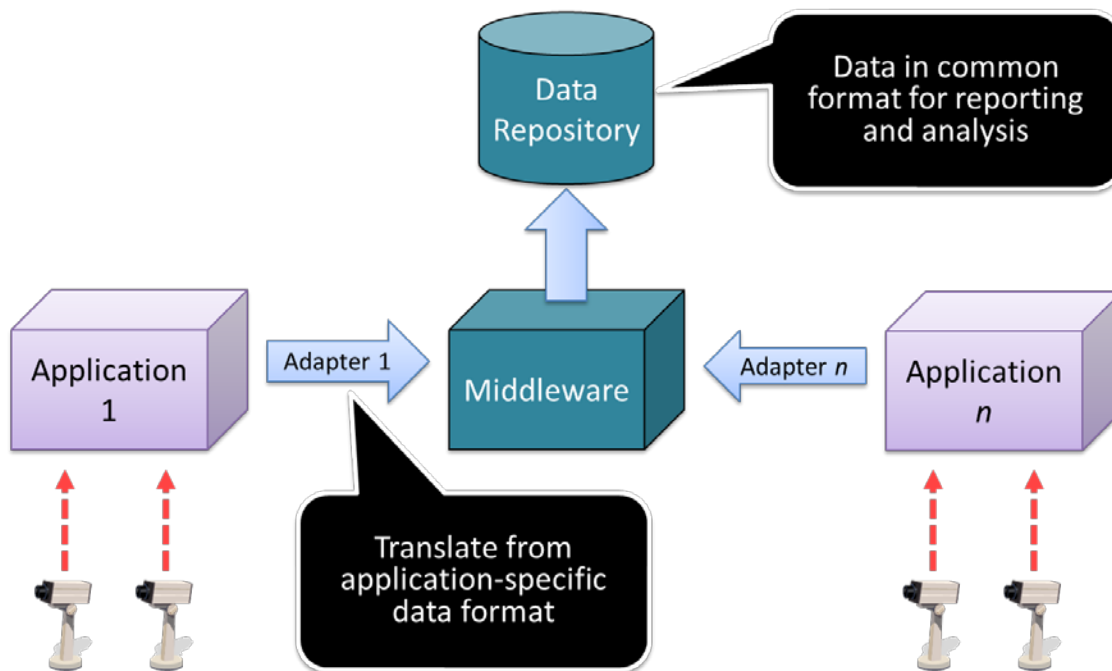
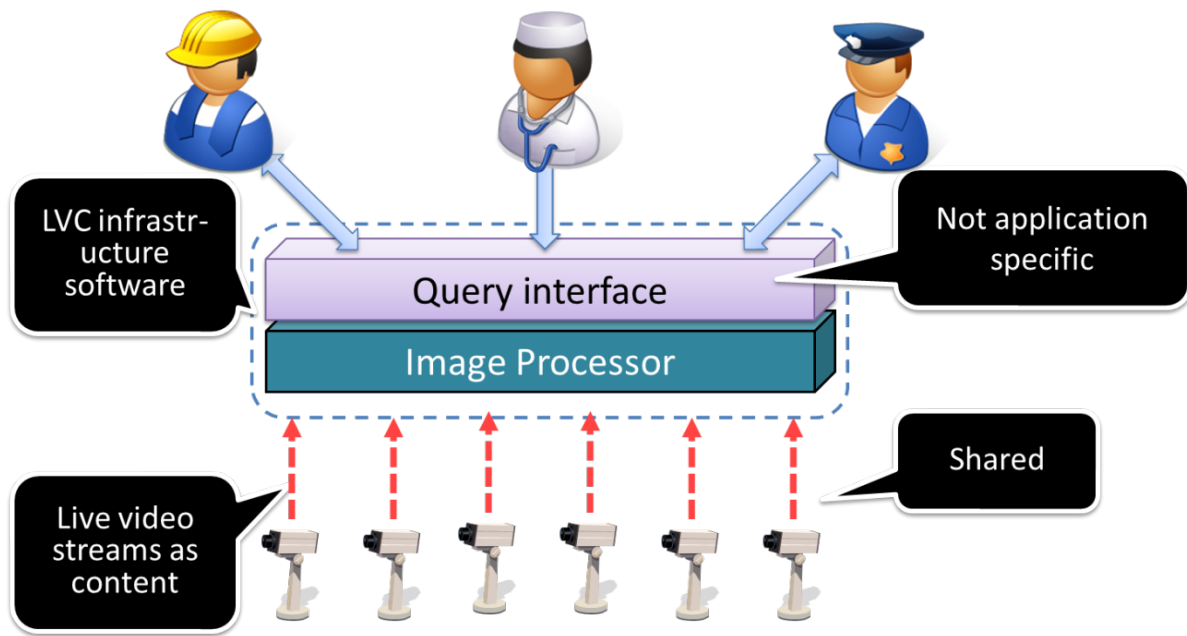


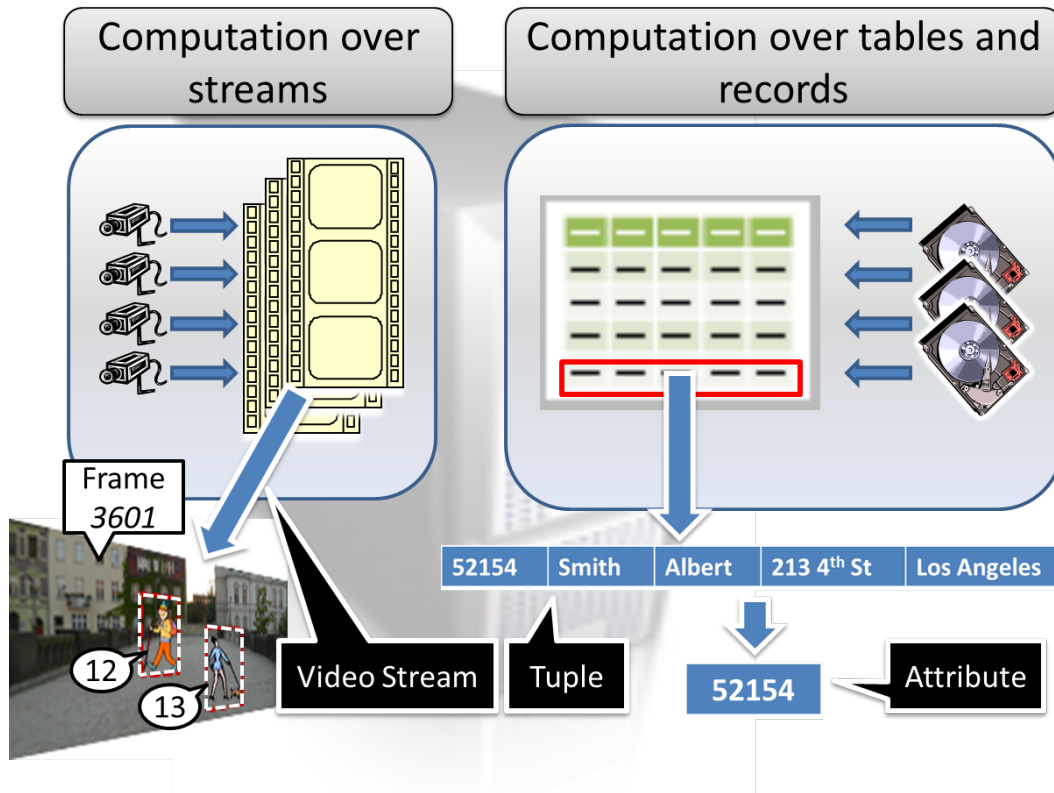
Figure 16. Applications built as information silos utilizing middleware and application-specific data adapters.





**Figure 17. LVC approach showing common image processor and query interface.**

The LVC approach leverages a common video processing software infrastructure to provide a common programmable interface to clients and a shareable pool of camera resources; illustrated in Figure 17. The goal is to create an ecosystem for collaboration and information sharing to allow users to draw new insights that are not possible with siloed information frameworks. This approach facilitates rapid application development by allowing application architects and software developers to focus their time and resources on the business problem, rather than having to devote time and resources to develop core stream processing functionality for each application. This approach is similar to how business application software leverages a common database platform; the application designers and programmers focus their efforts on the business problem and rely on the programmatic interface and SQL to persist business data and retrieve data for reporting purposes.



**Figure 18. LVC stream data model contrasted with relational record and disk data model.**

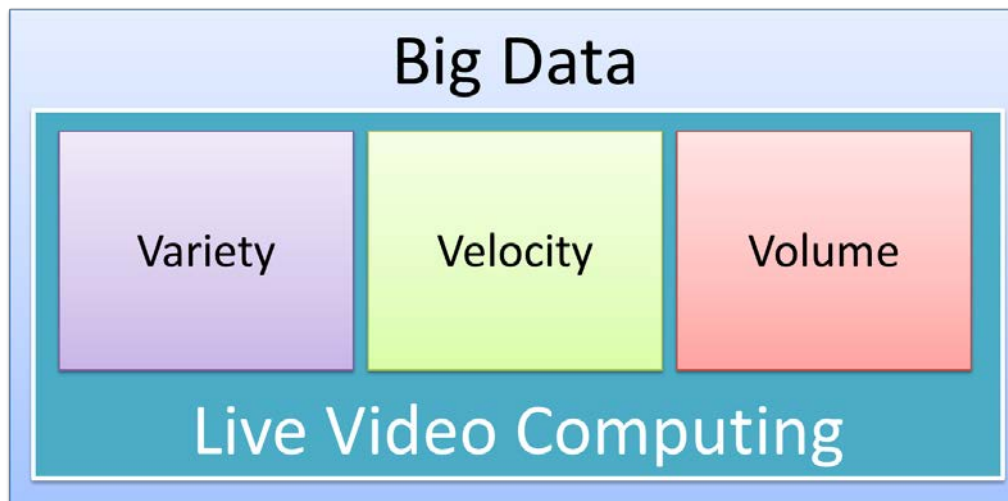
Traditional DBMSs orient data in tables, such that each table contains records (or tuples in the relational vernacular). Each record in a table has a common attribute structure, illustrated in the right side of Figure 18. LVC is stream-oriented; operating over video streams. That is, the video streams are queried their content conceptually similar to how files residing disk drives are also queried for the content they hold. A comparison of concepts between traditional database computing and LVC is presented in Table 2, extending what is presented in Figure 18 with additional comparisons of similar concepts that exist between these two platforms. *Live Video Query Language (LVQL)* is the query language of the LVC prototype implementation. It can specify events in terms of spatio-temporal observations and correlations of objects in video streams.

**Table 2: Comparison of LVC and traditional DBMS concepts**

	LVC	DBMS
Storage	Camera	Hard Drive
Relation	Video stream	Record
Data unit	Video frame	Tuple
Data granularity	Object	Attribute
Query language	LVQL	SQL

*Live Video Computing is Big Data*

The term big data is used to describe large quantities of data that exhibits complex relationships (White, 2012). For example, sensors on an airplane measure numerous different physical properties, and the promptness with which actionable results can be derived from them is imperative. Other examples include measurements pertaining to particle collisions in physics (Dimopoulos & Landsberg, 2001). Due to the massive storage and computation requirements inherent to working with large and complex data sets, traditional software solutions such as relational databases and desktop computer statistical packages such as SPSS (Norusis, 1990) are not well suited to be applied to big data problems.



**Figure 19.** LVC exhibits the three popular attributes of big data; variety, velocity and volume.

In a Gartner report (M. Beyer, 2011), Beyer identifies three distinguishing properties for describing big data; they are namely volume, velocity and variety. LVC also exhibits these same attributes (Figure 19). LVC is concerned with providing real-time responses to clients (velocity) based upon the present situation that is observed. Video streams are treated as an unending sequence of frames of imagery (volume) and the cameras can be placed to monitor a wide variety of situations ranging from war scenarios to Mars exploration to airport surveillance (variety; i.e. the imagery data is diverse and not structured).

A number of data processing algorithms have been applied to extract information and understanding from big data, including spatial analysis, sentiment analysis, neural networks, cluster analysis, supervised and unsupervised learning, etc. LVC utilizes a distributed computing approach that includes database and real-time stream processing concepts. Whereas traditional data mining frameworks and algorithms operated on large static data sets, LVC leverages real-time data streams to allow end users (i.e. not programmers in an Information Technology department) to leverage real-time data for decision making and notifications.

### Summary

In order to achieve a solution that can leverage video streams to gain information and insight in an automated fashion, a number of components from a number of different fields of computer science, mathematics and statistics must be combined and made to operate in harmony. This chapter provides a brief review of a number of areas that are fundamental to engaging in computing over live video streams and processing the temporally oriented sequences of images they contain.

Early multimedia database systems leveraged manually entered metadata annotations in order to serve mixed media files to users and client applications, for purposes such as education, finding representative content for television news stories, managing personal music collections, etc. As computer processing and storage capacity advanced, more information and knowledge could be mined from the bits and bytes that represent the raw multimedia data. This led to algorithms to segment images into logically similar sub-regions, algorithms to represent portions of media as feature vectors for indexing and similarity computations, methods to find the salient objects depicted in video streams, and to track the identified objects as they move about the scene. This chapter also includes a review of more “high-level” techniques, such as data clustering, shot boundary detection, indices for efficient comparison of high-dimensional features, and the semantic gap.

Finally, the concept of LVC and sharable data and infrastructure is introduced, and the application of the term “big data”; the concept of leveraging massive quantities of both stored and real-time information to provide for real-time decision-making based upon said data.

## CHAPTER 3: INTRODUCTION TO THE LVDBMS

LVC refers to theoretical aspects of the live video computing framework. The LVDBMS is an implementation of the LVC framework with objectives that include algorithm and data structure development. It is intended to be scalable and adaptable to manage a plethora of video streams while providing real-time query responses. As surveillance scenarios can be uninteresting and monotonous to watch, maintaining a high level of vigilance for long periods of time, waiting for an event that occurs infrequently is fatiguing. Operators can become biased, become distracted, may be required by law to take periodic rest breaks, etc. Additionally, due to the lowering costs of camera hardware, the pervasiveness of network infrastructure (both wired and wireless) and the growing needs for maintaining security (for safety purposes, theft prevention, etc.) the number of video streams that must be monitored is increasing. The human factor limitations and the vast quantity, variety and velocity of videos that need to be monitored contribute to the fact that many camera networks are relegated to be used primarily for data archiving and post-crime investigations. As a result, in some instances of camera network deployments, police are no more likely to catch criminals in places where numerous cameras are deployed, compared to areas where only a few cameras are deployed.

This work presents efforts pertaining to contributions to LVC and more specifically development efforts pertaining to the LVDBMS. Note that a portion of this content, including the results presented here, have previously been published by the author in conference proceedings and in journals, for example, (Aved, Hua, & Gurappa, 2011; Aved & Hua, 2012; R. Peng, Aved, & Hua, 2010). Select highlights of the LVDBMS include:

- A *data model* characterizing different classes of objects.

- The *LVQL query language* which can specify events of interest. LVQL queries are posed to the LVDBMS as continuous queries and may be associated with an action (such as “notify the operator”) when an event is detected.
- A *privacy specification language (PSL)* for specifying privacy policies. Privacy policies apply to objects observed in video streams and specify when their appearance should be redacted from output streams.
- A *4-tier architecture* of loosely connected layers that implement web services communication interfaces amenable to rapid application development and porting to other environments such as a public or private cloud.
- A *framework to match objects observed in multiple video streams*. The essence of this framework models objects as bipartite graphs. Objects are recognized by applying a distance function and threshold to the graph.
- *Runtime query optimization* to reduce and potentially eliminate duplicate computation of intermediate query results.
- An *LVDBMS prototype* implementing said functionality presented with corresponding performance results; both qualitative (tracking precision and recall) and quantitative (index maintenance overhead, etc).

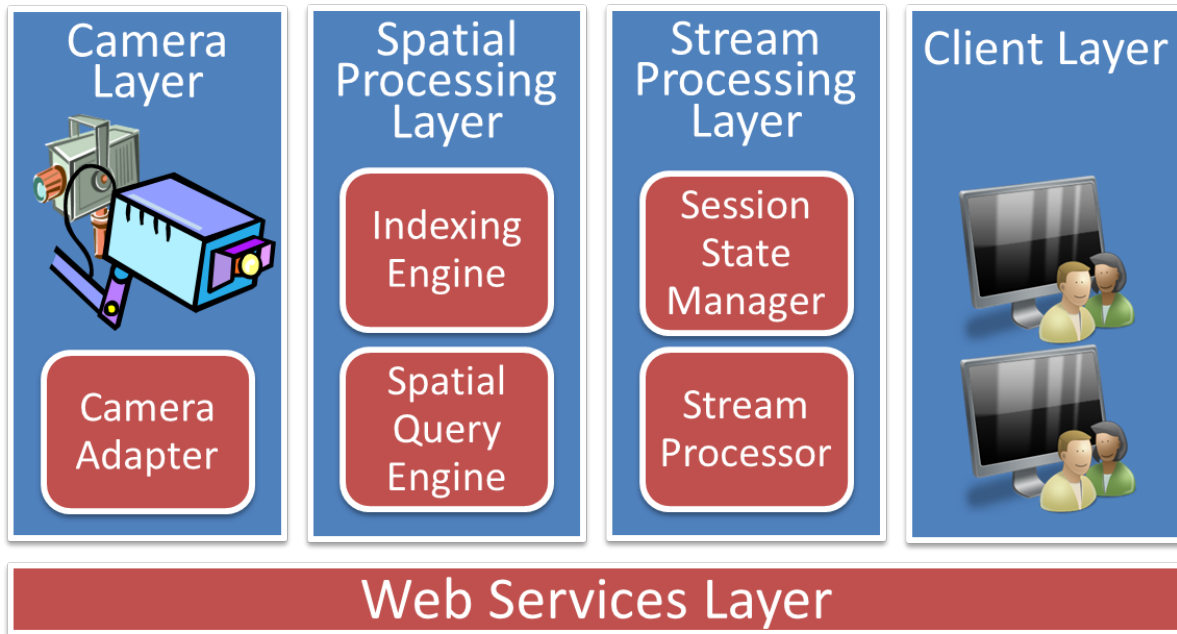
The remainder of this chapter provides details of select functionality and the LVDBMS environment. Later sections expand upon specific functional areas such as query optimization and privacy filter specification and implementation.

## LVDBMS Architecture

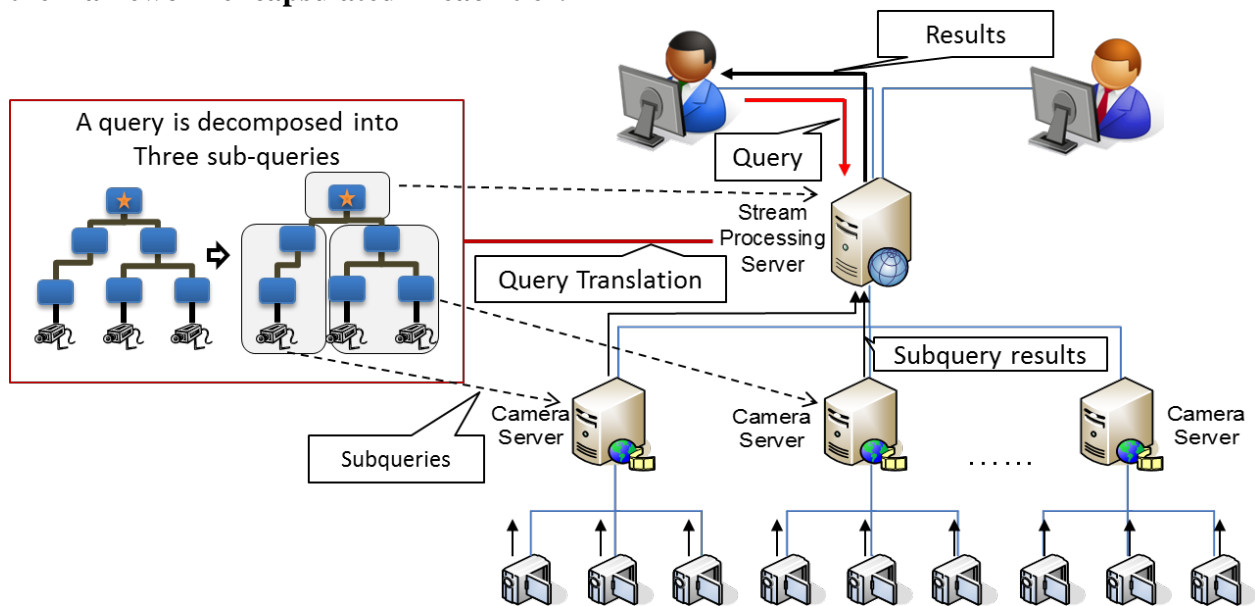
The components of the LVDBMS are logically grouped into four tiers, as illustrated in Figure 20. Each tier defines one or more web service interfaces to facilitate communication between the tiers. The four tiers include:

- The *camera layer*, which encompasses cameras and their corresponding adapters. Camera adapters are conceptually similar to device drivers in computer systems, allowing for disparate camera device hardware to connect with a standard LVDBMS interface.
- The *spatial processing layer*, which processes the metadata and video streams from the camera adapters, passing results to the stream processing layer. A host in this layer communicates with multiple camera adapters, but a camera adapter communicates with only a single spatial processing layer host.
- The *stream processing layer* receives subquery evaluation streams from spatial processing layer hosts and computes final query results for delivery to clients. As this interfaces with end users and applications (i.e. the client layer), it contains logic for managing authentication, connections and session state with LVDBMS clients.
- The *client layer* encompasses LVDBMS end users and client applications. Clients authenticate and interact with the LVDBMS by browsing the catalog of cameras, submitting queries and receiving query results. Representative images of the LVDBMS *graphical user interface (GUI)* are depicted in Figure 22. Queries are specified in the area “Query Description” and buttons “Query 1”, “Query 2”, etc. recall pre-written queries. The “Send Query” button submits a query for evaluation.

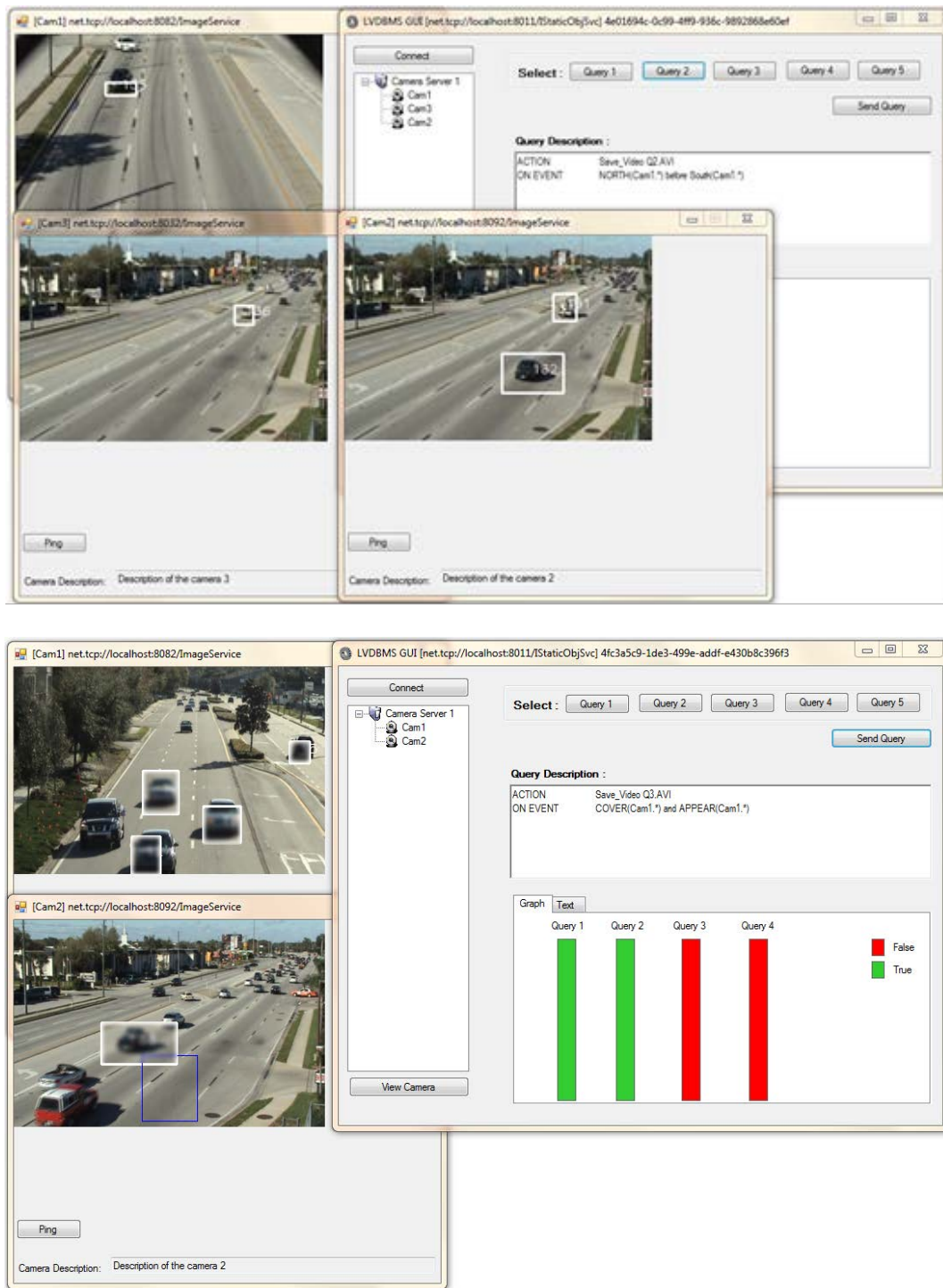




**Figure 20. Logical 4-tier architecture of the LVDBMS prototype and major components of the framework encapsulated in each tier.**



**Figure 21. LVDBMS prototype, illustrating query, subqueries and results.**



**Figure 22. Example images of the LVDBMS; the appearances of some objects have been redacted.**

The LVDBMS illustration depicted in Figure 20 is refined in Figure 21, which illustrates how a query flows down through the LVDBMS architecture, and then how data and query results flow back up through the layers and back to the client. An initial query is posed by an end user or client application to the LVDBMS. This initial query is submitted to the stream processing layer host to which the client is connected. The stream processing layer host maintains metadata pertaining to available spatial processing layer hosts (also referred to as camera servers, as they interface with cameras via their adapters and perform processing) and their associated cameras. With this information the stream processing layer host translates a query into one or more subqueries. Each subquery corresponds to a particular camera server host, where it will be sent for evaluation. Camera adapters process imagery from camera sensors and translate it into a stream of images and corresponding metadata, which is sent to its respective camera server. Metadata associated with each video frame from the camera adapter includes information pertaining to the frame itself (i.e. timestamp, sequence number, etc.) and to objects observed within the frame and segmented out by the camera adapter (i.e. object identifier, a bounding box identifying the location of the object within the frame, etc). Subqueries evaluate LVQL expressions over video streams (specifically, over the intersection of video streams specified by the query and video streams managed by a particular camera server to which the subquery was sent) and stream subquery evaluation results back to the respective stream processing layer host. The stream processing layer host receives one or more intermediate results for each evaluation time step and computes a final query result (for the particular point in time), which is then delivered back to the end user or client application.

### LVDBMS Data Model

LVC, and correspondingly the LVDBMS, is concerned with computation over video streams. As such, the event and data models revolve around objects that are observable by imaging sensors and depicted in temporally oriented frames in the video streams that emanate from these sensors. Therefore, it follows that an event (i.e. a *simple event*) is defined to be occurrence of an action that may be observed by one (or more) cameras and represented in frame data in corresponding data streams. We note that in this work, the terms video stream and camera stream are used interchangeably, as are enabling hardware device terms such as camera and imaging sensor.

From the perspective of an LVDBMS client, events may be specified in LVQL by using a combination of spatial and temporal components, or operators. Thus, a user can leverage LVQL to specify a *complex event* in terms of simple events that are related temporally. For example, a simple event could be a person (or more generally some object) appearing in a scene or moving in front of a desk (where the term scene refers to some portion of the real world that is observed by a camera and rendered into a sequence of frames in a video stream). A complex event relates simple events with temporal operators. For example, a complex event could be defined as a person first appearing in a scene and then, within some threshold of time, moving in front of a desk. (Since the LVQL presented in this work is 2-dimensional, there is no distinction between touching and in-front-of, as that type of scene information is not captured by the cameras.)

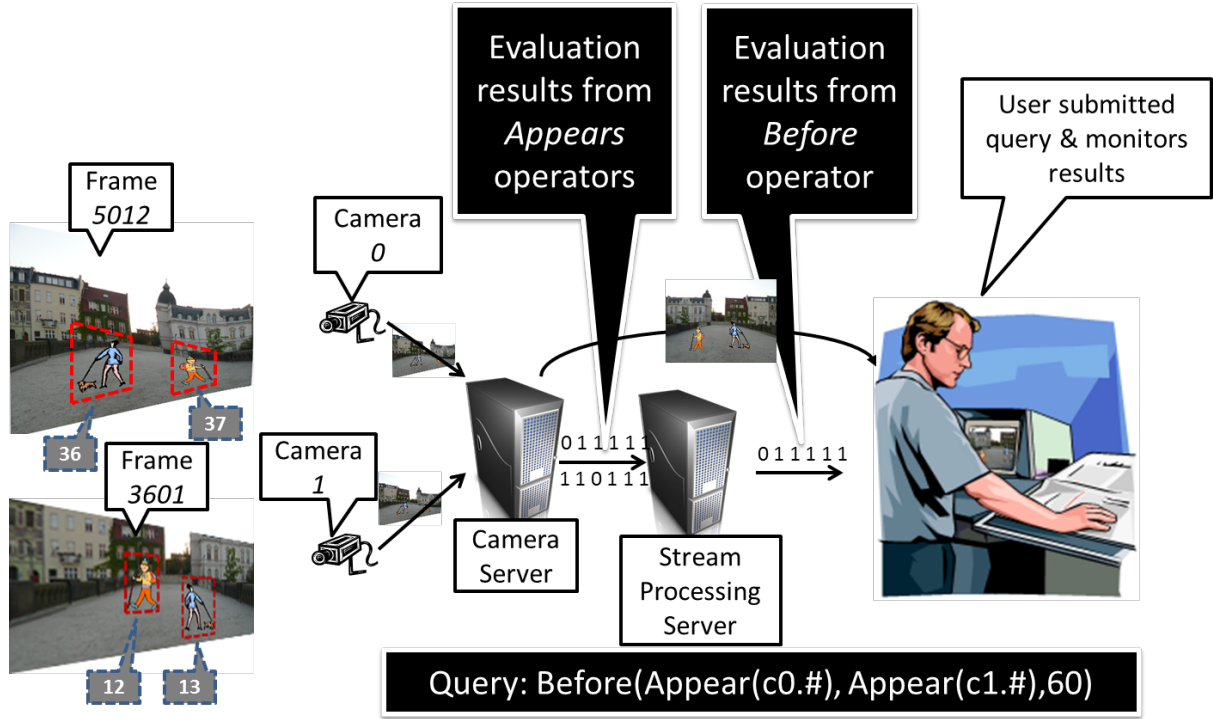
A spatiotemporal query is formulated in LVQL. This query specification defines which video streams will be monitored for the occurrence of an event. That is, if the query specifies that a particular video stream will be monitored for the appearance of an object, if an object subsequently appears in a different video stream, there will be no impact upon the query result.

An object is a fundamental component of an event specification. As indicated in Table 3, there are two basic types of objects that are recognized: dynamic objects are detected automatically by the image processing software, and static objects are indicated by users of the system. The third classes of objects are cross-camera dynamic objects. These are dynamic objects that were first recognized in one video stream and subsequently recognized in a second stream. The inclusion of this object class simplifies the expression of queries that define events correlating objects that appear in multiple video streams. Note that in each respective stream these objects also qualify as dynamic objects.

**Table 3: Comparison classes of LVC objects**

Object class	Description
Static	Objects of this class are defined by the user and do not move within the scene. For example, a static object may be defined (drawn) over a window or door for subsequent use in a query.
Dynamic	Salient objects that are detected automatically within a video stream. A model of the scene background is maintained and as an object passes through the scene, its appearance is distinguished from the background. If its size is beyond a threshold it is segmented, assigned a unique identifier and tracked.
Cross-camera dynamic	Static objects detected in one video stream and subsequently matched to an object in a second video stream are classified as cross-camera dynamic objects.

Another view of the data flow in the LVDBMS is presented in Figure 23. Starting from the left, two cameras observe the same scene from different vantage points. Two frames are depicted at a particular point in time, from the two cameras. Within the scene two objects are observed, assigned identifiers (unique to the video stream) and tracked within their respective video streams. Within each stream these objects are dynamic objects. However, a query may be defined specifying an event that involves both cameras; for example, the event may be that an object appears in both the first camera stream and then the second stream. In that case, these two objects are also considered to be cross-camera dynamic objects.



**Figure 23. Depiction of the data flow in the LVDBMS, from frames to query result.**

Continuing with the example in Figure 23, the scene is segmented and objects are tracked by each camera's respective camera adapter (not shown) and sent to the camera server, which resides in the spatial processing layer. The camera server uses the metadata received from the camera adapter to process the spatial operators and send the stream of results to the stream processing server residing in the stream processing layer. (Note that the pound sign in the operand to the *Appear()* operator corresponds to an object of type cross-camera dynamic; if it simply had a dynamic operator (denoted by an asterisk) the query would not correlate objects across camera stream.) The final query result is streamed to the user from the stream processing layer host. When a user requests to monitor the imagery from a video stream, the images come from the camera server. This allows the user to observe the same images in sequence with query evaluation results and eliminates a potential capacity bottleneck if multiple users view images

from the same camera simultaneously. Also, by serving the pictures from an LVDBMS host (rather than the camera hardware directly), authorization information pertaining to the user's session may be consulted in order to determine if the user should have direct viewing access to the raw imagery of the scene.

### Introduction to the LVQL Query Language

LVQL is the query language of the LVDBMS. Analysts and programmers may leverage this query language in to develop applications that interact with video streams. As such, the programmers and application designers need only know the details of the query language, and do not need to spend time developing stream processing algorithms or low-level details of the LVDBMS. LVQL permits for the specification of an event and a corresponding action to be defined over a video stream (or a set of video streams). It is a declarative language, meaning that the user defines a logical event specification and not the particular flow of control or algorithms that will be executed to determine the query result. An LVQL expression specifies a spatio-temporal event, and an action that is to be triggered when the event is recognized. The basic form of a query (specifically, an *ActionEvent*) is as follows:

ACTION *UserSpecifiedAction*

ON EVENT *EventSpecification*

Which signifies that an action *UserSpecifiedAction* corresponds with *EventSpecification* and will be executed the first time a query evaluation result of *true* is returned. *EventSpecification* is an event specification that is generated by a context free grammar which consists of a set of rules, or productions, which can be utilized to express (describe) an event. A simplified set of LVQL

productions is presented in Figure 24; items shown in light blue represent tokens recognized by the language.

As shown in Figure 24, an LVQL statement consists of either an *ActionEvent* or a *View Definition Language* (VDL) production. In the case of an *ActionEvent*, which specifies a query, the event definition must contain a spatial operator (e.g. *Appear*, *North*, *Meet*, etc.) The VDL is used to define privacy filters and views over video streams, and is discussed in detail later in this work.

Declaring an event in LVQL entails expressing the event in terms of spatial, temporal and Boolean operators. The simplest event that can be expressed is the appearance of an object in a video stream by using the *Appear()* operator. The *Appear()* operator accepts two arguments (i.e. operands), the first operand specifies the video stream, object class (and possibly filter criteria) that the operator will be applied to, and the second is a threshold. (All spatial operators accept a threshold argument.) The threshold for the *Appear()* operator specifies the minimum size of an object that will satisfy the appearance condition, in terms of the area of the *minimum bounding rectangle* (MBR) that contains the object. For example, *Appear(sl.\*, 200)* will return true each time it is evaluated if a dynamic object with an MBR of area greater than or equal to 200 is observed in the current video frame. In the case of a spatial operator such as *North()*, three arguments are accepted; the first two correspond to objects in the video stream, and the third is again a threshold. *North()* returns true if the object specified by the first operand is above the object specified by the second operand, in a stream. The third argument, the threshold, specifies the amount of separation between these objects (i.e. the distance between the bottom of the upper object's MBR and the top of the lower object's MBR). For example a value of 10 pixels means



the upper object must be at least 10 pixels above the lower object. Note that this threshold can be negative, allowing MBRs to overlap.

```

Lvql := ActionEvent | VDL
ActionEvent := [action UserSpecifiedAction] on EventSpecification
EventSpecification := NotSpTmplEvent ( BooleanOperator NotSpTmplEvent )
NotSpTmplEvent := [not] SpatialTemporalEvent
SpatialTemporalEvent := CompositSpatialEvent | CompositTemporalEvent
CompositSpatialEvent := appear | north | northwest | inside | meet | ...
CompositTemporalEvent := before | meets
BooleanOperator := and | or | not BooleanOperator
VDL := VCmdType view ViewIdentifier over VStreamIdent [ set VPrivFilter ]
VCmdType := create | update | delete
VTargetStmt := target eq ( querytargets | nonquerytargets |
    previouslymasked | none )
VTmpScpStmt := temporalscope eq ( querynonactive | queryactive |
    permanent | none )
VObjScpStmt := objectscope eq ( static | dynamic | crosscameradynamic
    | none )
VStreamIdent := ( Cameraidentifier | ViewIdentifier )
Cameraidentifier := camIdent
ViewIdentifier := viewIdent

```

Figure 24. A simplification of the LVQL grammar, including privacy *view definition language* (VDL) productions.



Figure 25. Illustration depicting dynamic and static objects.

Spatial, temporal (and Boolean) operators can be combined by using Boolean operators. For example, to identify objects that are between 200 and 300 pixels of area in a stream, one could formulate the expression *And(Appear(s1.\*, 200), Not(Appear(s1.\*, 300)))*. Note that operands effectively act as data filters. For example, three types of objects can be associated with video streams; dynamic, static and cross-camera dynamic. Specifying an operator and providing it with an operand argument of one of these types will return only objects of that respective type to the operator. For example, given a video stream with a static object defined in it, say, *s1.12* (where *s1* corresponds to stream number *1* and static object number *12*), the operator *Appear(s1.12, 50)* will only return true if the static object *12* is greater than 50 pixels in area and will ignore any dynamic and cross-camera dynamic objects that might be detected in the stream.

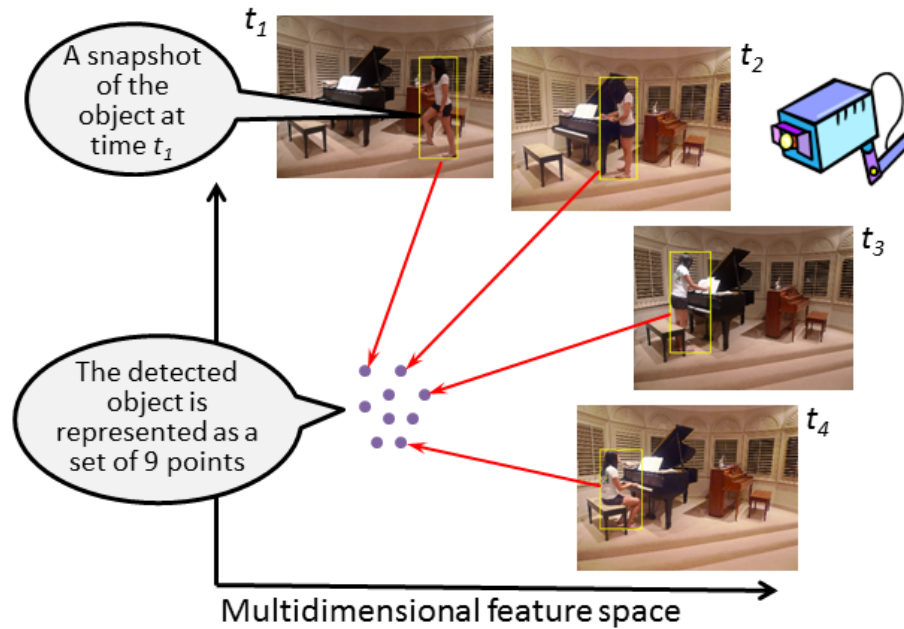
### Summary

This chapter presents an introduction to the LVDBMS. While LVC refers to the theoretical computation model over live video streams, the LVDBMS is the corresponding prototype testbed implementation. By providing a computing infrastructure that is accessible to users and client applications via a web services interface by utilizing a declarative computer language, software architects and programmers can focus on solving the business problem they are posed with, and not having to focus on low-level details of stream processing. Thus one goal of the LVDBMS is to provide a platform to facilitate rapid application development.

The components LVDBMS can logically be grouped into four tiers, the lowest consisting of physical hardware. Next is the spatial processing layer, then the stream processing layer and finally the client layer. Queries originate in the client layer and are pushed down to the stream processing layer and then to the spatial processing layer. Data originates in the camera layer and

flows upwards. As it moves upwards it is transformed; from a stream of imagery in the lower layer to streams of subquery evaluations to a stream of Boolean query evaluations. Thus, the LVDBMS can scale to process more video streams by the addition of additional processing nodes in the stream and spatial processing layers.

This chapter also introduces LVQL, the query language of the LVDBMS. Spatio-temporal events can be specified in LVQL, and when the LVDBMS observes a specified event in video streams corresponding to the query definition, an action (such as *notify operator* or *save video*) can be executed.



**Figure 26. Illustration of an object moving about a scene and corresponding instances, in a feature space.**

## **CHAPTER 4: AN INFORMATICS-BASED APPROACH TO OBJECT TRACKING**

The subject matter of this chapter is the object modeling and recognition technique that matches objects observed in multiple video streams. This correlation of objects across multiple video streams facilitates the expressiveness the LVQL. This chapter discusses the techniques used to model an object and the distance function that realizes the object recognition.

### Introduction

The multi-camera object tracking implementation in the LVDBMS is based upon concepts gleaned from MIL. This tracking implementation differs from the MIL introduction in a previous chapter due to the fact that objects are not explicitly labeled negatively; each object corresponds to a bag of instances which are all positive examples. Additionally, each bag represents a distinct object (as opposed to a bag representing some larger entity like a scene which can contain many distinct concepts and corresponding feature vector representations). Thus, each bag represents a salient object that has been segmented from a video stream, and the instances correspond to the appearances of said object in distinct frames. The object is tracked from frame to frame, and as the object's appearance changes shape (i.e. a human's legs and arms would move while they walk), the instances in the bag represent different poses and possibly vantage points of the same object (for example, as the object moves about the scene they may appear larger or smaller, the illumination may change, etc). Thus these changing appearances will be represented in the bag. Note that samples do not necessarily need to be taken from consecutive frames of video; for example if an object is not moving very fast then its appearance from one frame to the next will not change significantly. See Figure 26 for an example of an

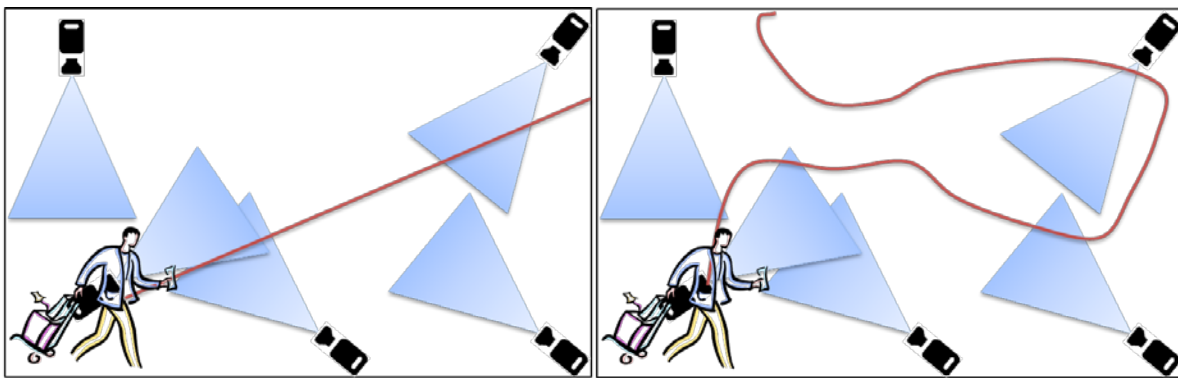
object moving about a scene and changing structure and pose as she moves about. Each appearance is modeled by a feature vector and the series of feature vectors corresponding to the observations are collected in an associated bag.

In the LVDBMS implementation the number of instances a bag may contain is bounded and instances are added and removed in a first in first out order. Thus, the comparison between two objects is based upon the minimum of the number of observations and the maximum capacity of a bag (i.e.,  $\min(\text{number\_of\_observations}, \text{bag\_capacity})$ ), and two bags do not need to contain the same number of observations in order for them to be compared (that is, for the application of a distance function). As new observations of the object are recorded, the bag is updated to include the new instances. If the inclusion of new instances exceeds the bag's capacity the oldest instances are removed. This behavior is acceptable because real-time surveillance queries are generally concerned with events that have happened very recently. In order to query for long-term historical events, a different model of data storage and retrieval would need to be utilized, as the observations of the objects may exceed the capacity of the primary memory of the computers hosting the application.

#### Previous Multi-Camera Object Tracking Work

Tracking objects across multiple uncelebrated cameras that have non-overlapping fields of view (Figure 9 right) is a difficult problem; as objects move about the terrain where the cameras are deployed the object appear to have one size when it is observed by one particular camera, and a different size or appearance when observed by a second camera due to its having a different spatial relationships between itself and the cameras. Furthermore, some length of time may pass between initial and subsequent observations. A number of multi-camera research

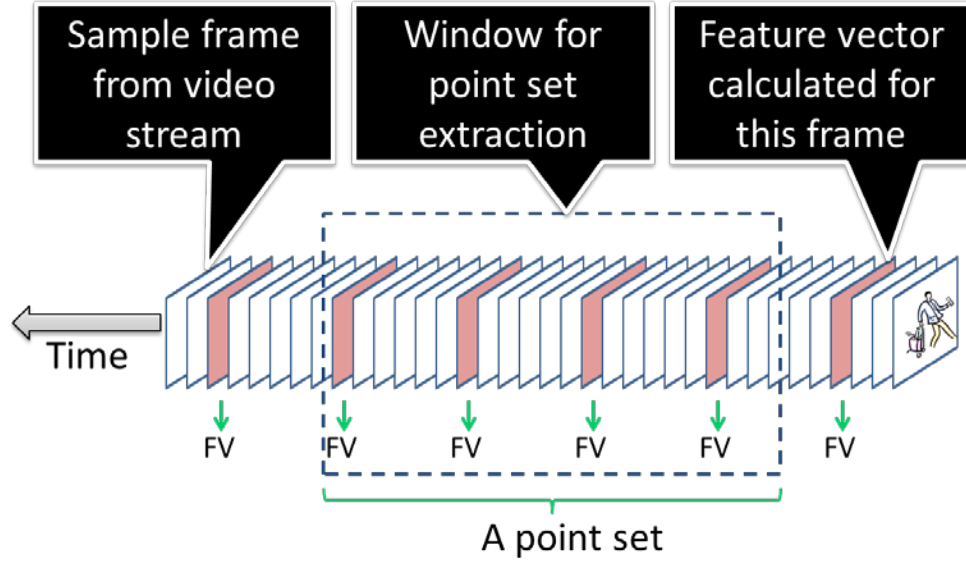
models require calibrations pertaining to the spatial relationship between the cameras, and assume either non-overlapping fields of view, or that objects in motion will maintain a consistent path between observations (Bowden, Gilbert, & KaewTraKulPong, 2006; Du & Piater, 2007; Hemayed, 2003; Hu, 1962; Javed et al., 2003; Song & Roy-Chowdhury, 2007; Tieu, Dalley, & Grimson, 2005; Yilmaz et al., 2006). Factoring in the fixed speed and track of an object as a part of the feature vector that describes an object can contribute to the precision of the system's tracking ability, assuming that the speed and track constraint is realized; e.g. (Hu, 1962; Javed et al., 2003). Thus, these assumptions (and correspondingly, systems) perform best when the object movement is non-random; i.e. when the systems are deployed within a building and the objects are constrained by corridors or walls, or when they are applied to a network such as the railroad system or road network (i.e. Figure 27 left). Also, these systems assume that the camera locations are fixed, and if they were to be moved or repositioned, the calibration phase would need to be executed again. The object tracking technique described in this chapter can accommodate unconstrained object motion, illustrated in Figure 27 (right).



**Figure 27. (left) An object moves about in a straight path, and (right) an object moves along a random path.**

### Cross-Camera Object Tracking in the LVDBMS

Salient objects observed in scenes by a camera are first tracked within the stream by an existing single-camera tracking technique, for example the technique presented in (Hampapur et al., 2005) which tracks an object based upon its appearance in a scene. We refer to the class of trackers that track objects within a single video stream as frame-to-frame trackers. The LVDBMS cross-camera tracker relies on frame-to-frame trackers to track objects within each respective video stream. Moving salient objects are identified and tracked within each camera stream. Each identified object is assigned an identifier that is unique to the stream in which it is observed. As the object moves, its appearance is captured in each frame of video. Each capture of an object's appearance is processed and a representative feature vector is computed based upon its appearance. These cumulative feature vectors for each object are stored in a bag that corresponds with the object's identifier (i.e., the bag has an identifier of the format  $\langle stream\_id, object\_id \rangle$  which is unique to each LVDBMS implementation for each object). Note that depending upon the circumstances it may not be necessary to capture feature vectors from each consecutive frame in which an object is observed; for example if an object's appearance does not change greatly between frames it may be acceptable (in terms of matching precision and recall) to capture every 2<sup>nd</sup>, 5<sup>th</sup>, etc. observation. As video streams are considered to communicate an infinite number of frames of video, the bag containing the observations (instances) of an object's appearance have a maximum number of instances they can contain. The most recently observed instances are maintained in first in first out order within a sliding window (e.g. see Figure 28).



**Figure 28. Illustration of feature vectors calculated based upon an object's appearance in every  $n^{\text{th}}$  frame.**

Initially as objects are observed within their respective streams, and objects are not correlated with objects appearing in other streams, there is a one-to-one mapping between objects and bags; that is, each bag refers to a particular object in a stream. In order to determine if an object in one stream is the same object that is observed in a second stream, the two bags corresponding to the two objects are compared by applying a distance function to compute the similarity of the bags. If two bags are similar (within some threshold) then the bags that correspond to the two objects are merged (that is, the most recent instances the bags contain are merged; adhering to the sliding observation window threshold). The merged bag is then updated with observations from the object from both video streams. This provides the mapping from an object in one video stream to an object in a second video stream. Note that in the case of multiple video streams, it is possible that an object is observed in and matched to objects appearing in more than two streams, in which case the bag would provide a mapping among each object in each respective stream in which it is observed.



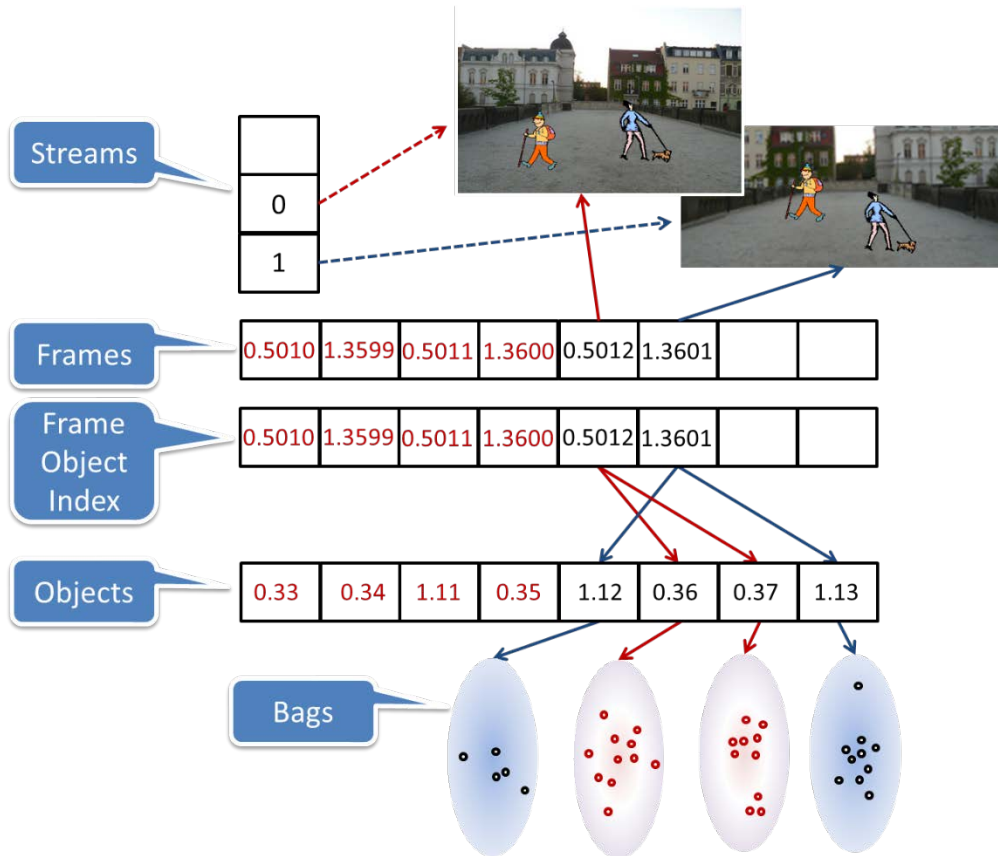
### LVDBMS Cross-Camera Tracking Implementation

This section explains implementation details of the cross-camera tracking logic implemented in the LVDBMS. The LVDBMS prototype implements specific algorithms for tracking objects across video streams; that is, specific routines that compare the unmatched bags of objects between pairs of streams. As this object matching inherently requires some CPU and primary memory resources when running, it is only executed when a query exists that searches for an event that spans video streams (specifically, the query contains a cross-camera tracking operand).

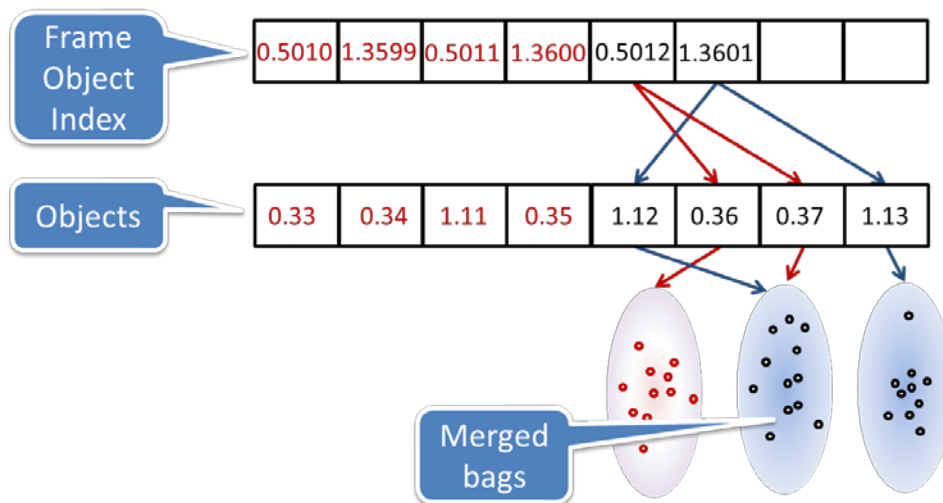
In order to reduce false matches, once an object is matched from a stream  $s_1$  to a second stream  $s_2$ , that object is no longer considered for any additional  $\{s_1, s_2\}$  matches (and symmetrically,  $\{s_2, s_1\}$ ). Also, the object matches are considered only among the streams specified by the query. For example, if four cameras (and thus streams) are registered with an LVDBMS implementation and a query specifies an event that spans two of the streams, the matching logic processes only those two specific streams for potential object matches. Note that even though the matching is a binary relationship (and the distance function is a binary function), three streams may be related by applying two spatial operators with two cross-camera operands, for example the first operand could specify the matching  $\{s_1, s_2\}$  and the second the matching  $\{s_2, s_3\}$ , and thus an object could be matched among (and a bag correlated with) three streams,  $\{s_1, s_2, s_3\}$ .

To facilitate the cross-camera tracking and retrieval of object matching correlations by queries, a number of metadata structures are maintained; major components are illustrated in Figure 29, which are implemented in spatial processing layer hosts. The “Streams” structure maintains the current video streams associated with the host, and a mapping to the current (most

recently received) frame from each stream. Streams are indexed by a number that is unique to each stream. The “Frames” structure maintains a window of recent frames for each stream, items it contains are indexed by the pair  $\{stream\_id, frame\_id\}$  where  $frame\_id$  increases monotonically with each frame received. Each object identified in a video stream is assigned a numeric identifier (again, increased monotonically) that is unique to each stream. The “Frame Object Index” provides a mapping from  $\{stream\_id, frame\_id\}$  pairs to the dynamic objects that are observed (i.e. tracked) in each frame; this is a one-to-none relationship if no dynamic objects are observed within a frame, and a one-to-many relationship if multiple objects are observed in a particular frame. The “Objects” structure maintains a mapping from objects,  $\{stream\_id, object\_id\}$ , to their corresponding bag. In the case where objects in distinct streams are associated with the same real-world object (that is, the objects are matched to each other across the streams), their respective bags are merged and this structure is updated such that both stream-object identifiers point to the merged bag, Figure 30.



**Figure 29. Metadata structures implemented in the LVDBMS to facilitate cross-camera tracking and queries.**



**Figure 30. Object metadata structure showing two objects, 1.12 and 0.37, which have been merged.**

Each instance (we use feature vector and instance interchangeably) contained in a bag can be interpreted as a data point in a multidimensional feature space (where the dimensionality of the feature space is the number of components in the feature vector). In the LVDBMS, bags have a maximum capacity. At the onset, when an object first appears in a video stream, its bag will contain a single instance. As the object is observed in subsequent frames additional instance are added to the bag.

A distance function is applied to a pairing of bags to determine their similarity; if the distance between the bags (i.e. a smaller distance means they share more similarity) is below a threshold then the bags (corresponding to objects observed in different video streams) are considered to represent the same physical object, and the bags are merged. Each bag (i.e. its centroid) can be mapped to a point in a feature space, and the assumption is that bags corresponding to similar objects (based upon their modeled appearances) are located closely in this feature space (i.e. cluster analysis). The pairing of bags that will be matched can be modeled as a bipartite graph (Zha, He, Ding, Simon, & Gu, 2001). Note that as the bags initially contain only one instance and an object may appear in one stream before it appears in another, bags with differing number of instances may be compared. The actual comparison is dependent upon the particular distance function that the system is configured to apply, however, there is a system-defined minimum threshold such that if a bag contains fewer instances than this “lower watermark” it will not be considered for matching. The idea being that if a bag contains too few instances, the instances it contains may not be sufficiently representative of the object for cross-camera matching purposes.

More formally, the stream matching problem can be formulated as follows. Given the appearance of an object  $O_a$  in stream  $\alpha$ , the problem is to determine if some object  $O_b$  appearing

in some stream  $\beta$  correspond to the same physical entity. The corresponding query is approximately formulated as *when some object  $O_a$  appearing in  $\alpha$  correlates with an object  $O_b$  in stream  $\beta$ , execute the specified action*. Thus, the search space for the object matching can be constrained to objects observed in streams  $\alpha$  and  $\beta$ , and at issue is how to measure the similarity between  $O_a$  and  $O_b$ .

Let  $A$  and  $B$  represent point sets to  $O_a$  and  $O_b$ , respectively, such that  $A = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_k\}$  and  $B = \{\vec{x}'_1, \vec{x}'_2, \dots, \vec{x}'_{k'}\}$  where  $\vec{x} \in O_a$ ,  $\vec{x}' \in O_b$ ,  $k = |O_a|$ ,  $k' = |O_b|$  and  $k$  is not necessarily equal to  $k'$ . Let  $G$  be a bipartite graph  $G = (V, E)$  where  $V = A \cup B$  and  $V$  consists of the vertices in  $G$  and  $E$  the edges. There are a number of distance functions that can be applied to  $G$ , for example, one may measure the distance between the two farthest points (the point in  $A$  and the point in  $B$  that results in a maximal Euclidean distance), the distance between the two closest points; the distance from the centroid of each point set, etc. The *normalized distance* between the two point sets may be defined as follows:

$$Dist(O_a, O_b) = \frac{\sum_{e \in E} \sqrt{e}}{|V|} \quad (7)$$

Such that the sum of lengths of edges in  $E$ , is the minimum possible when considering all mappings from  $A$  to  $B$ . Objects are matched from  $\alpha$  to  $\beta$  (without loss of generality) by comparing each unmatched object in  $\alpha$  to each unmatched object in  $\beta$  that exist (i.e. may be observed) in the current frames of each respective stream. Therefore, if all objects in both streams were to be unmatched at some particular time, the number of comparisons taking place would be at most  $(|\alpha| + |\beta|)^2$ , where  $|\alpha|$  and  $|\beta|$  are the number of dynamic objects in the latest

frame of each respective stream. The Hungarian algorithm (Kuhn, 1955; Mills-Tettey, Stentz, & Dias, 2007; Munkres, 1957) may be used to determine the edges in  $E$  to which the distance function is applied. Note that in the circumstances that  $k \neq k'$ , 0 vectors may be injected, or the farthest instances in the set with larger cardinality may be ignored, etc.

Another distance metric is the *closest point distance*, which is based upon the Hausdorff (Huttenlocher, Klanderman, & Rucklidge, 1993) distance; it finds the Euclidean distance between two points in two point sets that are the closest to each other:

$$Dist(A, B) = \min_{a \in A} \min_{b \in B} \|a - b\| \quad (8)$$

such that  $A$  and  $B$  are point sets and  $\|\cdot\|$  represents the Euclidean distance.

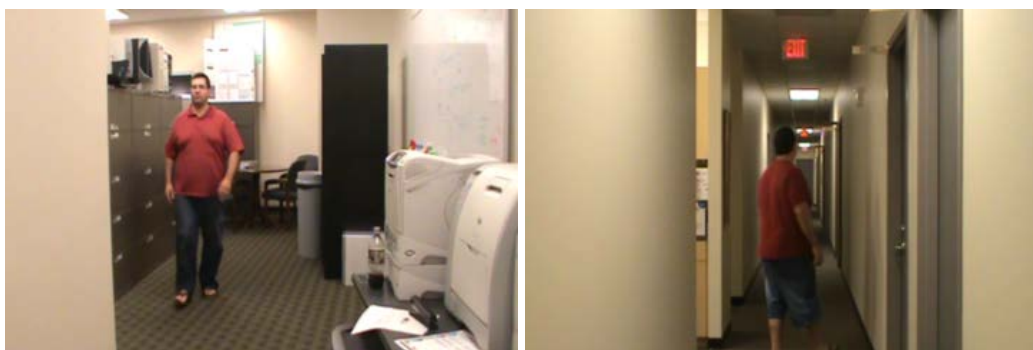
### Performance Evaluation

To evaluate the performance of the cross-camera tracking, videos from the CAVIAR (R. Fisher, 2011) project are utilized. The CAVIAR collection of videos show a number of different scenarios observing humans (leaving an object, walking, fighting, etc.) from multiple vantage points. These scenes shown from two perspectives are used in this study to measure the cross-camera tracking functionality and gauge the merits of the bag comparison technique when an object is observed from different angles. The results presented indicate that the bag comparison is robust to these differences in angles, and also the slight differences in scene lighting observed in these videos. Note also that the timing of occurrences of objects in these videos are not temporally aligned, which also would simulate an object being observed by one camera and then later observed by a second. In addition to the CAVIAR videos (Figure 31), some scenes were created to test specific scenarios; these additional scenes have a first camera positioned in a room

and a second camera located in a hallway. These scenes have different backgrounds but the same objects moving about in them, being observed from different angels, e.g. see Figure 32. Note also that the illumination is different between the two scenes.



**Figure 31.** Sample frames from CAVIAR dataset; *walk3* (left) and *OneShopOneWait2Cor* (right).



**Figure 32.** Sample video frames from two cameras, one in a room and the second outside the room (*hallway sequence*).

This video sequence was recorded with only a few people as to avoid segmentation and tracking errors that occur when numerous people are observed in crowds. The cross-camera technique presented here is designed to track single objects; when people appear in crowds errors related to segmentation and object extraction occur, for example due to obstruction.

### *Evaluation Scenario Setup*

Although the LVDBMS is designed and optimized for live video streams, evaluations are conducted with prerecorded videos in order to permit experiments to be repeated with deterministic input so system parameters may be adjusted or debugged. In the LVDBMS data flow hierarchy, a sequence of video frames comes from an image sensor and is transmitted to the associated camera adapter, where background modeling, object segmentation and tracking routines are executed. An OpenCV function is utilized to grab each frame of video in the initial processing stage. Whether that video frame is initially written to a memory location by a hardware camera driver or by a video decoding codec, is indistinguishable to the OpenCV frame grabbing function return value and is indistinguishable to LVDBMS layers above the camera adapter. Thus the use of pre-recorded video sequences does not add to or detract from the performance of the various algorithms implemented and results presented.

For the experiments presented here, the minimum object size is specified to be greater than 200 pixels, in order to not consider artifacts due to background subtraction errors or slight camera movements, for example. A feature vector of 21 dimensions is used; this includes 18 components coming from a histogram representing the object in the RGB color space and three components based upon Hu moments (Hu, 1962). These moments are popular because they are straight-forward to compute and are invariant to rotation, scale and translation. (Of course, a downside to using features that are invariant to rotation, scale and translation is that they can no



longer distinguish objects based upon those factors, for example, they cannot distinguish if one object is larger than another, etc.)

### *Evaluation Based Upon Relative Quality*

The goal is to improve the quality of data, in this case the quality of the query evaluation results that entail cross-camera object matching. In order to improve this process, it must be measured in order to establish a baseline measurement and so that improvements can be quantified. Surveillance systems can be optimized for different performance criteria than general-purpose multimedia retrieval applications, and to support this distinction we present the *Relative Quality (RQ)* metric. More specifically, this distinction pertains to the treatment and priority of *false positives (FP)* and *false negatives (FN)*. In traditional multimedia retrieval applications, a false positive can be ignored by a human user retrieving content and an FN may mean the user needs to conduct additional search iterations or refine query keywords. Often the FN and FP rates are tunable, for example, one might increase one at the cost of reducing the other, etc. However, in a security scenario, an FN may mean that an event of interest has occurred and the system did not perform the requisite action. Also, given the importance of not wanting to miss the occurrence of an event, a higher FP rate may be acceptable. Thus, the RQ metric introduced in this section is parameterized to distinguish between the FP and FN by tuning two parameters,  $\alpha$  and  $\beta$ . This metric can be used when adjusting system parameters to gauge their effect in terms of meeting a predefined threshold in terms of recognition accuracy, computation overhead (i.e. configuring different feature vector lengths and components), etc.

The performance results presented in this section are presented based upon the common understanding and usage of FN, FP, *true positive (TP)* and *true negative (TN)*. If an object is queried and it exists in the data store and is returned by the query, TP is incremented, else FP is

incremented. Likewise, if the object does not exist in the data store and is returned, FN is incremented, else TN is incremented. Thus, RQ, precision and recall can be related in terms of these components as follows:

$$Relative\ Quality = \frac{TP}{\alpha \cdot FP + \beta \cdot FN + TP} \quad (9)$$

$$Precision = \frac{TP}{TP + FP} \quad (10)$$

$$Recall = \frac{TP}{TP + FN} \quad (11)$$

Such that  $0 \leq \alpha \leq 1 \leq \beta$  and when  $\alpha = \beta = 1$ , RQ is Accuracy, i.e.  $TP/(FP+FN+TP)$ . For a discussion of the Accuracy metric the reader is referred to the discussion in (C. W. Fisher, Lauria, & Matheus, 2009).

#### *Performance of Cross-Camera Tracking*

This section presents cross-camera tracking results in terms of precision and recall. The first set of results, Table 4, are based upon input from the CAVIAR video *OneShopOneWait2cor* and correspond to plots shown in Figure 33. Note that data shown in Table 4 is sampled at 10-second intervals. In this image sequence people are entering and leaving the field of view observed by the camera and the peaks in the plots correspond with people entering the scene. In Figure 33 the results are plotted, first fixing  $\alpha$  and changing  $\beta$ , and then fixing  $\beta$  and changing  $\alpha$ . This illustrates  $\alpha$  and  $\beta$  can be adjusted to shift (skew) the results in terms of FN and FP so that system parameters can be adjusted in order to achieve target error rates.

**Table 4. CAVIAR video *OneShopOneWait2cor***

TP	FP	FN	Precision	Recall
223	38	0	0.85	1
285	73	0	0.79	1
391	108	0	0.78	1
492	146	0	0.77	1
588	191	0	0.75	1
706	230	0	0.75	1
759	265	0	0.74	1
844	314	0	0.72	1

Next, performance results are presented for a second CAVIAR video, *Walkers*. Which depicts people walking about a scene that is observed by two cameras. Select system parameters for this evaluation scenario are presented in Table 5, and evaluations with both the normalized distance and the closest point distance are presented in Table 6 and Table 7. In this video sequence between one and five people may be observed in the field of view and results are presented at ten-second intervals. Note that the *timestamp*, *running precision* and *recall* columns are cumulative. The *number of bags* column depicts the total number of bags in the index, corresponding to both videos. *Index maintenance* provides wall-clock time, in milliseconds, of the time required to maintain and update the index metadata structures as new observations are encountered and propagated throughout the index (note that this time also includes lock contention waiting time; index updates and query evaluations are performed using different threads and data structures must be locked in order to not encounter concurrency-related errors). In these experiments the query evaluation period (time between query evaluations) is one second, and the index maintenance time is well below this threshold. Results are plotted in Figure 34 and corresponding values illustrated in Table 6 and Table 7. This evaluation was executed for 159 seconds. When two bags are compared, the number of data points used in the comparison is equal to the cardinality of the smaller bag (i.e.  $k = k'$ ).

**Table 5. System parameters for *Walkers* evaluations**

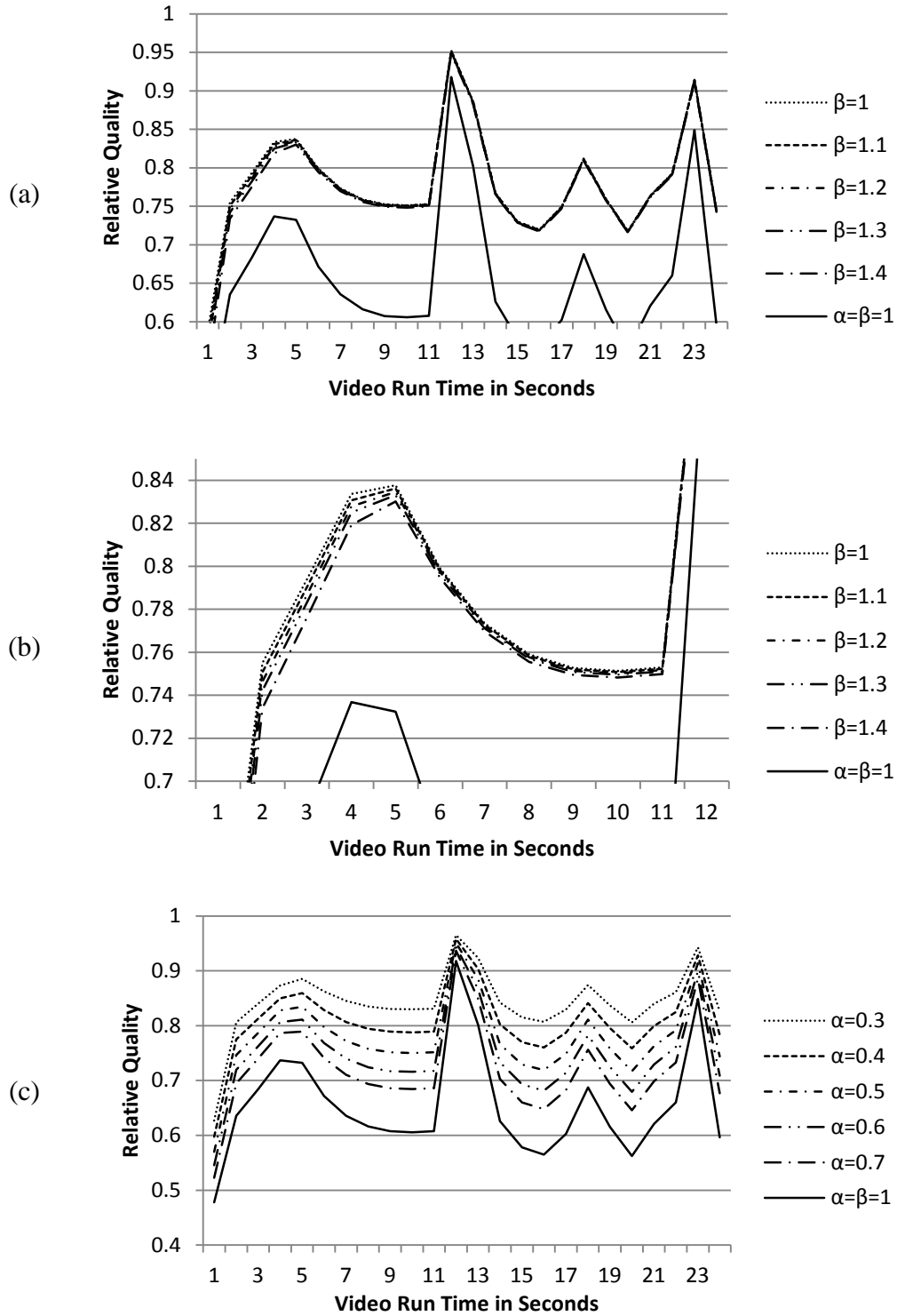
Parameter	Description	Value
Number of histogram bins	Feature vector dimensionality	18
Dynamic object queue length	Number of image frames retained in FIFO database. I.e. maximum length of q_img_bitmap.	200
Cluster manager bag capacity	Maximum number of instances a bag can contain.	25
Inclusion distance threshold	Two bags are considered for merging if the sum of their standard deviations for all dimensions, multiplied by this value, is less than what is returned by the bag distance function.	1.5
Minimum bag comparison size	The minimum cardinality of a bag for it to be considered for object matching. Bags with fewer instances are ignored.	10

**Table 6. Walkers–closest point distance**

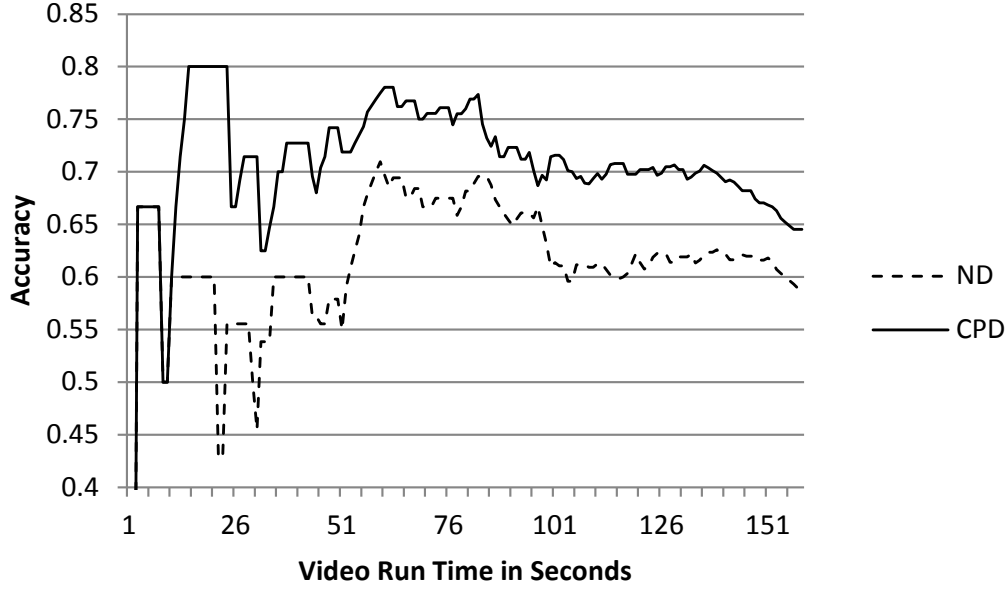
Time stamp	Index maint.	Num bags	TP	FP	TN	FN	Running precision	Running recall
1	86.0049	4	0	0	0	1	0.0000	0.0000
11	9.0005	6	3	1	0	1	0.7500	0.7500
21	1.0001	11	8	1	0	1	0.8889	0.8889
31	2.0001	21	10	3	0	1	0.7692	0.9090
41	1.0001	26	16	5	0	1	0.7619	0.9411
51	26.0015	32	23	8	0	1	0.7419	0.9583
61	55.0032	32	32	8	0	1	0.8000	0.9696
71	34.0019	40	34	10	0	1	0.7727	0.9714
81	134.0077	44	40	11	0	1	0.7843	0.9756
91	1.0000	61	47	17	0	1	0.7343	0.9791
101	496.0284	77	73	28	0	1	0.7227	0.9864
111	283.0162	97	88	37	0	1	0.7040	0.9887

**Table 7. Walkers–normalized distance**

Time stamp	Index maint.	Num bags	TP	FP	TN	FN	Running precision	Running recall
1	62.0035	4	0	0	0	1	0	0
11	8.0005	6	3	1	0	1	0.7500	0.7500
21	2.0001	16	3	1	0	1	0.7500	0.7500
31	5.0003	24	5	5	0	1	0.5000	0.8333
41	3.0001	25	9	5	0	1	0.6428	0.9000
51	5.0002	32	11	8	0	1	0.5789	0.9166
61	32.0019	32	23	9	0	1	0.7185	0.9583
71	1.0001	39	26	12	0	1	0.6842	0.9629
81	6.0003	44	28	13	0	1	0.6829	0.9655
91	27.0015	60	37	19	0	1	0.6607	0.9736
101	10.0006	71	48	27	0	1	0.6400	0.9795
111	7.0004	79	68	42	0	1	0.6181	0.9855



**Figure 33. Relative Quality for *OneShopOneWait2cor*; (a) varying  $\beta$  with fixed  $\alpha=0.5$ , (b) an enlarged view of a portion of (a), and (c) varying  $\alpha$  with fixed  $\beta=1.2$ .**



**Figure 34. Normalized distance (ND) and closest point distance (CPD) plotted over time.**

The third set of results comes from the videos illustrated in Figure 32 (*hallway sequence*). In this scenario a camera is positioned within a room and a second camera outside; objects appear in either one video stream or the second; the cameras do not have overlapping fields of views. Additionally the illumination of the scene is slightly different with the lighting inside the room brighter than in the hallway (as can be observed from the representative images in the figure). Objects in this video sequence appear large compared to the relative size of the frames (field of view) and thus representative feature vectors can be calculated. Evaluation results in terms of the relative quality metric (the best-performing metric that was evaluated) are presented in Figure 35 and correspondingly, Table 8.

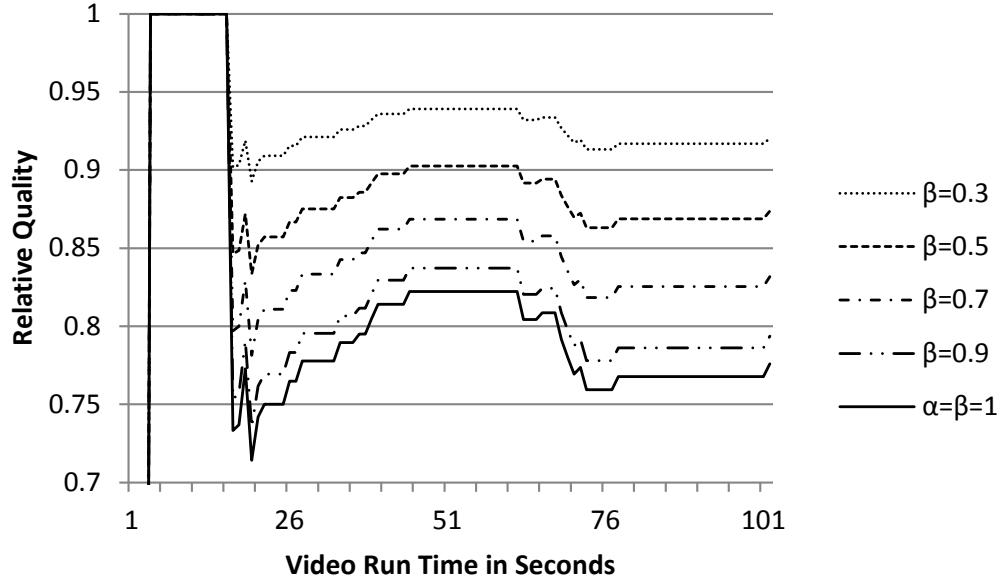


Figure 35. Tracking results for *hallway sequence* videos; where  $\alpha=1$  and various  $\beta$ .

Table 8. Normalized distance for *hallway sequence* videos

Time stamp	Index maint	Num clusters	TP	FP	TN	FN	Running precision	Running recall
1	150.0086	2	0	0	0	0	0	0
11	9.0005	8	8	0	0	0	1.0000	1.0000
21	13.0008	10	23	0	0	8	1.0000	0.7419
31	0	16	28	0	0	8	1.0000	0.7777
41	0	22	35	0	0	8	1.0000	0.8139
51	0	24	37	0	0	8	1.0000	0.8222
61	0	24	37	0	0	8	1.0000	0.8222
71	6.0003	24	40	0	0	12	1.0000	0.7692
81	0	28	43	0	0	13	1.0000	0.7678
91	0	28	43	0	0	13	1.0000	0.7678
101	0	30	43	0	0	13	1.0000	0.7678

#### *Inclusion Distance Threshold*

By leveraging bags and distance functions, the process of matching an object from one video stream to a second is transformed to a clustering and retrieval problem in a multidimensional feature space. The matching between two bags is essentially a *1-nearest neighbor (INN)* query where the bag in the first video is the query point (or, without loss of

generality the second video). Even in a sparse space with distant clusters, a 1NN query would return a match, even though the objects are not particularly similar. In order to prevent against these types of FP a distance threshold is applied, called the *InclusionDistanceThreshold (IDT)*, which effectively creates a bounding hyper-sphere around the query point. Expanding the IDT can result in more TP at the potential cost of additional FPs.

### *Conclusions and Comments*

The quality of cross-camera tracking is dependent upon a number of factors, beginning with the effectiveness of the background subtraction technique that identifies the pixels in the video frames that correspond with salient objects, and the frame-to-frame tracker to continuously track an object and extract corresponding observations and thus feature vectors. Also important is the number of pixels in a scene that provide information pertaining to the appearance of an object; more pixels result in more representative feature vectors being calculated. One of the downsides of the CAVIAR video is the relatively low resolution of 384 pixels by 288 pixels. Thus, only a few pixels contribute to the representation of objects depicted in scenes, unless they are particularly close to the camera.

Table 8 provides a column showing the index maintenance overhead incurred, in terms of wall-clock time. Much of this overhead is due to memory allocation; the LVDBMS is implemented in C# and the .NET runtime manages memory and garbage collection. Some small programming optimizations are made (such as setting variables to null as quickly as possible in the code) but essentially the memory performance and management is left to the runtime.

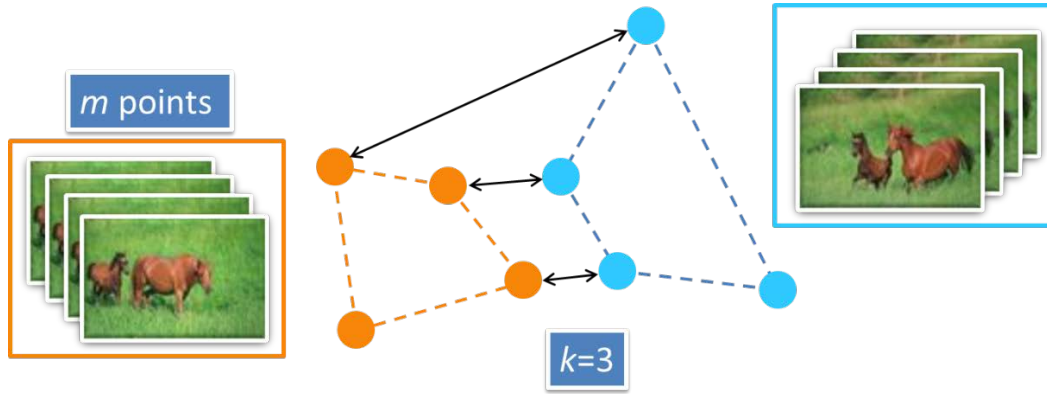
The best performance observed is based upon the distance function that minimizing the distances between the closest points between two point sets. Additionally, not all instances in a bag need to be compared; as an object moves about a scene, a subset of the observations from the



first camera may be more similar to observations by the second camera (e.g. if the subject first walks towards the camera and then turns around and walks a different direction, at least two sides of the subject will be captured). The best matching performance is thus obtained when the number of points,  $k$ , compared is less than the cardinality of the two respective bags; e.g., in some of the experiments presented the maximum bag size (cardinality) is 50 and  $k=10$ . Figure 36 provides an illustration of this concept, showing two image sequences with bag capacities fixed to four and  $k=3$ .

### Summary

Cross-camera matching capabilities are a useful component of a query language. This chapter presented cross-camera matching that is based upon distance functions applied to bipartite graphs. The scenario presented does not require prior calibration or overlapping fields of view, decreasing the overhead required to set up a camera network for surveillance, and facilitating possible future extensions of the network to include mobile devices. By leveraging accumulated appearances of observations of objects and matching upon the most similar subsets of these sequences, the cross-camera matching is robust to momentary observations that reflect periods of dissimilarity between the objects and is on par with results reported in computer vision literature pertaining to distributed computer vision camera tracking algorithms.



**Figure 36.** Illustration of instances corresponding to two image sequences with bags of cardinality 4 and  $k=3$ .

## **CHAPTER 5: MANAGING LIVE VIDEO DATA WITH PRIVACY PROTECTION**

There is an extensive body of work pertaining to video surveillance that focuses on image processing, however, there are only a few that focus on system usability, and specifically privacy. The work presented in this chapter discusses privacy extensions to the LVDBMS that enable the development of privacy-aware surveillance and monitoring applications. These extensions are privacy filters, which can be applied at a granular level to redact the appearance of objects in video streams to implement privacy policies. This framework facilitates the dissemination of privacy-aware video streams in real time. The goal of this framework is to facilitate privacy policies that are verifiable and is an important step towards the future certification of surveillance software in terms of privacy awareness and adherence to privacy standards.

### Introduction

Networks of connected cameras are expanding and widely researched; decreased hardware prices and the expansion of communication networks contribute to their popularity, in addition for increased safety and security concerns. However, research of their effectiveness has mixed results. This can be attributed in part to the fact that such a critical component to their effectiveness are the human operators who monitor the videos for occasional events, who can become fatigued, distracted, biased, etc. Therefore, much of the utility of such deployments is relegated to post-incident analysis.

To mitigate the human element of operator fatigue, and to increase the effectiveness of large camera networks, concepts from computer vision may be applied to increase the usability

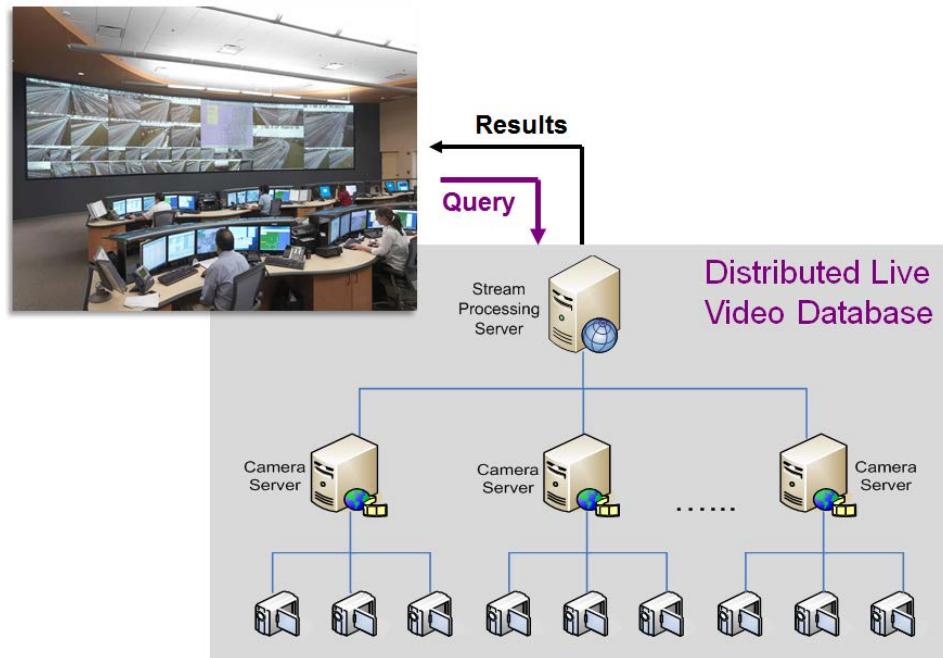
of the raw video streams. The LVDBMS is an example of a software system that is designed to automate the monitoring of live video streams, permitting events of interest to be specified and to notify operators when the events are observed. However, as cameras become deployed pervasively and as the intelligence and capabilities of monitoring software increases, privacy concerns are becoming paramount. Increased monitoring by government agencies and corporate employers raises concerns that the captured imagery will be used consistently with the purposes for which it was initially captures. For example, video is archived and recorded, but future regulation and law changes could permit the captured video to be used for unintended purposes and compromise the privacy of the people observed.

Furthermore, deploying a large camera network requires significant investments in both time and money. Thus it is desirable that camera infrastructure could be used for multiple purposes to maximize the return on investment and help justify deployment costs. For example, a camera network could be shared by police who want to monitor for crimes and collect evidence for investigations, and also by business managers and employees to ensure that customers have a good experience, minimal wait times; that facilities are used efficiently, etc. Thus, it is desirable that these resources are sharable among disparate entities that may be governed by different procedures and regulations. However, this shared usage makes the intended usage of captured imagery ambiguous, leading to uncertainty and privacy concerns. In order to alleviate these concerns and promote system usability, three things a general purpose software system for video monitoring and processing can adhere to are as follows:

1. To facilitate usability, ad hoc queries must be supported. Events of interest across domains (or in particular, even a single user) differ; for example, an event of interest to a fire department could be different than event of interest to a police department.

2. To cope with changing business needs and regulations, it is desirable to have the capability to rapidly develop and deploy customized applications with different purposes, for example, to generate usage statistics by monitoring traffic flow along a road or provide a monitoring service to let business clients know when a conference room is available.
3. As people are concerned with privacy, it is desirable that such applications implement and adhere to standards to protect the privacy of the individuals observed.

To satisfy the first two requirements outlined above, the LVDBMS can be leveraged, as it provides a general-purpose platform for video stream applications and the capability to monitor and query a large number of video streams. Users can specify ad hoc queries in terms of spatial and temporal event specifications and be notified when the event is detected.



**Figure 37. Example illustrating the LVDBMS deployed in a traffic management center.**

To tackle the third issue, in this chapter a privacy framework is presented. Part of this framework is a privacy specification language that has been integrated into LVQL. It permits for privacy policies to be specified, and then granularly enforced by the LVDBMS. For example, a video camera can provide real-time monitoring of a scene, and an employee can be given access to observe a redacted video stream that shows objects moving but hides their identifying information. In the case that a pre-defined event is encountered, the system can save an unredacted clip of video for later offline analysis. The goal is to allow for general trends to be recognized and observed while maintaining an appropriate level of privacy for the individuals observed. As another example, consider a traffic camera that monitors a section of highway. *Department of Transportation (DOT)* employees use such cameras to monitor and ensure the efficiency and availability of transportation networks; Figure 37. Often, television news stations are also given access to view and broadcast live video feeds. However, the intended use by the news stations is different than that of the DOT employees, as they generally want to provide to their viewers a general concept of traffic flow. If an accident were to occur on a highway, it would be undesirable to show related identifying information such as the license plate of an automobile involved. To address this situation a privacy filter could be defined that would apply to objects that exceed a certain size (e.g. if the accident were to occur close to the camera).

### Background

As camera networks increase in expansiveness and smarter algorithms are developed to monitor them, more of people's lives will be observed and recorded. Combine this with increasing quantities of storage at decreasing prices, and what is recorded will be able to be stored and saved for longer periods of time, if not indefinitely. The outcome is the potential for

more of our lives to be monitored, recorded, stored and analyzed. Thus, there will be a growing need for the privacy-conscious treatment of video content. Such software needs to be configurable, as socially acceptable norms differ from culture to culture and the software that monitors and processes observed content likewise needs to be adaptable to the local cultural norms (Caloyannides, 2003; Danielson, 2002).

Currently, a number of privacy-aware systems have been developed. Basically these systems detect movement and redact the detected objects. In (Senior et al., 2005) the appearances of objects are redacted and replaced with colored blobs, where the color signifies some event like the object having crossed a virtual trip wire defined in the system. In (Dufaux & Ebrahimi, 2008) an MPEG-4 encoder is developed that encodes the visual appearance of the object to be redacted in the stream itself, and a paired decoder can decode the redacted appearance as appropriate. However, these systems do not provide sufficient functionality to determine if an object should have its appearance removed from the video stream or not.

Another class of privacy preservation is termed *privacy-preserving data publishing (PPDP)*. A number of entities make large quantities of anonymized data available for public consumption. This type of data is intended to allow for the identification of global trends, for example the spread of disease, city capacity planning, and building classifiers for machine learning applications. If such data were to be released in its raw form, for example patient medical records, privacy laws could be violated and people could be embarrassed, etc. Thus, algorithms (and systems) relevant to the field of PPDP seek to preserve privacy, for example, by modifying the data so that it is not identifiable (e.g., translating a person's age from a specific number like 51, to a range like 50-59), by monitoring queries to ensure that privacy isn't violated, and otherwise perturbing the raw data into a form that can be publically disseminated.

Most PPDP algorithms focus on databases containing statistical data that is oriented in tabular form. Privacy may be applied to such data by restricting how the data is queried (e.g. the number of queries submitted by a particular user or the amount of overlap in data queried), by modifying the raw data (e.g. reporting categorical data or averaging data by zip code) and perturbing the output (e.g. modifying the data by applying some type of random error such that the parameters of the error distribution are known, so that the error can be removed at a global scale) (Adam & Worthmann, 1989; Fung, Wang, Chen, & Yu, 2010). These methods preserve privacy by making the data less granular or by inducing error. Thus, there is a tradeoff between precision and privacy, such that the more the reported values differ from the original data, the more securely privacy is preserved (but potentially more skewed aggregated results).

Privacy preservation and perturbation can also be applied to streaming video. In an ideal situation, no identifying information will be leaked (Caloyannides, 2003). Unfortunately, privacy cannot be guaranteed in the presence of auxiliary information (i.e. when information can be retrieved from multiple data sources, such as both the National Institute of Health and local hospital websites) (Dwork, 2008). For example, information can be leaked via ancillary channels such as the time combined with camera location; for example, someone observed entering an office at 8am and leaving every day at 5pm (Saini, Atrey, Mehrotra, Emmanuel, & Kankanhalli, 2010). Developing robustness to these ancillary avenues of violating the preservation of privacy is beyond the scope of the privacy preserving framework presented in this chapter. The objective of this work is to make the appearance of an object appearing in a video stream unrecognizable if it is associated with a privacy filter. We note, however, that arbitrary stationary objects can have their appearances blocked by having a user specify a static object on top of them, and then define a privacy filter that applies to static objects, for example.



The privacy preserving framework presented in this chapter is designed to be flexible such that it can be selectively applied to different objects, based upon current queries active in the system, the class of the object or other available contextual data that can be leveraged. The appearances of the objects are redacted by drawing on top of them filled bounding boxes; other privacy preserving techniques that are applicable to video are not encountered (such as creating “fake” objects, speeding up or slowing the video temporally).

### Privacy Filter Framework Objectives

Nowadays people tend to accept that they are being recorded while they are in public spaces, for example by security monitoring systems and observed by security personnel to identify potentially harmful or dangerous situations. However, people would feel violated if they were observed and recorded for one particular purpose only to find out that later their information was used for a different purpose. For example had the individual known the later usage of their information, they may have chosen to not visit a particular place or use an alternate mode of transportation, etc.

As the capability to process and store raw video is increasing, the potential to correlate people and actions observed in videos with other data sources in order to gain more in-depth information is increasing. As a progressive move towards a solution to this predicament, we introduce privacy filters. Privacy filters can be applied to specific objects observed in a video, or to all objects. Their application can be hierarchical and tied to a user’s level of access. For example, the identity of shoppers in a mall can be redacted such that security guards can observe people going into and out of stores or traffic flow, but a user with a higher level of access could view the unredacted video stream for quality assurance purposes or to save as evidence for a later

investigation. Thus, the privacy filters presented in this chapter are intended to redact the appearances of individual objects, while maximizing the usefulness and utility of the video stream as a whole. Note that privacy filters apply only to the redacted appearance of an object in the output video stream; privacy filters do not affect the applicability of an object to satisfying the condition of a query.

### *Scope and Assumptions of Privacy Preservation and the LVDBMS*

This section identifies the assumptions and intended scope of applicability of privacy filters as they pertain to LVC and specifically the LVDBMS implementation. The privacy framework revolves around privacy filters. A privacy filter facilitates a particular privacy policy; a privacy policy specifies the circumstances under which the identity (and correspondingly, appearance) of an object, must be redacted from being outputted by the privacy-preserving framework. The intended goal is that the appearance of objects “passing through” a privacy filter will be modified such that their appearance is no longer identifiable based upon the color values of the pixels that contribute to the appearance of the object, as observed by the imaging device (i.e. camera). Thus, a privacy policy defines the circumstances under which an object’s appearance will be redacted. This criteria can be granular (i.e. applicable to a specific object), broad (applicable to all objects or the entire video stream) or somewhere in-between (by associating a privacy filter with a query which itself is defined in terms of spatial and temporal criteria). Therefore, the primary scope of privacy filters are salient objects observed in video streams, not the scene background per se or other information that may be leaked, such as the location of the camera, location of obscured objects, the time of data (e.g. which can be approximated based upon knowing the location of the camera and observing shadows). Also note that privacy filters are applied to the output video stream when it is externalized from the

LVDBMS system; it does not apply to internal metadata structures (which are not made available to users or communicated outside of the LVDBMS framework).

Additionally, the physical security of cameras or processing nodes, or the security of the transmittal of video information between cameras and processing nodes, or communication between processing nodes, is not considered as a part of the privacy implementation in this framework. We do not consider the security of the host operating systems that host LVDBMS software, etc. Security measures such as encryption and the physical security of assets can be ensured either through processes external to the LVDBMS, or in future versions of the LVDBMS. Additionally, privacy attacks directed at the system such as specially crafted queries designed to leak privacy information, or users who masquerade as other users, are not considered as well. Although certain security measures are implemented, such as privacy-preserving views and a requirement of users to supply a valid username and password combination in order to connect to the LVDBMS and view video streams, we do not specifically provide safeguards to protect against the circumvention of these safeguards by a malicious user or groups of malicious users.

### *Overview of Privacy Framework*

Privacy filters are designed to be maximally flexible in order to provide for a solid and flexible framework that is capable of implementing a multitude different privacy policies. Privacy filters can be applied hierarchically at different levels with the LVDBMS tiers. Also, multiple privacy filters can be applied to a particular video stream at the same time, as illustrated in Figure 38.

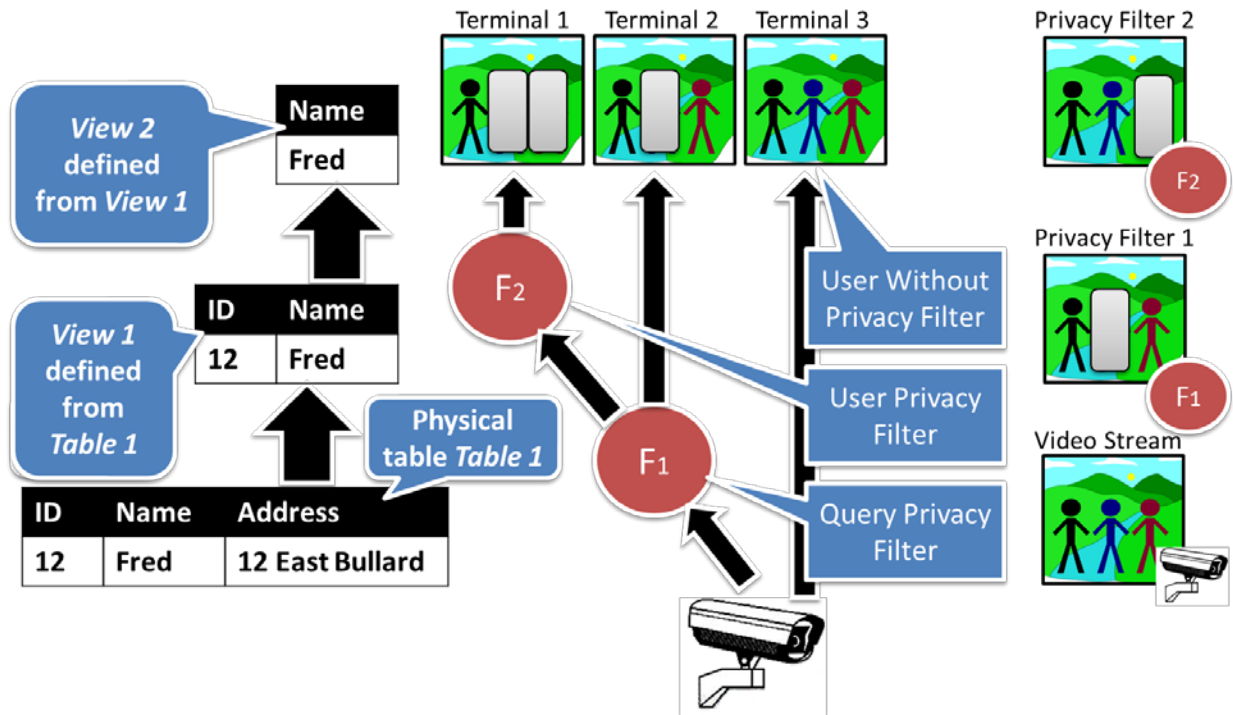


Figure 38. Comparison of cascading relational database views (left) vs. cascading privacy filters (right).

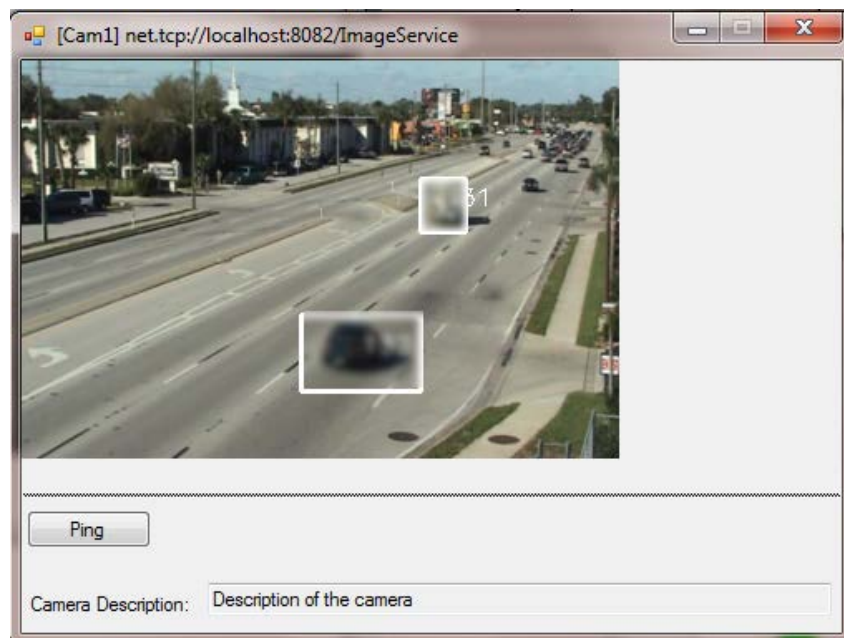


Figure 39. Video stream illustrating a privacy filter with a Gaussian blurred MBR.

Privacy filters can have different levels of granularity, for example, a privacy filter could apply to all objects in a video stream, or to only a particular class of objects. When privacy filters are combined, the most stringent granularity of privacy will be applied as a result of the combined filter. A similar concept exists in relational databases; a user may be granted access to a particular view of the data, and that view could be defined atop other views (i.e. Figure 38 left). Privacy filters can be associated with cameras, queries, user groups and views. A privacy filter associated with a camera will apply to all consumers of the video stream produced by the camera. A privacy filter at this level will have a broad impact as it will propagate to all consumers of the respective video stream. At the query level, privacy filters apply to all objects that contribute to the query condition being evaluated to true. As such, it effects only the consumers of the output of the query. Privacy filters applied at this level have a moderate impact. Privacy filters applied to user groups impact only users associated with the group. Privacy filters defined at this level have a small scope of impact. Privacy filters also may be applied to views. A view is defined as an alias to a video stream but provided the added capability of being able to be associated with privacy filters. Users can subsequently be given access to observe a stream via a view and thus implicitly associated with any privacy filters that have been defined with the view. (Note that such a privacy filter applies only to the stream that is associated with the view; if a user combines a view with a different stream in a query, the privacy filter will not be applied to that second stream.) Privacy filters at this level have a moderate impact, as they are applied to all consumers of the view.

Figure 39 provides a representative example of how a video stream with objects associated with privacy filters might look when rendered to a user's GUI. In this case the privacy filter is rendered with as a Gaussian blur operator (Szeliski, 2010). Applying a blur operator to

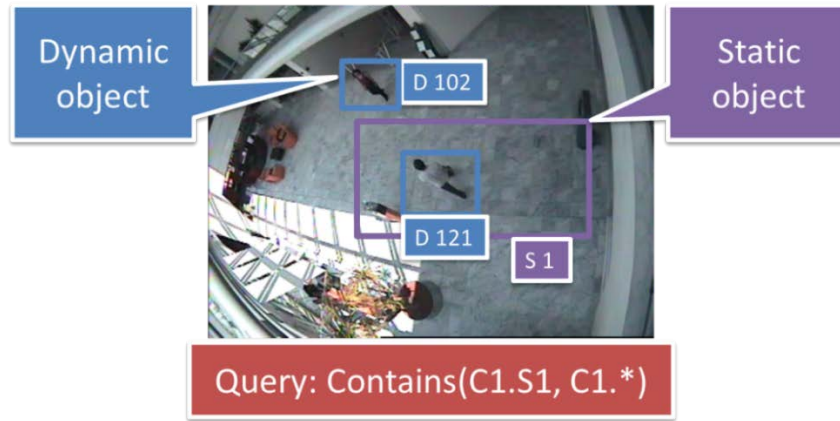
objects obscures their appearance and is not greatly detracting from the quality of the video. Other options are to use the average pixel value inside the MBR, or to simply set the MBR contents to a solid color such as black.

### *Defining Privacy Filters with the Privacy Specification Language*

A privacy filter is the instantiation of a privacy policy; it is defined by the 3-tuple  $\{target, temporal\_scope, object\_scope\}$  where *target*, *temporal\_scope* and *object\_scope* are defined in Table 9, Table 10 and Table 11. If an LVDBMS entity (e.g. a user, video stream or view) that is associated with a privacy filter interacts with a second LVDBMS entity, the privacy filter will apply to the output of their interaction. For example, if a user is associated with a privacy filter and then views a video stream, the privacy filter associated with that user will apply to the video stream when it is being viewed by that user. (However, if a second user views the same stream and is not associated with any privacy filters, the privacy filter from the first user will not apply to the second user or the stream from the second user's perspective.) Similarly, if a user is associated with a privacy filter and that user creates a query, the query will also be associated with that privacy filter.

If an object that is observed in a video stream is associated with a privacy filter, the object's appearance may be redacted when the stream is output from the LVDBMS. That is, a privacy filter that is associated with a stream applies to objects observed in that stream in accordance with the 3-tuple of values it is defined by. The *target* attribute of a privacy filter pertains to objects that are the target of queries (an object is a target of a query if it contributes to the query being evaluated to true). The possible values for this attribute are given in Table 9. As an example, if a privacy filter has a *target* attribute value of *query targets*, then the privacy filter will apply to all objects observed in the stream which contribute to a query (defined over the

stream) condition being evaluated to true. Thus, given a privacy filter defined as  $\{Query\ targets, None, None\}$ , and a query condition  $Appear(c1.*, 250)$ , then all objects observed in the stream greater than 250 pixels in area will be associated with the privacy filter. Objects having area less than 250 pixels will not be associated with the filter. Figure 40 provides an illustrative example with the *Contains()* operator; any dynamic objects appearing within rectangle *s1* satisfy the query condition. Possible values of the *temporal scope* attribute are given in Table 10. The temporal scope pertains the period a privacy filter is active in the time domain; a value of *permanent* means it is always active. The other potential values allow the life of a privacy filter to be correlated with the presence (or absence) of a query. The *object scope* (Table 11) specifies classes of objects a privacy filter is associated with. For example, a privacy filter can be defined to be associated with all dynamic objects, or all static objects in a video stream. If a privacy filter has an attribute with the value *none*, that attribute will not be considered when determining which objects in a video stream the privacy filter will be applied to.



**Figure 40. Query targets: object *D121* satisfies the query condition, *D102* does not.**

**Table 9. Privacy filter attribute *target***

Value	Description	Priority
None	No privacy; attribute ignored	1
Query targets	Targets of active queries are obscured	2
Non-query targets	Objects that are not targets of active queries are masked. An active query may obscure their identity	2
Previously masked	Specifies that objects that were previously masked will continue to be masked	2
All	All object identities are masked, regardless of query status	3

**Table 10. Privacy filter attribute *temporal scope***

Value	Description	Priority
None	No privacy; attribute ignored	1
Query non-active	Privacy settings apply only when a query is not active	2
Query active	Privacy settings apply only when a privacy-enabled query is active (in the case of privacy applied to a camera, for example)	2
Permanent	Privacy settings apply for the lifetime of the object or camera or query	3

**Table 11. Privacy filter attribute *object scope***

Value	Description	Priority
None	No privacy; attribute ignored	1
Cross-camera dynamic	Objects that are first detected in another camera	2
Dynamic	Dynamic (automatically detected) objects	2
All	All classes of objects qualify	3



Privacy filters are specified with the PSL extension of LVQL. The PSL allows privacy filters to be defined and associated with cameras, queries, user groups and views. Note that any objects of these types can be associated with zero or more privacy filters at any particular time. The association of additional privacy filters to an object increases the level of privacy associated with that object; that is, privacy filters behave only in an additive fashion and one cannot add an additional privacy filter to an object to reduce its level of privacy. If two users are accessing a video stream and one user is associated with a privacy filter (e.g. via a group membership), the privacy filter will not apply to the second user. Thus, the first user might view a video stream and the identities of the objects will be hidden only to the first user and not to the second. If the first user issues any queries, their privacy filters will also be associated with the queries.

PSL allows one to create, modify and delete privacy filters and associate or disassociate them with various objects that reside in the LVDBMS. The syntax of specifying a privacy filter is given in Figure 41. By leveraging the same interface to specify privacy filters as LVQL uses to specify queries, privacy filters can be scripted alongside queries. From an implementation perspective, the same facilities that parse and process LVQL commands are leveraged to implement the PSL extension.

```
{CREATE | UPDATE | DELETE} FILTER filter_identifier
    [TARGET = {QUERYTARGETS | NONQUERYTARGETS | PREVIOUSLYMASKED}]
    [TEMPORALSCOPE = {QUERYNONACTIVE | QUERYACTIVE | PERMANENT}]
    [OBJECTSCOPE = {STATIC | DYNAMIC | CROSSCAMERADYNAMIC}]
{CREATE | UPDATE | DELETE} VIEW view_identifier OVER stream_identifier
    [WITH filter_identifier]
{ASSOCIATE | DISASSOCIATE} GROUP group_identifier WITH
    {FILTER | VIEW} filter_identifier
{CREATE | DELETE} USERGROUP group_identifier
{ASSOCIATE | DISASSOCIATE} USER user_identifier WITH group_identifier
```

**Figure 41. The PSL extension of LVQL; colored text illustrates user-supplied values.**

### *Combinations of Privacy Filters*

Multiple privacy filters can be applied to an object, either explicitly (directly via the PSL) or implicitly (such as when a user who is associated with a privacy filter views a video stream). When multiple privacy filters are applicable to an object they are combined into a single effective privacy filter to determine which objects they will be associated with. Each attribute type of the privacy filter 3-tuple will be compared independently. Each attribute value is associated with a priority value. When privacy filters are combined, if two filters have the same value for an attribute (e.g. they both have the value *query active* for the temporal scope) then the resultant privacy filter will have the same value. If they have values that have different priorities, the value associated with the higher priority will be retained. In the case of differing attribute values but the values have the same priority, then the attribute will assume the value associated with the next-higher priority. For example, if two privacy filters are combined, one has object scope value *cross-camera dynamic* and the other *dynamic*, the resulting attribute will be *all*.

### *Formal Specification of the Privacy Filter Model*

A privacy filter can be described in terms of access and sanitation functions. Given a set of streams  $\mathbb{S}$  and a set of active queries  $\mathbb{Q}$  posed over said streams, we can define a stream  $\mathcal{S} \in \mathbb{S}$  as a sequence of frames,  $\mathcal{S} = \{f_i, f_{i+1}, \dots, f_{i+k-1}\}$ , such that  $|k - i|$  is a sufficient quantity (i.e. sliding window) of frames to resolve any query  $q \in \mathbb{Q}$ , and  $f_i \in \mathcal{S}$  corresponds to the current frame of video in stream  $\mathcal{S}$ .

A frame  $f_k$  can be obtained from  $\mathcal{S}$  by calling an access function:

$$\text{Acc}(\mathcal{S}, k) \rightarrow \text{frame} \quad (12)$$

As a stream that is associated with a privacy filter is transmitted from the LVDBMS, it is sanitized with a sanitizer function:

$$\mathbf{San}(\mathcal{S}, f) \rightarrow \mathbf{Acc}(\mathcal{S}, 1) \oplus \mathbf{Z}(f) \quad (13)$$

The  $\oplus$  operator perturbs the pixel values in the frame in accordance to the bitmap mask  $Z$  which indicates which pixels need to be obscured by privacy filter  $f$ . If a stream is associated with multiple privacy filters (i.e. the stream is associated with a view and is being accessed through the view) then the privacy filters can be combined with the  $*$  operator and the sanitation function becomes:

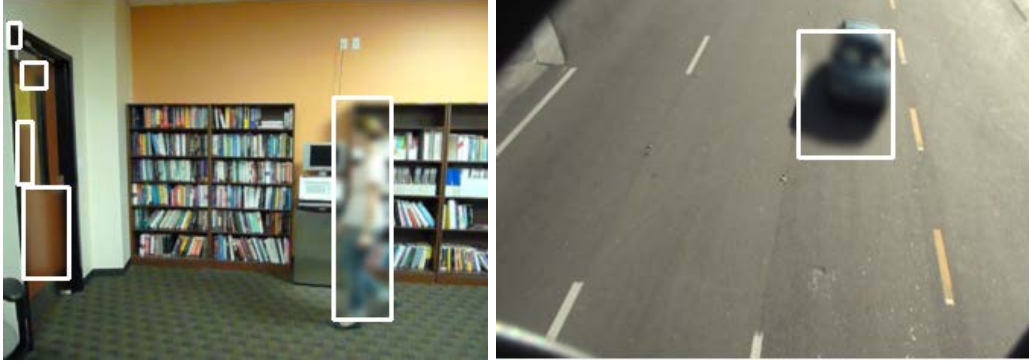
$$\mathbf{San}(\mathcal{S}, f) \rightarrow \mathbf{Acc}(\mathcal{S}, 1) \oplus \mathbf{Z}(f * f') \quad (14)$$

Such that  $f'$  is a second privacy filter.  $Z$  behaves deterministically with respect to its input. Note that some sanitation functions can choose whether or not to respond to a query based upon the query history, or perturb the image with additive noise according to a statistical distribution with known parameters.

### Performance Evaluation

To evaluate privacy filter effectiveness three sets of videos were utilized. The first set of videos was created in an academic building on the University of Central Florida campus; video sequences with two cameras positioned inside rooms and a third camera positioned in a hallway outside of the rooms. This is a challenging object matching scenario because the two rooms have slightly different levels of ambient lighting, and the objects appear to be different sizes due to their proximity to the camera. The hallway has windows along one side providing lighting from outside which causes its illumination to differ from the room scenes. A total of five people are

depicted in this video set, with at most three appearing in a scene at any one time. The second set is from the CAVIAR video library (R. Fisher, 2011). The videos pairs selected for experiments observe the same scene from two different angles; a front view and a side view. This is a challenging video sequence, due to its low resolution a minimal number of pixels contribute to the appearance an objects, making it difficult to distinguish between different the different objects. The third video sequence depicts automobile traffic driving along a road during the daytime. Appearing in this video are automobiles and a few pedestrians (walking or riding a bicycle). In these sequences automobiles drive down lanes in a road and are observed from an overhead view looking downwards. The automobiles are rigid and do not change shape as they move, although they do cast shadows on the road which change as the angle with the sun changes due to vehicle motion (and in some sequences road curvature). In the scenarios presented here, prerecorded videos were utilized to allow the experiments to be repeated with different system parameters. In these experiments, objects having an area of less than 200 pixels are ignored (unless explicitly stated otherwise). Two computers were used to host LVDBMS software; stream and spatial processing layers were hosted on a computer running the Windows 7 Ultimate operating system with 3GHz Pentium IV CPU and 3GB RAM. The camera adapter and client GUI components were hosted on a 2.54 GHz Core-2 Due CPU with 4GB RAM, again running the Windows 7 Ultimate operating system. Both hosts are connected via a gigabit Ethernet switch.



**Figure 42. Privacy filter examples from the first (left) and third (right) video series.**

### *Privacy Filter Effectiveness*

The principal concept of a privacy filter is to redact the appearance of an object it applies to from the output video stream; for example see Figure 42. In the LVDBMS prototype five privacy filter rendering methods are implemented; *average pixel value of bounding box*, *blur*, *solid black*, *outline* and *none*; where outline shows the object's MBR and is used for debugging purposes and none does not redact an object's appearance from the output stream.

The two scenes illustrated in Figure 42 show frames from scenes with privacy filters associated with objects. In this case the query contains an *Appear()* operator. In the case of the frame shown on the left, a person is walking. One can observe four FP regions being redacted due to the door opening and closing, which can be attributed to background modeling errors. As can be observed from the figure, a blur privacy filter rendering method is used in this example. Applying a blur operator removes personal identifying information from the video while allowing an operator to observe the behavior, and the blurred object is not as visually distracting as, say, the solid black rendering method which creates significantly more visual contrast between the object and its background. Other methods for removing the appearance of an object are described in literature, such as increasing the size of the obstructing bounding box (to mask the

size of the object), creating “ghost” boxes (to mask the occurrence of a real object) or simply to substitute the background pixels in the place of the object (also to hide the appearance of the object). However, adding “ghost” objects or hiding the fact that an object is observed in the video stream would decrease the utility of the video from a surveillance perspective and are not further investigated in this work.

### *Object Tracking Effectiveness*

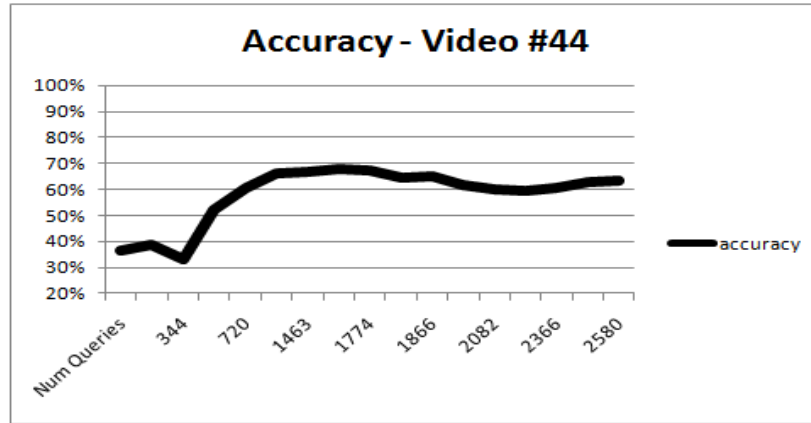
Cross-camera tracking results are presented in this section, which gauge the effectiveness of queries to resolve to the correct result when the specified event is defined over two video streams. As privacy filters may be associated with such queries, privacy filter performance can be determined by the performance of the queries they are associated with.

The first set of results presented are based upon the first video sequence described earlier; the scenario with two cameras in two rooms and a third camera in a hallway. Results are presented in terms of the Accuracy metric, whose equation is given as:

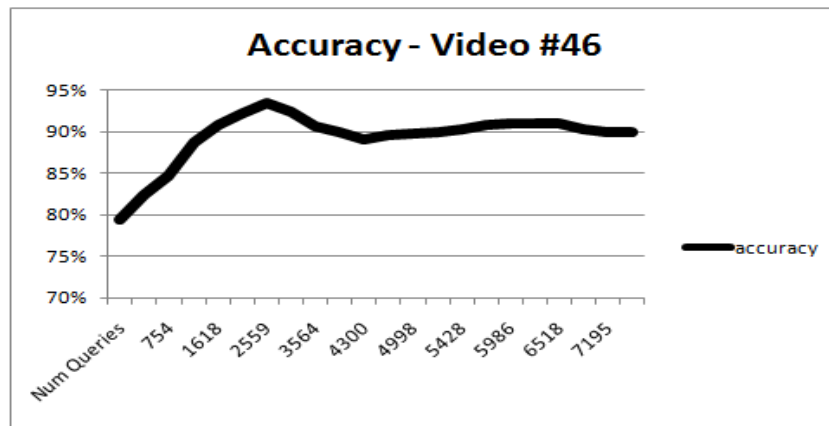
$$Accuracy = \frac{TP}{FP+FN+TP} \quad (15)$$

Two plots of the object recognition performance plots are provided in Figure 43 and Figure 44. What is plotted is the accuracy of an object recognized in one video stream being matched to the correct object in a second video stream and the correct bags subsequently combined. In order to maximize the number of object matching evaluations, the matching logic is executed for each frame of video, and if an object match is recorded, it is disregarded and objects are re-matched in each subsequent video frame (i.e. object matches are not recorded across frames). As can be observed from the plots, initially the bags corresponding to the objects have fewer instances and matches are based upon matching fewer observations. The *normalized distance* function utilized

to gauge the similarity of the two point sets. Therefore, object matching is performed as follows; objects are matched from their appearance in one video to their appearance in a second. An object's bag in one video is utilized as a query point to retrieve its 1NN; if another feature point corresponding to a bag in the second video is within the IDT, it is considered for a match. If the proper matching object from the other video is returned, TP is incremented, else it is an FP. (In the case of an FP either the matching point does not exist in the index and an incorrect object was returned, or the matching object does exist in the index but a different object lying closer to the query point was returned.) If no object lies within the IDT and there is no corresponding object in the index, TN is incremented else FN is incremented (FN meaning that the proper object exists in the index and is farther from the query point than the IDT permits for matching consideration).



**Figure 43. Cross-camera query evaluation accuracy for video sequence #44.**



**Figure 44. Cross-camera query evaluation accuracy for video sequence #46.**



Continuing with the query evaluations, an additional four queries were evaluated over CAVIAR videos, two queries are posed over single video streams and two involve cross-camera tracking. The queries and results are presented in Table 13. The first two queries presented involve only a single camera stream and are not dependent upon the performance of the cross-camera matching. Their accuracy falls back to the performance of properly segmenting an object from the background of the video and then tracking that object within the view of a single camera. A separate section of privacy result evaluations is not presented because that result set would be equal to what is presented in the *accuracy* column here.

In order to arrive at the results presented in Table 13, query accuracy was evaluated manually based upon a human observer determining if the requisite event is depicted in the video stream and recorded at five second intervals over a two minute evaluation period. For example, if the query result, as calculated by the LVDBMS, was correct, then TP would be incremented.

**Table 12. Privacy filters corresponding to scenario in Figure 45**

LVDBMS Objects	Associated privacy filter
Camera	<i>None</i>
TMC operator (user 1)	<i>None</i>
News station (user 2)	<i>None</i>
View 1	Target = <i>query targets</i>

In the scenarios presented thus far, each frame is evaluated independently and privacy filters are applied. If a background segmentation or query evaluation error were to occur then the appearance of an object would not be redacted from the output video stream. Additionally, if two objects appear similarly, then they cannot be distinguished without some other type of identifying information (such as the detection of an RFID-enabled badge). An alternative

scenario would be to redact the entire video and only show detected objects which are not positively associated with privacy filters.

### *Holistic Demonstration of a Privacy Filter*

This subsection provides a demonstration of a privacy filter evaluated over a traffic video from the third video dataset. The active query in the system checks for the appearance of an object via the *Appear()* operator, and is illustrated in Figure 45.

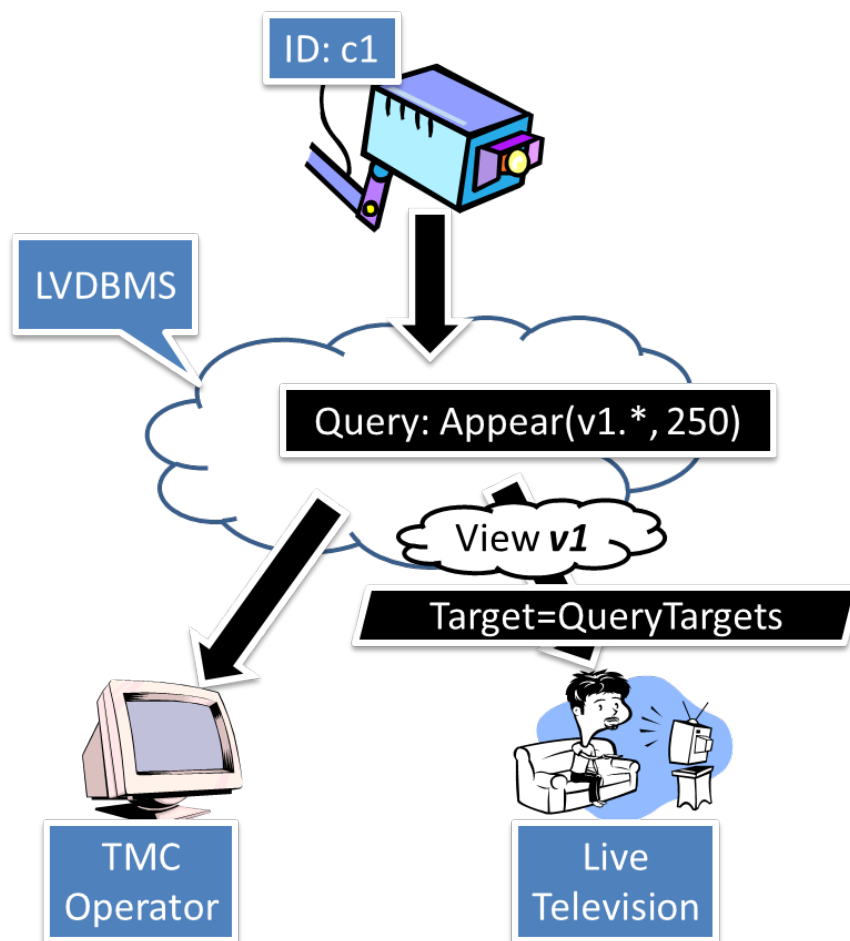
In this example a live video feed is originating from a traffic camera *c1*. A view, *v1*, has been defined over the *c1* video stream in the LVDBMS. Associated objects and corresponding privacy filters are indicated in Table 12.

**Table 13. Query accuracy evaluation results**

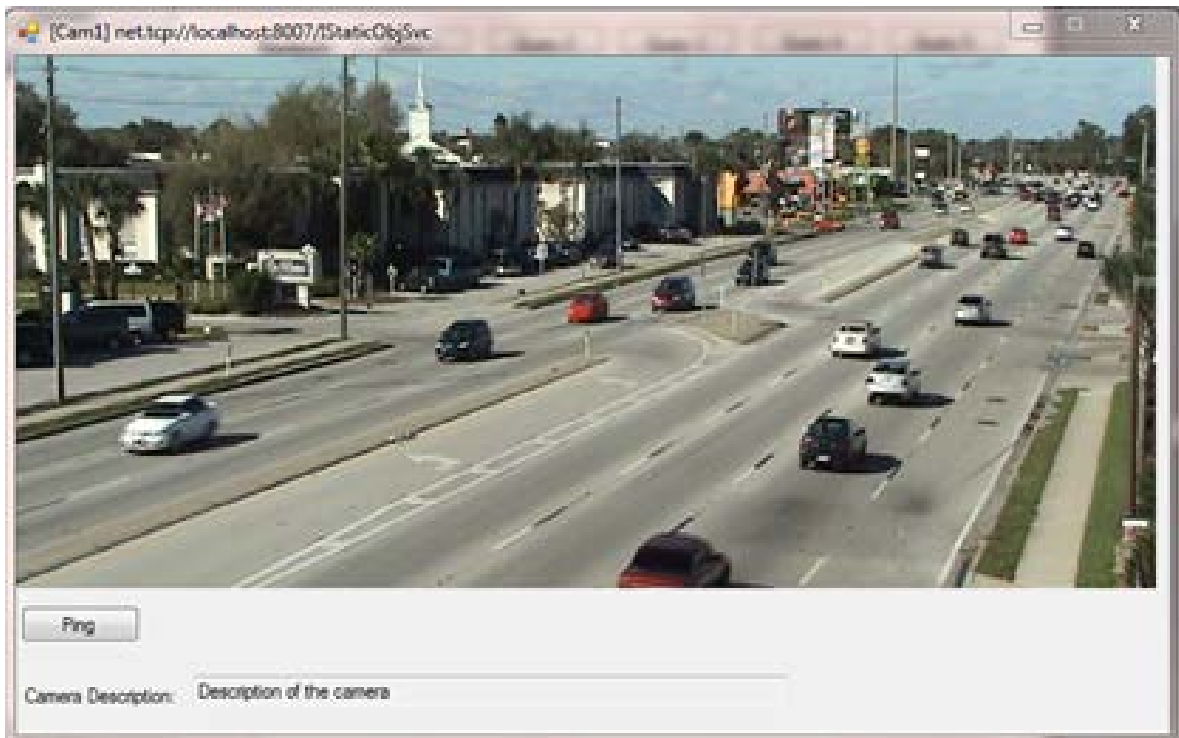
Query Name	Description	Accuracy
Appear	True if objects with area greater than 100 pixels appears in the frame, else false	100%
North before south	True if there exists an object is moving with downward motion. Before operator has a window size of 20 frames; if the object stops or changes direction for less than 20 frames it is still considered true.	100%
Appear across cameras	A person appears in camera 1 and then is recognized when they appear in a second camera.	83% (TP=20, FN=4)
Appear, then cover across cameras	An object appears in camera 1 then goes through a door (outlined by a static object) in the second camera.	91% (TP=22, FN=2)

In this scenario a single camera is generating the video stream input into the LVDBMS. The *traffic management center (TMC)* operator can view the unredacted video stream directly; Figure 46. However, a live (real-time) news feed created for dissemination to a news station passes through a view, *v1*, defined over the video stream *c1*. As this view has a privacy filter

associated with it (i.e. see Table 12), the privacy filter is applied to the output video stream that is disseminated for public consumption (Figure 47).



**Figure 45. Holistic privacy filter example showing a camera, a view with associated privacy filter and two video consumers.**



**Figure 46. Imagery as observed by the TMC operator.**



**Figure 47. Video stream as observed through the view; note the blur effect.**

### Summary

Networks of connected cameras assume an increasingly important role in ensuring safety and are finding applications in many diverse application areas, to include healthcare monitoring, viewing of inaccessible areas such as the inside of chemical tanks and the tops of bridges. In this chapter a privacy preserving framework was presented. The LVQL language allows for privacy-preserving views to be defined, and privacy filters to be specified. The combination of a declarative query language and a general purpose real-time video stream processing data infrastructure provides a framework for rapid privacy-aware video processing application development, by affording application developers and designers the ability to focus on their core business problems, and not on creating stream processing functions that are ancillary to their primary goals. The privacy preserving framework presented in this chapter, and corresponding experimental results, indicate that the privacy preserving techniques presented in this chapter can effectively be applied to an LVC environment and are amenable to real-time processing.

## CHAPTER 6: EFFICIENT QUERY PROCESSING

As the number of cameras deployed and monitored continue to increase, the need for economical processing algorithms will also increase. The LVDBMS is a stream processing database environment for LVC. Users of the LVDBMS can submit queries describing events, and be notified when the queried event is observed. In order for the LVDBMS to be able to handle a maximal number of queries given an implementation consisting of fixed infrastructure resources (i.e. available CPU, primary memory and network capacity), the speed and efficiency with which queries can be evaluated and their results processed is of prime importance.

In a database system, a component called a query optimizer assumes a pivotal function in the construction and execution of queries. The query optimizer receives a query, in an intermediate form, and outputs a selected query evaluation plan that is ready for execution by the host database platform. In this chapter a query optimizer is presented that is designed to generate queries optimized for a dynamic stream processing environment. When generating a query execution plan, this query optimizer considers current queries that are executing in the system and trailers an execution plan that attempts to take advantage of intermediate computation that is already being performed for existing queries with the idea of eliminating redundant computation as a way to conserve system resources. Thus, in this section a query optimizer is presented and evaluated in the LVDBMS test bed environment. Also presented are query execution optimizations designed to benefit from the multicore processors that are prevalent in modern computers.

## Introduction

Environmental monitoring by video cameras is applicable to a number of domains ranging from transportation, security and manufacturing to battle field scenarios. However, with current algorithms, there is a wide performance gap between what we would like to have automated, and what we actually can automate. Thus, human observers continue to play a vital role in the detection of critical events that are observed by video cameras. However, for a human to continuously observe video of a scene and maintain high levels of vigilance in spite of distractions and fatigue is very difficult, especially if they must watch for long periods of time in order to observe an event that happens very infrequently.

Thus, a number of video monitoring solutions have been created to address the problem of event detection, however, most of these are applicable to only a narrow scenario (Velipasalar et al., 2010). The approach of building a vertical video processing application to solve a problem in one particular domain can result in information silos and inefficient use of resources. Furthermore, the deployment of a large scale network of video surveillance hardware is expensive and time consuming. It would be desirable if such infrastructure could be shared for multiple applications, thus increasing its potential return on investment. Similar to how a general purpose relational database may be utilized to develop a variety of business software applications, the LVDBMS is a video stream processing platform designed to satisfy real-time processing and throughput needs that can be leveraged by developers and architects creating stream processing business solutions. The concept of LVC treats networks of video cameras as data input devices and leverages the LVQL query language to facilitate rapid stream processing application development.

The LVDBMS (Aved & Hua, 2012; R. Peng et al., 2010) is a prototype test bed implementation of an LVC environment. It improves upon a multitude of existing video and multimedia database solutions, e.g. (Ahanger & Little, 1996; Ahmedali & Clark, 2006; Antani, Kasturi, & Jain, 2002; Hampapur et al., 2005; Velipasalar et al., 2010). However, many of these existing methods utilize offline processing and video storage that is not feasible in a video stream processing environment due to the velocity and volume of the incoming data streams. As an example, the BilVideo (Catarci et al., 2003) video database solution permits search based upon spatial and temporal features, but the necessary feature extraction phase is conducted offline. Stream data in the LVDBMS must be processed online and with little delay, as processing performance and overhead must be minimized in order to maintain minimal detection delays after an event has occurred.

The LVDBMS is a 4-tier stream processing database environment designed for automated event detection, with applications in real-time surveillance with a large number of video cameras. Users submit events of interest to be monitored in the form of LVQL queries; when the system detects that an event has been observed, an associated action can be executed. For example, sending a user notification or recording a portion of a video stream to storage media.

### Background

In non-video-stream processing environments a significant amount of work has been performed pertaining to query optimization. For example, some previous works (Finkelstein, 1982; Hall, 1974) utilize heuristics to find common sub expressions. Other works utilize a pre-processing step to generate metadata about the queries (Grant & Minker, 1981; W. Kim, 1984).



Another work pertaining to multiple-query optimization is (Sellis, 1988), but it also is oriented towards queries posed over relational databases. In the domain of distributed stream processing and networks of sensors, a multitude of work pertaining to optimizing queries and multiple queries over data streams has been done, for example, (Ahmad & Cetintemel, 2004; Babcock, Babu, Datar, Motwani, & Widom, 2002; Babu & Widom, 2004; Makhoul et al., 2000; Pietzuch et al., 2006).

LVC databases differ from relational databases and sensor stream processing applications. For example, in a surveillance deployment, a “hot stream”, i.e. a video stream that is utilized by a multitude of queries due to a strategically positioned camera, may become a focal point of system resource contention. In traditional DBMSs, hotspots are addressed by efficient and granular data locking techniques, or by caching popular data. Such techniques are not applicable to an LVC environment (e.g. caching would not be feasible due to the high velocity and variety and rapid processing requirements that pertain to data extracted from video streams).

### Query Processing

This section introduces query processing in the LVC environment.

#### *Overview*

In this work LVQL is discussed; the high-level declarative query language of the LVDBMS. It is compared with *Structured Query Language (SQL)*, which is a widely-known and popular declarative query language. Declarative query languages express the desired query results that should be returned; they do not specify the processing control flow or algorithms that should be implemented in order to obtain the desired results; the host database system must find an efficient plan to process the query. Note that query optimization hints are beyond the scope of

this work; e.g. (Bruno, Chaudhuri, & Ramamurthy, 2009), which provide a way for the query writer to influence the query optimizer in cases where the query optimizer might make a poor choice pertaining to the execution plan it selects. (One of the downsides of query hints is that when queries are imbedded in applications and have hints, the hints may be pertinent for a particular distribution of the data or a specific version of the host database environment, and over time may lead to worse performance.)

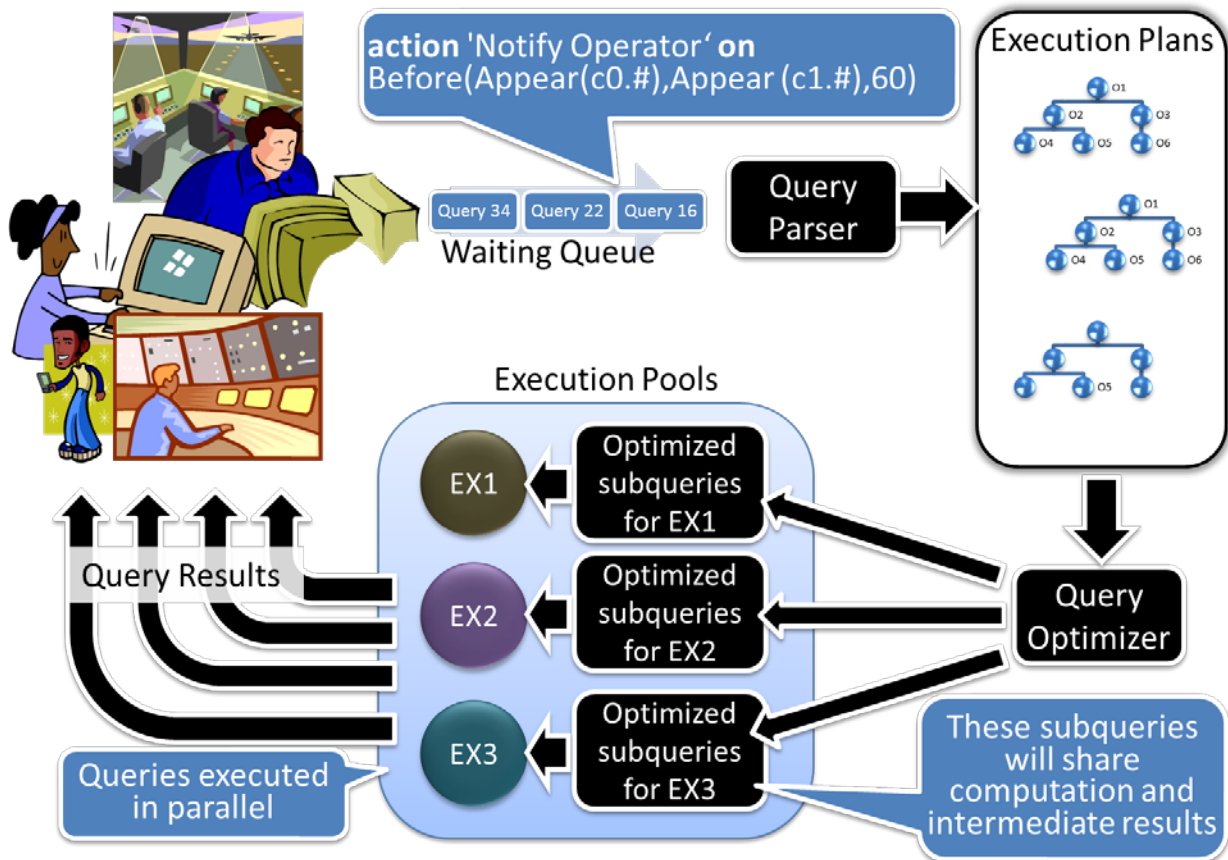
### *The Query Parser and Translator*

Before a query can be sent to the query optimizer, its syntax must be verified and parsed. The LVDBMS query parser ensures that received LVQL queries are syntactically correct, and then it generates an algebraic tree version of the query. This algebraic tree is sent to the query optimization module, which generates an efficient execution plan and metadata that corresponds to the query that is used by the execution engine to schedule the execution of the query. Next, the query is scheduled for execution, and results are returned to the user. Queries continue to execute until terminated or an error occurs (for example, if a video source goes offline or ceases to exist in the system). An overview of the query lifecycle is give in Figure 48.

LVQL is defined by a set of productions that specify the strings (i.e. queries) that are valid under the language definition, and may be referred to as the logical algebra, which defines the expressions that are valid in the language and is tied to the data model. When the query parser receives a query, it attempts to parse it against this grammar, which is given in *Extended Backus-Naur form (EBNF)*. The first step the parser performs is lexical analysis; parsing the query into discrete tokens by a scanner looking for pre-defined sequences of characters that comprise tokens in the language. The second step is syntactic analysis which entails parsing the query. The parsing component checks that the input query string matches a valid query

expression in accordance with the LVQL grammar. The output of the query parser is an expression tree; an intermediate format of the query. This is then passed off to the query optimizer which generates the actual plan that will be executed by the database management system to produce the query result; this execution plan is expressed in what is called the physical algebra, which is platform-dependent. The physical algebra consists of the specific steps that will be undertaken to derive the query result (i.e. data flows) and also the algorithms and associated data structures that are implemented at each step. As such, the physical algebra is specific to a particular architecture and platform, and even a particular version of the DBMS. The execution plan expressed in the physical algebra is also commonly referred to as the query plan, or the query execution plan. Figure 49 provides a graphical illustration of the various stages of preparing a query for execution along with the LVDBMS components that transform the query from one form to the next. Note that the LVDBMS utilizes the Coco/R compiler generator (Aho & Ullman, 1972).

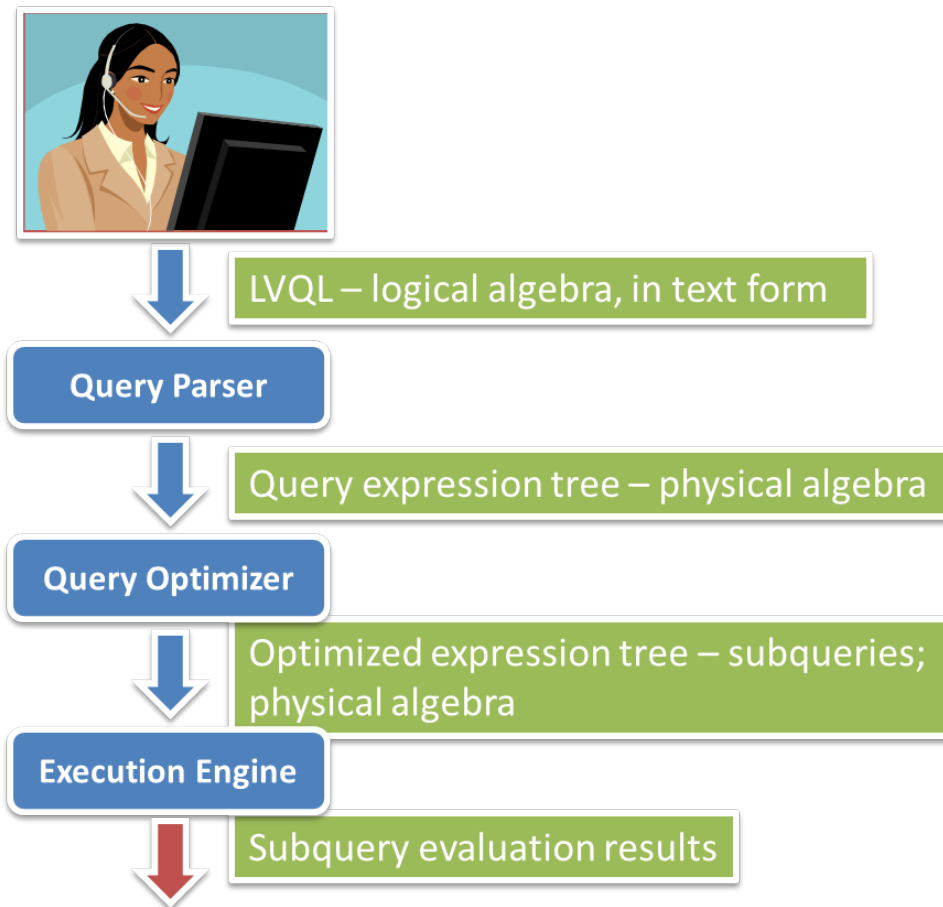
In traditional database management systems the query optimizer first generates a series of plans which all produce the same final result, but vary in the specific algorithms and processing steps undertaken to achieve that result. Query plan generation and optimization is a difficult and computationally intensive procedure, and typical query optimizers use heuristics to guide them in the process. A popular heuristic pertains to the utilization of an index (e.g. a hash index) to perform record retrieval based upon an identifying data field; if it is estimated that less than 10% of the records will qualify for a retrieval based upon data statistics the index will be utilized (and thus random disk accesses), else, the entire table will be scanned serially for qualifying values.



**Figure 48. Query lifecycle; from the inception of a query to results delivered to the issuer.**

### *The Query Optimizer*

The high-level language LVQL does not specify the logic or specific algorithms and data structures that will be required to derive and return a query result; it is the job of the query optimizer to determine these and then hand this off for execution. The input to the query optimizer is an intermediate tree structure that represents the query, and is the output of the query parser module. The output of the query optimizer is one or more subqueries. Each subquery corresponds to a spatial layer host where it will be executed. The subquery placement is dictated by the video streams associated with the respective spatial processing layer host.



**Figure 49. Query transformation steps, from inception to execution.**

Internally to the query optimizer it is generally the case that a series of equivalent execution plans are generated. They are equivalent in the sense that they return the same final query result, but differ in their internal data flows, data reductions and algorithms implemented at the various steps (Chaudhuri, 1998). Cost estimates are applied to these, and a “good” one is selected. The various execution plans can have vastly different runtime characteristics and the selection of execution plans can have an overarching effect on the overall performance of the DBMS. Query plan selection can be done by consulting the statics in the metadata that describes the available data to derive estimates pertaining to both the cost of executing a particular operation, and a summary of the resultant data that is output by that operation.

The query optimizer for the LVDBMS is designed to work a bit differently than traditional query optimizers for relational DBMSs. The statistical summary data that a query optimizer uses to select a query plan either isn't available for a stream-oriented platform, isn't applicable, or would change so frequently that it would be cost-prohibitive to keep current. The LVDBMS query optimizer optimizes queries at runtime and only considers "static" metadata that pertains to video streams, such as which spatial layer host they are associated with. Given a spatial layer host that a query will need to be executed on, the LVDBMS query optimizer takes into account the queries that are currently executing on that host in order to derive a good execution plan for the new subquery.

An additional criteria that can be considered by traditional query optimizers is the type of query that it received; for example in the case of a batch query, a user will not be receiving the results immediately and an execution plan that maximizes the number of batch queries that can run concurrently with reasonable performance is a potential optimization criteria. In the case of an interactive query, where a user is waiting for the query result, the time to returning an initial query result (or portion thereof) may be the optimizing factor. In the case of the LVC we have two primary criteria that queries are optimized for:

- Real-time query results, and
- Scalability; the number of queries that may be concurrently executed.

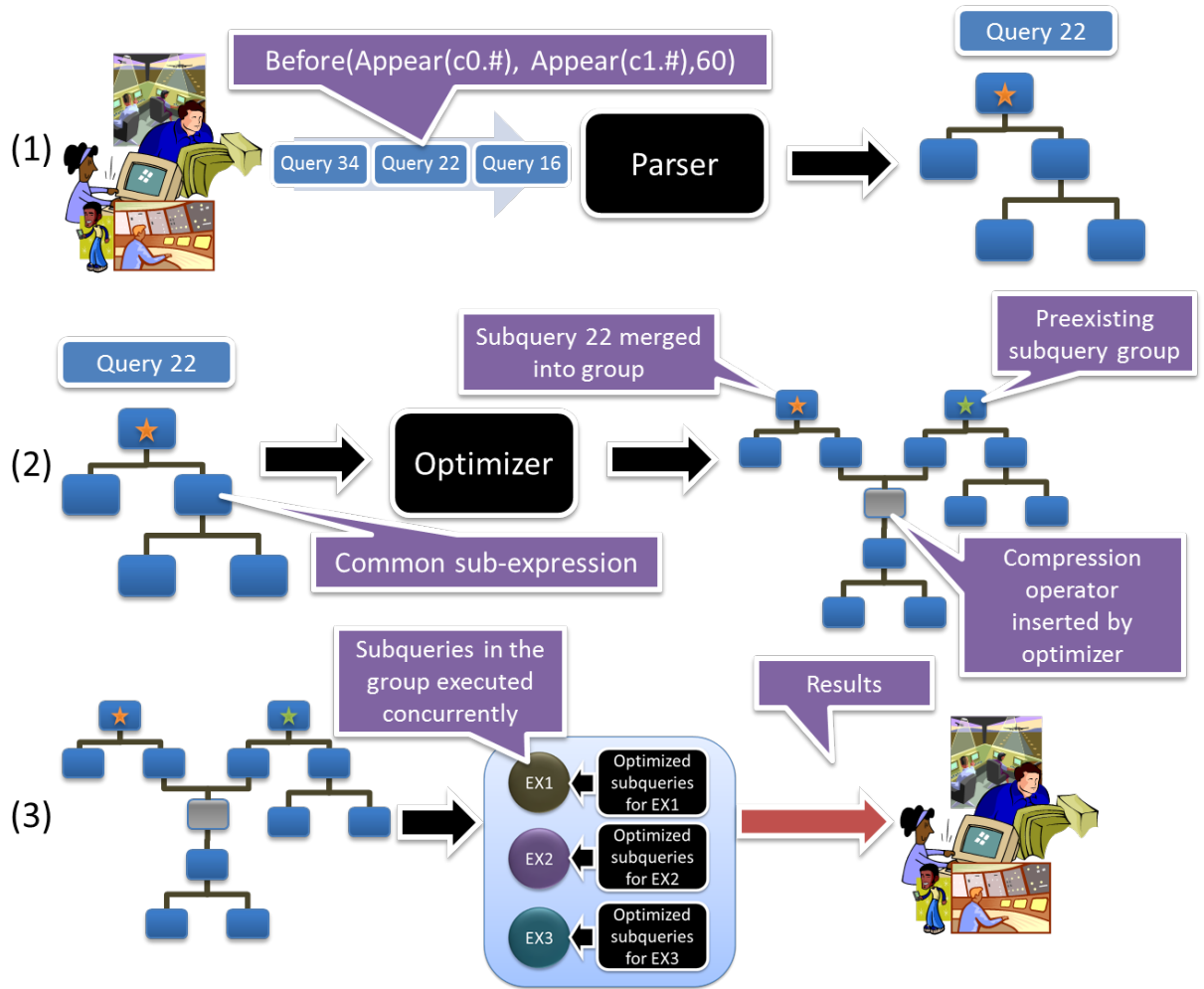
In the LVBMS subqueries on each spatial processing layer host are executed in groups, called query groups. An overview of the LVDBMS query optimization process is provided in Figure 50: in (1) the query is posed by a user in LVQL. The parser translates it into an internal query tree (equation), which is inputted to the query optimizer. (2) The query optimizer considers the other subqueries that are currently executing on a particular spatial processing layer node and

creates an execution plan that is tailored to the runtime environment (a specific query group) on that node. (3) Queries in the group are executed and results returned to the user.

When the query optimizer considers a specific spatial processing layer node to execute a query on, it considers the current queries executing on that node to ensure a subquery can be executed and access all of the video streams it needs, to compute its result. Queries are grouped and executed together based upon their ability to share computation and input data streams. If some sharing is possible among subqueries, they are grouped together and their execution trees merged by inserting compression operators (in the case of execution overlap) or caching operands (in the case when they share a common data stream).

#### *LVDBMS Query Optimization*

The LVDBMS query optimization environment consists of two primary components, the run-time query optimizer and the query execution engine. The query optimizer is designed to consider the currently executing subqueries on a particular node and, if possible, merge a new subquery into an existing query group. The execution engine is multithreaded and each query group runs within its own processing thread. The subquery merging procedure is designed to permit the newly introduced subquery to execute and either share execution results or an input data stream with other subqueries. If a new subquery does not share any commonalities with currently executing subqueries, a new query group is allocated for it. In order to merge subqueries the query optimizer can rewrite the query tree and merge queries by injecting one of two new nodes into the tree (which also serve as subquery merge points). The multithreaded query execution engine is designed to take advantage of modern CPU's which have multiple cores and large caches.

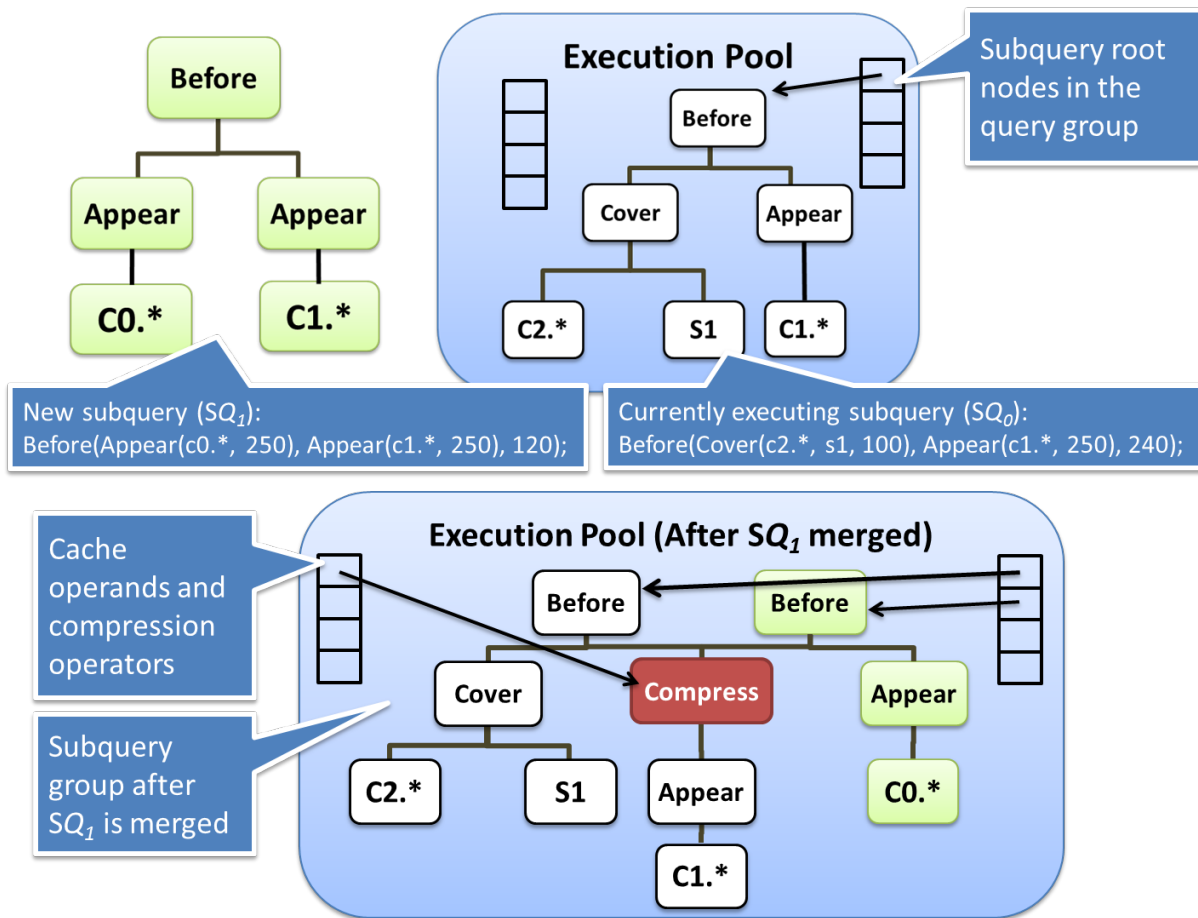


**Figure 50. Transformations undergone by a query; from query to subquery to results.**

The subquery merging process involves comparing two tree-like graph structures to identify the largest common sub-graph (Jungnickel, 2004). The comparison is between a new subquery, which is oriented as a tree, and the subqueries in the query group, which are groups of connected trees where each connected subquery has its own distinct root. If the new subquery is the same as one of the currently executing subqueries, or the new subquery shares at least one common operator, then a new compression operator is allocated and merged into the subquery tree (unless there already exists one at that point). The new subquery is merged to point to that compression operator, and the existing subquery below the merge point is moved below the new



compression operator. Note that in order for this merging to occur, in addition to the common nodes being the same operators, they must have identical operands (in terms of both query streams referenced and any associated thresholds). If the match is only an operand that specifies a particular data stream, then a caching node is allocated, merged into the existing tree, the existing operand moved below it, and the new subquery references the caching node in the place of its previously existing operand that had corresponded to the video stream. This is illustrated in Figure 51. By injecting this caching operand, fetches to data outside of the query group are reduced. Data fetches are required in order for the query operators to obtain objects and metadata that pertain to current video frames.



**Figure 51. Before and after illustrations of a subquery merged into a query group.**

Once the query optimizer has received a query, its next task is to decompose the query into subqueries for execution on spatial layer processing hosts. Subqueries are created based upon the video streams available to a spatial processing layer host. Given a subquery and a spatial processing layer host upon which it will execute, the next task is to determine if there is overlap between the new subquery and any existing query groups. If the subquery graph and query group graphs contain a common subgraph the subquery will be merged into the query group else a new query group will be allocated containing the subquery.

Subquery matching into a group is performed by creating a hash that represents each possible subtree in the subquery graph, and comparing them to hashes representing each possible subgraph corresponding to the query group. Note that the hash structure for the query group can be cached between uses and maintained along with the subqueries in the group in order to reduce the overhead of recreating the hash structure for each new subquery. If the matching subgraph contains operators (in addition to stream operands) then a new compression operator is allocated and inserted as the merge point, and the query group data structures are updated to track and maintain the new compression operator. If the match is only an operand, then a caching operand is allocated and inserted into the query group. This process is illustrated in Figure 51 with the common LVQL sub-expression being *Appear(c1.\*, 250)*.

#### *LVDBMS Query Execution Environment*

While the query optimizer translates a query from the logical to the physical algebras, the execution engine executes the operators that comprise the complex query and enqueue the subquery evaluations for transmittal to stream processing layer nodes. (A complex query is a query that implements a number of different data processing algorithms.) Subqueries are executed periodically, for example once each second. The execution engine is comprised of two

components, the execution engine that executes the query groups, and a metadata manager which maintains the state of the associated video streams; i.e. the current frame for each stream, the objects that are visible in the current frame, etc. The metadata manager also contains locking facilities to ensure that the metadata is maintained in a consistent state despite the numerous reader and writer threads that keep the metadata up to date. When a subquery is evaluated, the first step of execution is for the operand nodes to perform a fetch from the metadata store. Each operand corresponds to a video stream, and can contain some filter criteria (for example, to fetch only dynamic objects or static objects or a particular object, for example). Thus each operand fetches the current frame and object information pertaining to its corresponding stream. It then returns that data to the operand's calling operator. Thus query execution is a data-driven process beginning at the base of the execution graph with the operands and ending with the root node of the subquery returning its subquery evaluation result. This subquery result is then enqueued for transmittal to the corresponding stream processing layer host for further processing; this process is illustrated in Figure 52. In the case a query was decomposed into multiple subqueries, the subquery results must be combined to produce the final query result.

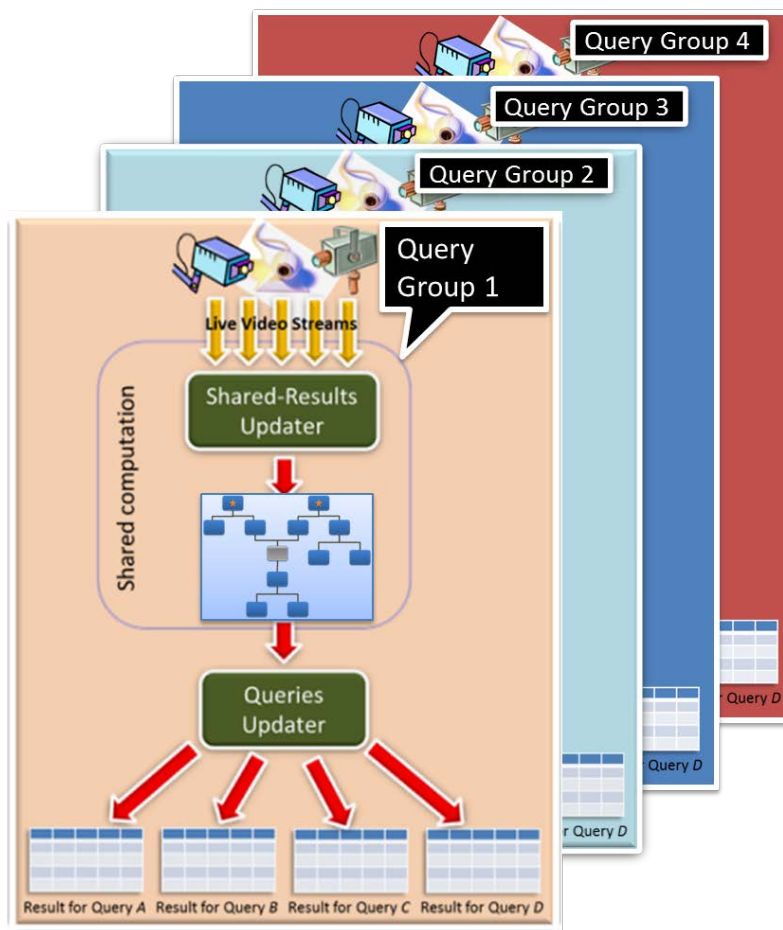
### *Cost Estimation*

Queries expressed in physical algebra have specific processing algorithms and data structures associated with the processing steps. Since the processing algorithms are known, cost estimates can be associated with the various algorithms (i.e. operators and operands in the physical algebra). The cost estimate for each operator can be estimated based upon the resources they require to return an intermediate result at each step of query processing. For the LVDBMS these costs are based upon the CPU and primary memory footprint that is required for them to evaluate their input. Thus, the various stages of the query can be summed in order to compute a

cost estimate for the entire query. By summing the subqueries in a query group, a cost estimate can be derived for the query group.

**Table 14. Evaluation costs of various operators and operands**

Operators and operands		Evaluation cost
Operands	Caching	2
	Dynamic, Static	5
	Cross-camera	20
Operators	Appear	5
	And/Or/Not	1
	Before/Meet	10
	North/South	6
	Compression	1



**Figure 52. Example depicting four query groups in the LVDBMS query execution engine.**

Table 14 presents representative cost estimates for select LVQL operands and operators. The Boolean operators execute the simplest algorithms in order to compute their result and thus have the lowest cost estimate. Temporal operators accept three arguments; two arguments correspond to other operators the temporal operator is being evaluated over, and the third operator is a window length specifying a maximum buffer of intermediate evaluation results the temporal operator will consider when evaluating its return value. For example, the *Before()* operator maintains two buffers with intermediate evaluation results of each of its operator parameters which have a maximum capacity equal to its window length argument. The *Appear()* operator does not need to maintain execution state history between invocations and requires some CPU cycles to evaluate the objects pertaining to the video stream that correspond to the operand that is specified when it is invoked. The *Appear()* operator simply iterates through its input and determines which of the current objects, if any, have an area that is equal to or greater than the value specified in its threshold argument. By measuring the cost of executing a query group, and then adding a new subquery, the execution cost that is saved by sharing computation between the new subquery and the query group can be measured and quantified.

### Experimental Study

This section provides results pertaining to the execution efficiencies gained by query optimization. For these results a series of pre-recorded traffic videos was utilized; an example frame is provided in Figure 53. The particular frame of video depicted in this figure is from the video stream with identifier 2, identified as *c2* in the queries in Figure 54 and shows a static object, *565b46*, drawn in the lower right corner of the frame. *c2.s565b46* is a static object drawn by the user. *cn* refers to a particular video stream.



**Figure 53.** Example video frame showing a busy road and a static object (the blue rectangle).

```
Action 'q1' on Before(Appear(c0.*,250), Appear(c1.*,200),200);

Action 'q2' on NOT Before(
    Appear(c0.*,250),
    North(c2.s565b46, c2.*,250), 120);

Action 'q3' on Before(
    West(c2.*, c2.s565b46, 10),
    North(c2.s565b46, c2.*,250), 120);
```

**Figure 54.** Evaluation 1: queries utilized for testing the query optimizer.

```
Action 'q4' on Before(
    West(c2.*, c2.s565b46, 10),
    Before(Appear(c0.*,250),Appear(c1.*,200),200),120);

Action 'q5' on Before(
    West(c2.*, c2.s565b46, 10),
    Before(
        Before(Appear(c0.*,250),Appear(c1.*,200),200),
        Appear(c1.*,200),200),
    120);

Action 'q6' on Before(
    West(c2.*, c2.s565b46, 10),
    Before(
        Before(Appear(c0.*,250),Appear(c1.*,200),200),
        Before(Appear(c3.*,250),Appear(c0.*,400),200),
        200),
    120);
```

**Figure 55.** Evaluation 2: complex queries for optimization evaluations.

Pre-recorded videos were utilized for this performance study in order in order to allow experiments to be re-ran with different system parameters. Two evaluations are provided; each evaluation consists of a series of queries being submitted to the LVDBMS. For each evaluation, when the initial query is submitted, no other queries exist in the system. A new query group is allocated for the first query. Subsequent queries in the evaluation run are then merged into the query group, and the performance of this optimization step is reported in the tables provided in this section. The first evaluation run utilizes three simple queries, provided in Figure 54. For the second evaluation run, more complex queries were selected which have more operands and operators and provide a test of the query optimizer with complex query trees; provided in Figure 55. The tables in this section that provide query costs are given in terms of the query cost estimation metric detailed in the previous section.

The first evaluation run the queries provided in Figure 54 are submitted sequentially to the LVDBMS. Internal performance counters are implemented in the LVDBMS prototype to allow for query execution statistics to be collected. Fewer operands and operators mean lower computational requirements to evaluate queries, and also indicate the quality of the optimization. Snapshots of selected counters are presented in Table 15 at five-second intervals. After the first query is submitted, subsequent queries are submitted in between execution counter snapshots. The first query sent is *q1*, consisting of a *Before()* operator and two *Appear()* operators. As can be seen from the first row of the table, the physical algebra contains three operators and two operands (the operands corresponding to the two video streams, *c0* and *c1*). In their unoptimized forms the second query (*q2*) contains four operators and two operands, the third (*q3*) has three operators and again two operands, the first operand (*c2.\**) refers to all dynamic objects in video stream *c2*, and the second referring to a specific static object by its identifier. As the performance

statistics are collected at five-second intervals, the 10 data requests observed in the first row of the table correspond to the two operands performing five requests each. The next table, Table 16, provide the same performance counters, but with the query optimization logic enabled such that after the first query is submitted, subsequent queries may be merged into the query group and thus duplicate execution is eliminated. The results in this table show that due to the merging, facilitated by execution compression operators and caching operands, operator executions and data fetches are reduced. Table 17 provides a side-by-side comparison of the two scenarios; without and with the query optimization logic. From the cumulative summary results presented in this table the cost savings from the query optimization is apparent, as indicated in the last column of the table. Results from the second evaluation are presented in Table 18, corresponding to the queries indicated in Figure 55. These queries are significantly more complex than the first set. However, they also contain subqueries with significant operator and operand overlap and thus can achieve potentially large cost savings that the query optimizer can recognize and leverage.

**Table 15. Evaluation 1, performance counters without optimization**

# Operands	# Operators	# Executions	# Data requests	Evaluation cost
2	3	15	10	35
5	7	46	32	78
9	10	93	73	127

**Table 16. Evaluation 1, performance counters with optimization**

# Operands	# Operators	# Executions	# Data requests	Evaluation cost
2	3	15	10	35
4	7	42	28	69
6	10	80	56	102



**Table 17. Evaluation 1, cost efficiencies gained**

Query	Unoptimized query		Optimized query		
	Cost	Cumulative cost	Cost	Query group cost	Optimization saving
<i>q1</i>	35	35	35	35	N/a
<i>q2</i>	43	78	34	69	9
<i>q3</i>	49	127	33	102	25

**Table 18. Evaluation 2, cost efficiencies gained**

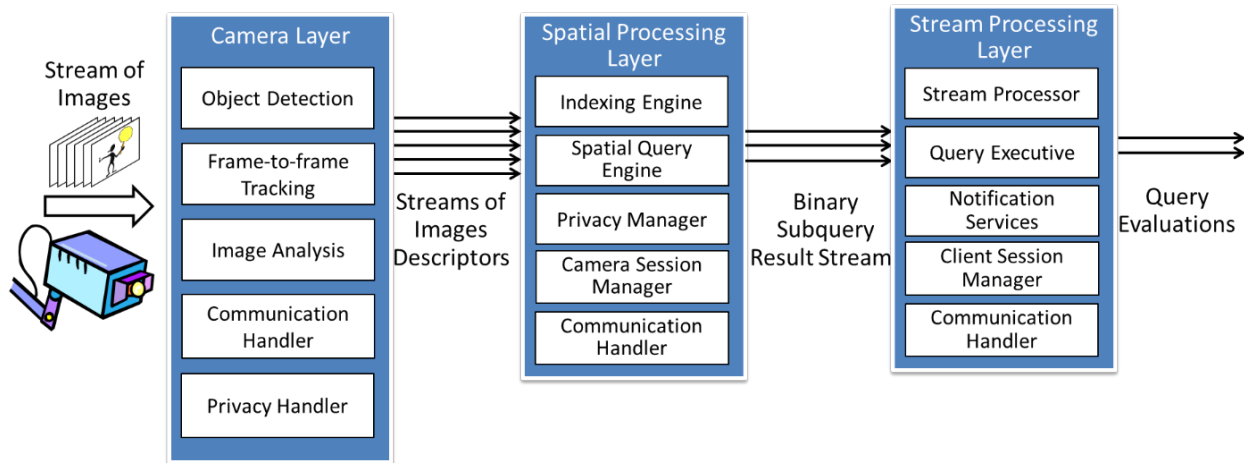
Query	Cost	Cum. cost	Query group cost	Opt. saving	Operands	Operators	Compression
							Operators
<i>q3</i>	67	67	67		4	5	0
<i>q4</i>	92	159	125	33	7	10	1
<i>q5</i>	117	276	208	91	11	17	2

### Summary

The translation of query algebra from the logical to the physical, and selection of the final query evaluation plan are critical steps in a database management system. These important steps are generally performed by the query optimizer. The execution plan specifies the data flow and steps the database management system will undertake to evaluate the query. The selection of poor query evaluation plans will result in poor system performance. Query optimizers in traditional relational databases consider a number of statistics pertaining to the data and table structures that the queries it optimizes are posed over. However, in an LVC environment much of this information is either not available or cannot be leveraged in a stream processing environment.

This chapter presents a query optimizer and associated execution environment that is designed for the LVC environment. It performs query optimization at runtime, taking a new query and finding any possible overlap with the existing queries in the system and rewriting the new query in order to minimize duplicate subexpressions and optimize the utilization of the query execution engine. Results presented in this chapter show that the query optimization

methods that were designed for the LVC environment and implemented in the LVDBMS prototype reduce query execution overhead by merging the physical algebra query trees. To facilitate the performance evaluation and the impact of the query optimization, a query cost metric was derived and used to present optimization performance results.



**Figure 56. The lower three LVDBMS tiers, showing major components.**

## CHAPTER 7: LVDBMS PROTOTYPE

### Introduction

The LVDBMS prototype is the testbed for implementing LVC concepts and gauging the feasibility and performance of selected algorithms, data structures and processes. The C# programming language and .NET runtime environment were selected for implementing the LVDBMS, along with the EMGU.NET (<http://www.emgu.com>) and OPENCV (Bradski & Kaehler, 2008) libraries. Early versions of the LVDBMS prototype utilized the high-performance Intel IPP library (<http://software.intel.com/en-us/intel-ipp>) and Intel compiler to take advantage of hardware single-instruction multiple data instructions available in certain CPUs.

### Prototype System Architecture

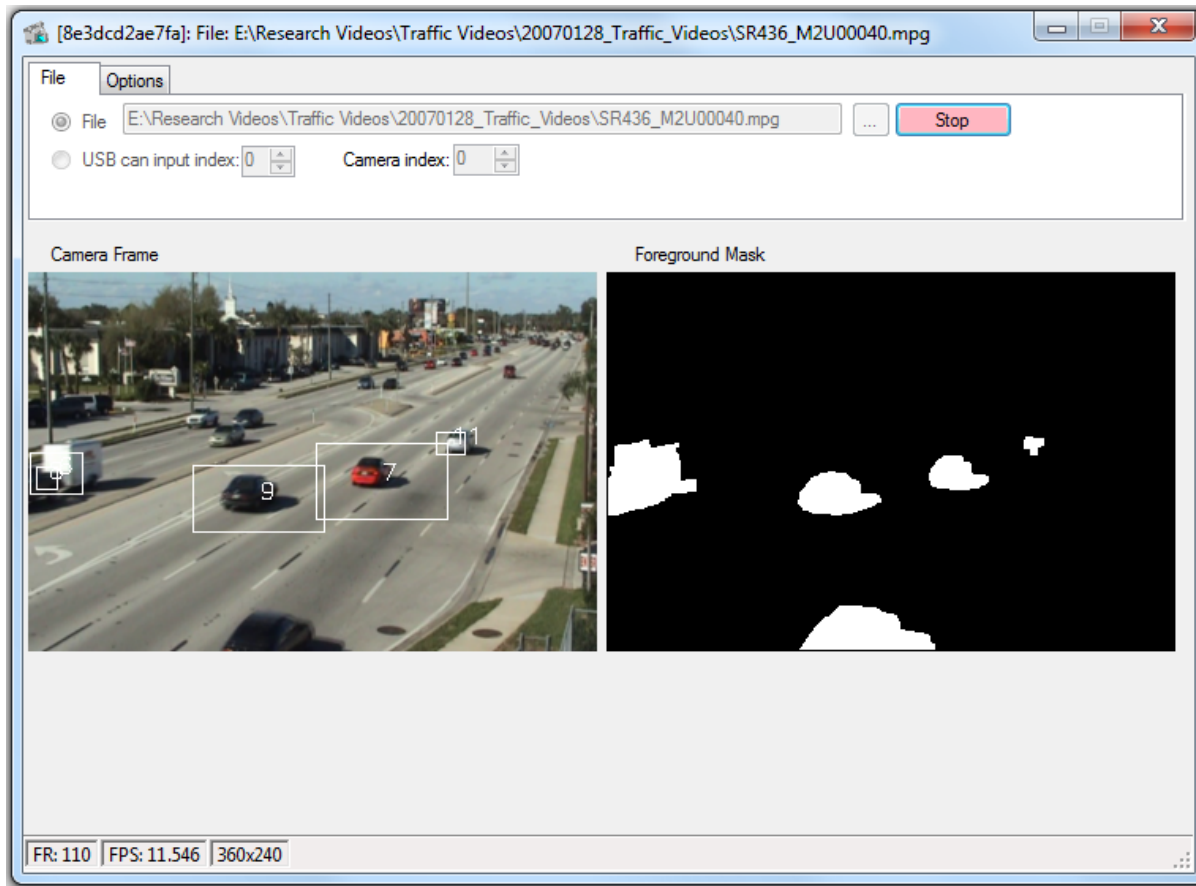
The LVDBMS is architected as a 4-tier distributed application, illustrated in Figure 20. In order to meet demands for increased capacity, it is designed to scale by adding additional processing nodes in the various layers. Each software layer utilizes a web services communication interface to facilitate communication between the tiers. The four LVDBMS application tiers are (1) the camera layer, (2) spatial processing layer, (3) stream processing layer and (4) client layer. The lower three tiers, along with major components, are illustrated in Figure 56.

#### *Camera Layer*

The camera layer is comprised of the image capture device and a software application referred to as an adapter. The image capture device or camera records a scene. In order to increase the flexibility of the LVDBMS with regards to the types of camera hardware it can interface with, no processing capabilities are assumed of the camera other than to record video

and transmit it to an adapter that can be coded to interface with that particular camera hardware. (Note that some cameras have compute facilities capable of running image processing algorithms such as background detection and object tracking.) The camera adapter runs on a host computer and can be connected to the camera via a physical communication medium that both the camera and host support, such as USB, a wireline network, coaxial cable, etc. A screen capture of the camera adapter GUI is provided in Figure 57.

The camera adapter receives the raw imagery from the camera and adds metadata pertaining to the frames and objects observed. By raw imagery it is mean a temporally ordered series of frames of video that is simply data; a two-dimensional matrix of pixel values corresponding to what was sensed by the imaging device. The camera adapter implements algorithms to model the scene background. As salient objects move across the background, a segmentation algorithm attempts to determine which pixels belong to the scene background and which pixels do not. Pixels that are not recorded as part of the background are grouped together into a blob. Blobs are assigned an identifier that is unique to each instance of a camera adapter (and thus, unique to each video stream) A tracking algorithm, referred to in this work as a frame-to-frame tracking algorithm. The frame-to-frame tracker maintains correspondences between objects, as they move, from one frame to the next. By maintaining these correspondences, the image analysis module can compute a feature vector corresponding to the visual appearance of each object in each frame of video. The output of the image analysis module is metadata describing each frame, e.g. a monotonically increasing frame number, timestamp when the frame was received, the number of objects in the frame and their identifiers, privacy filters associated with the camera, etc.



**Figure 57. Camera adapter, simulating a video stream from a pre-recorded video.**

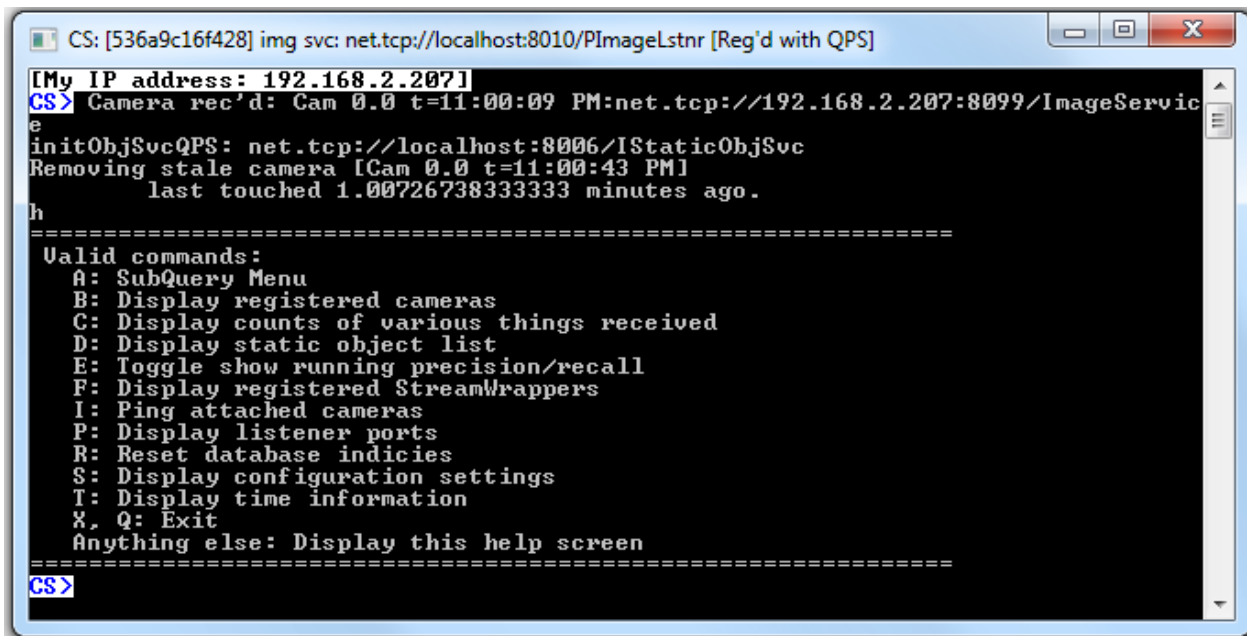
The communication handler establishes and maintains communication between the camera adapter and the spatial processing layer. On startup it registers the camera with the spatial processing layer host, including a description of the camera and its location. It contains queues for buffering and sending imagery and metadata to the spatial processing layer host. It also receives commands and camera-level privacy filters.

### *Spatial Processing Layer*

Spatial processing layer hosts receive imagery and metadata from camera adapters and maintain mappings associating video streams with camera adapters, indices of frames and

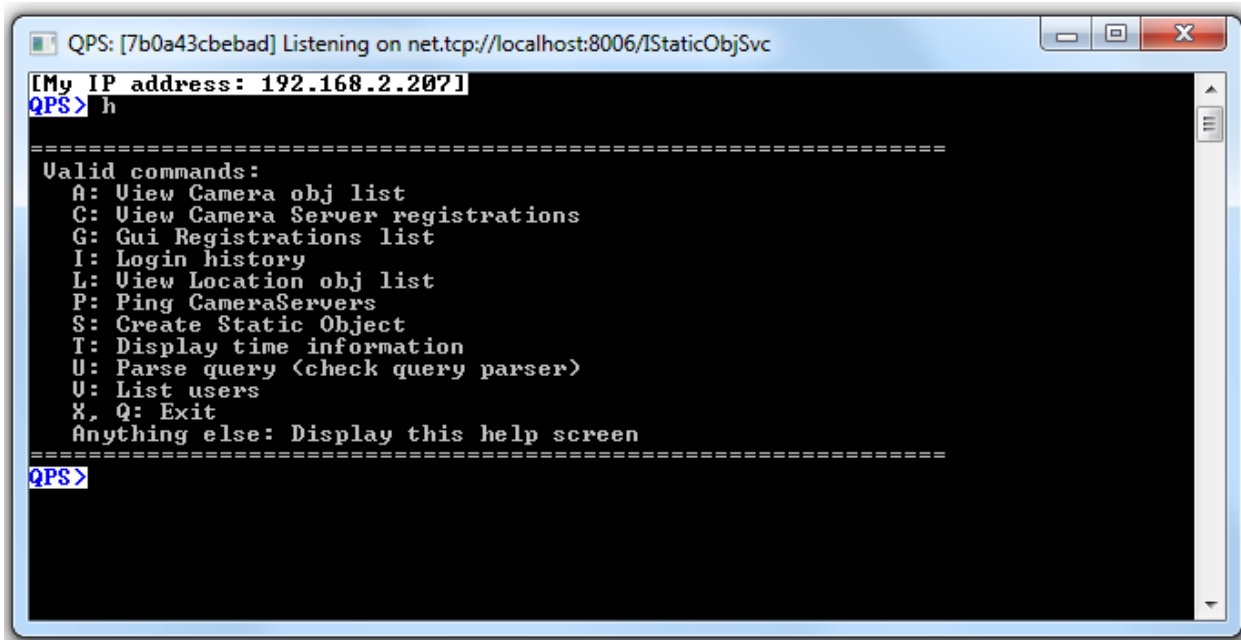
corresponding objects, a security matrix pertaining to which users and user groups can access cameras.

The communication handler manages communication sessions between camera adapters and spatial processing layer hosts, between spatial processing layer hosts and stream processing layer hosts and between itself and clients to serve imagery to clients viewing video streams directly. The privacy manager stores privacy filters for streams and views associated with the respective spatial processing layer host. These privacy filters are applied to imagery served to clients. The spatial query engine executes subqueries within query groups with execution pool as described previously. A screen capture from the GUI of a spatial processing layer service is given in Figure 58. From the output shown in the figure, a camera adapter registration was received and later timed out after the heartbeat service could not communicate with it for one minute.



```
CS: [536a9c16f428] img svc: net.tcp://localhost:8010/PImageLstnr [Reg'd with QPS]
[My IP address: 192.168.2.207]
CS> Camera rec'd: Cam 0.0 t=11:00:09 PM:net.tcp://192.168.2.207:8099/ImageService
initObjSvcQPS: net.tcp://localhost:8006/IStaticObjSvc
Removing stale camera [Cam 0.0 t=11:00:43 PM]
      last touched 1.0072673833333 minutes ago.
h
=====
Valid commands:
A: SubQuery Menu
B: Display registered cameras
C: Display counts of various things received
D: Display static object list
E: Toggle show running precision/recall
F: Display registered StreamWrappers
I: Ping attached cameras
P: Display listener ports
R: Reset database indices
S: Display configuration settings
T: Display time information
X, Q: Exit
Anything else: Display this help screen
=====
CS>
```

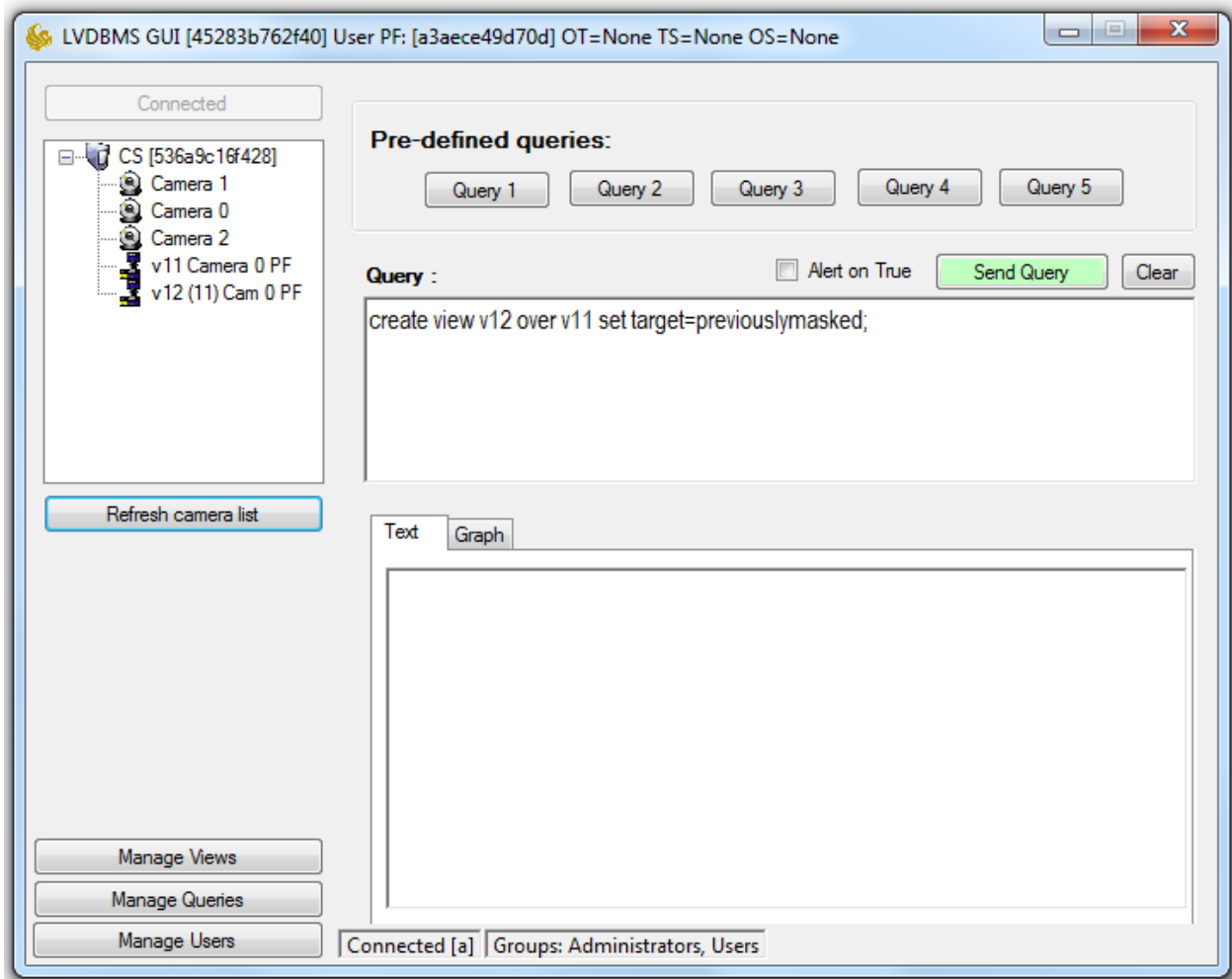
Figure 58. The camera server service, which runs in the spatial processing layer.



**Figure 59.** The query processing service, which runs in the stream processing layer.

#### *Stream Processing Layer*

Stream processing layer hosts receive subquery results that were evaluated on spatial processing layer hosts and compute the final query result each evaluation cycle. The query executive maintains metadata pertaining to the available spatial processing layer hosts and which video streams they are associated with; the last time communication was received from each host, etc. The client session maintains communication sessions with clients, including a periodic “heartbeat.” If communication with a client is lost for a period of time, any queries associated with that client can be aborted. The client session manager also maintains a list of user-level privacy filters associated with specific users. A screen capture showing the GUI of the spatial processing layer service is provided in Figure 59.



**Figure 60. The LVDBMS user client, showing a connection to a camera server and three video streams and two views.**

### *Client Layer*

The LVDBMS client provides the interface between human operators and the LVDBMS query processing layers. Users can browse available cameras, define views and privacy filters and submit queries to stream processing layer hosts. It also receives query results and provides them to the user, including notifications that are configured as a part of a query. Users who are members of the administrators group can view all active queries in the LVDBMS as well as perform other administrative functions such as privacy filter, user and user group management.



The screen capture of the LVDBMS provided in Figure 60 shows a client GUI connected to one camera server (the LVDBMS service that runs in the spatial processing layer) which has three video streams available for viewing or querying. One can observe a view,  $v11$ , defined over stream  $0$ , which is also associated with a privacy filter. A second view,  $v12$ , is defined over  $v11$  and the LVQL for defining this view is shown in the query window.

### Experimental Study

This section presents additional performance results pertaining to various aspects of the LVDBMS prototype. In order for objects to be identified across video streams, a number of internal data structures must be maintained, and algorithms developed to maintain them. Additionally, the some components of the LVDBMS prototype is structured as a series of pipeline stages, such that each stage implements some specific processing steps in the overall data flow. Most pipeline stages are driven by their own processing threads, thus, concurrency control mechanisms and thread-safe data operations are required in order to ensure proper system state is maintained. In many cases concurrency control reduces certain aspects of the system to a serial ordering (e.g. two threads cannot write to the same memory location at the same time and produce a deterministic result). Thus, waiting for and acquiring locks consumes some small portion of the computational overhead incurred in maintaining system state. These system-oriented results that did not properly fit into an earlier section is what are detailed in this section.

#### *Query Processing Performance*

Results presented in this section pertain to evaluating individual subqueries (i.e. not subqueries oriented in query groups). As only subqueries are evaluated, this time does not include communication overhead to combine subquery results in the stream processing tier.

The continuous queries (i.e. subqueries) in the LVDBMS are evaluated periodically. The intervals in which they are evaluated is referred to as the resolution of the query. The amount of time required to evaluate each subquery should be less than its resolution, else the subquery evaluations will be missed due to its evaluation time running over into the next evaluation time slot. To test the subquery evaluation performance of the execution engine and related metadata structures, an evaluation scenario comprised of executing five subquery simultaneously for a period of 120 seconds over ten randomly selected videos was performed. The test was repeated for ten different videos, with a query resolution of one second. The time taken for the evaluation procedure to conclude is recorded at one-second intervals. These evaluation times are plotted for each video in Figure 61. Additional details pertaining to the evaluation times is presented in Table 19. In order to simplify the information presented in the table, it has been normalized relative to a single video by dividing the values by five. The evaluation times are significantly less than the query resolution, and in most cases the standard deviation is larger than the average execution time. We feel this behavior is due to processes having to wait to acquire various read and write locks, and for .NET runtime memory allocations. Evaluating a subquery entails first checking that the subquery is runnable. (For example, if an exception occurs during the evaluation of a subquery it will be aborted. Such an exception could occur if a video stream was to go offline.) Next, the query root node is signaled to evaluate, resulting in a recursive query tree traversal down to the operands. The operands execute their data fetches (which incurs read locks on the metadata structures to obtain frame and object information). The recursive tree traversal continues, with evaluation results returned from the leaves of the tree back up to the final computation in the root node. Each subquery is evaluated in such fashion and the root

node's evaluation result is enqueued for transmittal to the stream processing layer host. Note that the query execution engine does not halt for the results to be transmitted via the network.

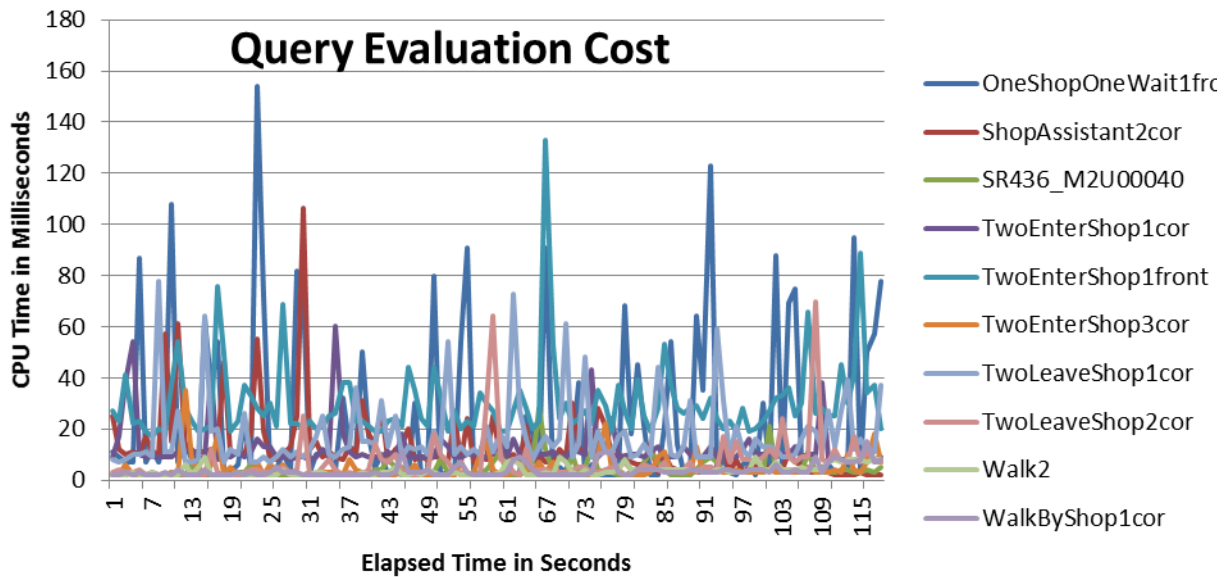
### Summary

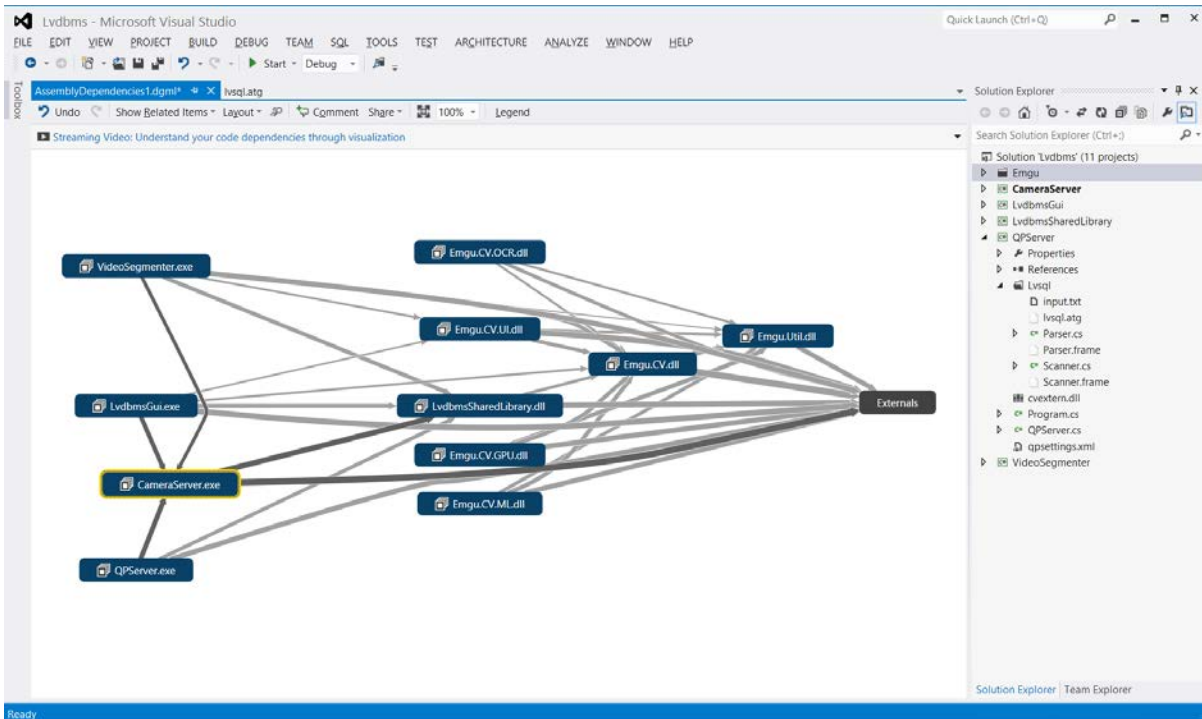
This section provided an overview of the LVDBMS prototype implementation of an LVC database. Its development began approximately three years ago as a C++ application in Microsoft Visual Studio 2008 utilizing OpenCV and the Intel Performance Primitives library. The decision was made to port the application to C# in order to leverage the *Language Integrated Query (LINQ)* features for performing various SQL-like functions on internal data types such as arrays. Due to the real-time performance requirements, in-memory data structures such as arrays, hash tables and lists are used, as opposed to utilizing a relational database that would involve transactions and writing data to hard disk. With the port to C#, the OpenCV wrapper EMGU CV was utilized for low-level computer vision algorithms and related data structures. The current version of the LVDBMS is developed in C# version 4.5 and Visual Studio 2012; Figure 62.

In this chapter the LVDBMS is described, and performance results for query evaluations were provided. Overall the LVDBMS successfully functions as a test bed for implementing and testing LVC algorithms in order to advance the state of the art in stream processing environments.

**Table 19. Average query evaluation in milliseconds of CPU time, by video**

Movie	Performance			
	Min	Max	Standard Deviation	Average
SR436_M2U00040 (3)	0.40	5.60	0.73	0.78
OneShopOneWait1front (2)	0.40	30.81	6.26	4.58
ShopAssistant2cor (2)	0.40	21.24	2.72	2.49
TwoEnterShop1cor (2)	1.60	12.00	1.68	2.42
TwoEnterShop1front (2)	3.40	26.60	3.05	5.97
TwoEnterShop3cor (2)	0.40	7.00	0.90	0.87
TwoLeaveShop1cor (2)	1.40	15.60	2.75	3.47
TwoLeaveShop2cor (2)	0.40	14.00	1.86	1.38
Walk2 (2)	0.40	1.80	0.40	0.72
WalkByShop1cor (2)	0.40	3.00	0.49	0.73

**Figure 61. Evaluation costs for a selection of queries in milliseconds; plotted at one-second intervals.**



**Figure 62.** Screen capture of Visual Studio 2012 IDE depicting a dependency graph of LVBMS assemblies.

## CHAPTER 8: CONCLUSIONS

### Summary of Contributions

Due to increasing needs to monitor areas for purposes such as proactive maintenance, security and quality assurance, there are increasingly more live video streams that need to be monitored. This manuscript presents work pertaining to LVC and presents the LVDBMS and its capabilities. The LVDBMS allows for the monitoring and detection of complex events which can be observed by cameras. LVQL, a declarative high-level query language facilitates the specification of complex events to be defined and monitored. Users can leverage LVQL for ad hoc monitoring tasks and application developers can leverage it for stream processing application development, similar to how traditional business applications utilize relational database systems for data processing and storage.

As cameras and imaging sensors continue to be installed, more and more of our lives can be captured, analyzed and correlated by computer systems. A privacy framework is presented which implements privacy policies that can be applied in real time to live video streams. This facilitates the real-time dissemination of privacy-aware video content. Efficient query processing techniques, web-service communication and a scalable 4-tier application architecture provide for a solution that can scale accommodate large camera networks. Experimental results show that the LVDBMS can effectively recognizes events observed in video streams, implement privacy policies and efficiently processing queries.

In conclusions, major contributions feature a prototype LVC implementation including:

- LVQL, a high-level query language for specifying events and interacting with the LVDBMS,

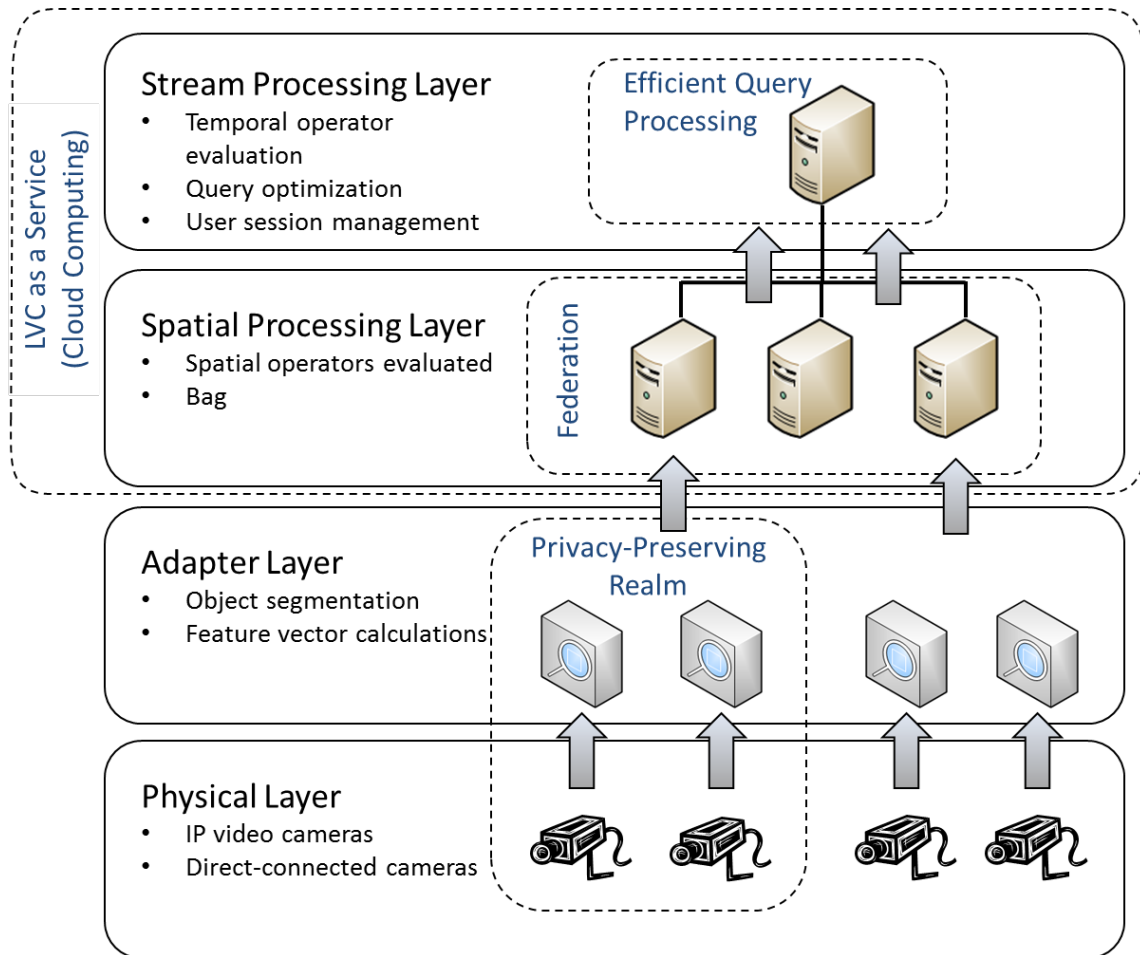
- A privacy-aware infrastructure permitting for the specification of privacy policies and their implementation in a real-time stream processing environment, and
- Efficient query processing and execution techniques to maximize compute memory resource usage.

### Future Work

There are a number of directions in which work pertaining to the LVDBMS can continue. It is hoped that advances published in this manuscript show the feasibility of LVC and excite future researchers to study real time stream processing platforms and continue to advance the state of the art. Figure 63 provides a visual overview of the LVDBMS software architecture. Annotations in dotted rectangles highlight LVDBMS components and indicate potential future areas of work. A summary of possible future extensions are as follows:

- **LVC as a Service:** The reformulation of the stream and spatial layers into a collection of loosely-connected components that are amenable to being hosted in a cloud computing environment. By re-architecting these upper layers, capacity can be added where needed in order to cope with demands and to mitigate the impact of intermittent hardware failures. Protocols for managing and processing units of work will need to be developed.
- **Efficient Query Processing:** Efficient utilization of computational resources can be facilitated by improved algorithms and data structures designed to more efficiently process requests and detect events. Such work could entail additional algebraic transformations of queries, and parallel query algorithms for highly parallel compute hardware such as GPUs.

- **Privacy-Preserving Realm:** Additional work can always be done to ensure privacy. The current prototype can leak privacy information in the presence of tracking or object segmentation errors. Additional privacy advancements could extend the PSL to better leverage events and context observed in video streams.
- **Federation:** Protocols for establishing relationships between disparate implementations to permit further sharing of resources. Trust-based relationships can be defined to specify priority in the presence of resource contention or the specification of privacy policies.



**Figure 63. Illustration of potential future works pertaining to LVC and the LVDBMS.**



## LIST OF REFERENCES

- Adali, S., Candan, K. S., Chen, S. S., Erol, K., & Subrahmanian, V. (1996). The advanced video information system: data structures and query processing. *Multimedia systems*, 4(4), 172–186.
- Adam, N. R., & Worthmann, J. C. (1989). Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys (CSUR)*, 21(4), 515–556.
- Ahanger, G., & Little, T. D. C. (1996). A survey of technologies for parsing and indexing digital video. *Journal of visual Communication and image representation*, 7(1), 28–43.
- Ahmad, Y., & Cetintemel, U. (2004). Network-aware query processing for stream-based applications (Vol. 30, pp. 456–467). Presented at the Thirtieth international conference on Very large Databases, VLDB Endowment.
- Ahmedali, T., & Clark, J. J. (2006). Collaborative multi-camera surveillance with automated person detection (pp. 39–39). Presented at the 3rd Canadian Conference on Computer and Robot Vision, IEEE.
- Aho, A. V., Hopcroft, J. E., & Ullman, J. (1983). *Data structures and algorithms*. Addison-Wesley Longman Publishing Co., Inc.
- Aho, A. V., & Ullman, J. D. (1972). *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc.
- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle (Vol. 1, pp. 267–281). Presented at the Second International Symposium on Information Theory, Akademiai Kiado.
- Anderberg, M. R. (1973). *Cluster analysis for applications*. DTIC Document.
- Antani, S., Kasturi, R., & Jain, R. (2002). A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. *Pattern recognition*, 35(4), 945–965.
- Aref, W. G., & Ilyas, I. F. (2001). Sp-gist: An extensible database index for supporting space partitioning trees. *Journal of Intelligent Information Systems*, 17(2), 215–240.
- Aref, W. G., & Samet, H. (1994). Hashing by proximity to process duplicates in spatial databases (pp. 347–354). Presented at the Proceedings of the third international conference on Information and knowledge management, ACM.
- Aref, W., & Samet, H. (1992). Uniquely reporting spatial objects: yet another operation for comparing spatial data structures (Vol. 1, pp. 178–189). Presented at the Proceedings of the Fifth International Symposium on Spatial Data Handling.

- Aved, A. J., & Hua, K. A. (2012). A general framework for managing and processing live video data with privacy protection. *Multimedia Systems*, 18(2), 123–143.
- Aved, A. J., Hua, K. A., & Gurappa, V. (2011). An Informatics-Based Approach to Object Tracking for Distributed Live Video Computing. In *Multimedia Communications, Services and Security* (pp. 120–128).
- Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems (pp. 1–16). Presented at the Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM.
- Babu, S., & Widom, J. (2004). StreaMon: an adaptive engine for stream query processing (pp. 931–932). Presented at the ACM SIGMOD international conference on Management of data, ACM.
- Bach, J. R., Fuller, C., Gupta, A., Hampapur, A., Horowitz, B., Humphrey, R., ... Shu, C. F. (1996). The Virage image search engine: An open framework for image management (pp. 76–87). Presented at the SPIE Storage and Retrieval for Image and Video Databases IV.
- Baeza-Yates, R., Cunto, W., Manber, U., & Wu, S. (1994). Proximity matching using fixed-queries trees (pp. 198–212). Presented at the Combinatorial Pattern Matching, Springer.
- Barbieri, M., Mekenkamp, G., Ceccarelli, M., & Nesvadba, J. (2001). The color browser: a content driven linear video browsing tool (pp. 627–630). Presented at the IEEE International Conference on Multimedia and Expo, IEEE.
- Beckmann, N., Kriegel, H. P., Schneider, R., & Seeger, B. (1990). *The R\*-tree: an efficient and robust access method for points and rectangles* (Vol. 19). ACM.
- Belhumeur, P., Hespanha, J., & Kriegman, D. (1996). Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *European Conference on Computer Vision*, 43–58.
- Bellman, R. E. (1986). *The Bellman Continuum: A Collection of the Works of Richard E. Bellman*. World Scientific Publishing Company Incorporated.
- Bellman, R., & Kalaba, R. (1959). On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2), 1–9.
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 509–517.
- Bentley, J. L. (1977). *Algorithms for Klee's rectangle problems*. Technical Report, Computer.
- Berclaz, J., Fleuret, F., Turetken, E., & Fua, P. (2011). Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9), 1806–1819.

- Bern, M. (1993). Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(2), 95–99.
- Beucher, S., & Lantuejoul, C. (1979). Use of watersheds in contour detection. Presented at the International workshop on image processing, real-time edge and motion detection.
- Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is “nearest neighbor” meaningful? *International Conference on Database Theory*, 217–235.
- Beyer, M. (2011). Gartner Says Solving ‘Big Data’ Challenge Involves More Than Just Managing Volumes of Data. *Gartner*. Retrieved from <http://www.gartner.com/it/page.jsp>
- Blaser, A. (1979). *Data base techniques for pictorial applications* (Vol. 81). Florence: Springer.
- Boggs, J. M. (1996). *The art of watching films*. ERIC.
- Bolle, R. M., Yeo, B. L., & Yeung, M. (1998). Video query: Research directions. *IBM Journal of Research and Development*, 42(2), 233–252.
- Boreczky, J. S., & Rowe, L. A. (1996). Comparison of video shot boundary detection techniques. *Journal of Electronic Imaging*, 5(2), 122–128.
- Bowden, R., Gilbert, A., & KaewTraKulPong, P. (2006). Tracking objects across uncalibrated arbitrary topology camera networks. *Intelligent Distributed Video Surveillance Systems*, 157–183.
- Boykov, Y., & Funka-Lea, G. (2006). Graph cuts and efficient ND image segmentation. *International Journal of Computer Vision*, 70(2), 109–131.
- Bozkaya, T., & Ozsoyoglu, M. (1999). Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3), 361–404.
- Bradski, G. (2000). The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*.
- Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O’Reilly Media, Incorporated.
- Bribiesca, E., & Guzman, A. (1980). How to describe pure form and how to measure differences in shapes using shape numbers. *Pattern Recognition*, 12(2), 101–112.
- Brunelli, R., Mich, O., & Modena, C. M. (1999). A Survey on the Automatic Indexing of Video Data. *Journal of visual communication and image representation*, 10(2), 78–112.
- Bruno, N., Chaudhuri, S., & Ramamurthy, R. (2009). Power hints for query optimization (pp. 469–480). Presented at the IEEE 25th International Conference on Data Engineering, IEEE.

- Caloyannides, M. A. (2003). Society cannot function without privacy. *IEEE Security and Privacy*, 1(3), 84–86.
- Camara-Chavez, G., Precioso, F., Cord, M., Phillip-Foliguet, S., & De A Araujo, A. (2007). Shot boundary detection by a hierarchical supervised approach (pp. 197–200). Presented at the Systems, Signals and Image Processing, 2007 and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services. 14th International Workshop on, IEEE.
- Catarci, T., Donderler, M., Saykol, E., Ulusoy, O., & Gudukbay, U. (2003). BilVideo: a video database management system. *Multimedia, IEEE*, 10(1), 66–70.
- Chang, N., & Fu, K. (1980). A relational database system for images. *Pictorial Information Systems*, 288–321.
- Chang, N. S., & Fu, K. S. (1980). Query-by-pictorial-example. *IEEE Transactions on Software Engineering*, (6), 519–524.
- Chang, S. F., Eleftheriadis, A., & McClintock, R. (1998). Next-generation content representation, creation, and searching for new-media applications in education. *Proceedings of the IEEE*, 86(5), 884–904.
- Chang, S. K., & Hsu, A. (1992). Image information systems: where do we go from here? *Knowledge and Data Engineering, IEEE Transactions on*, 4(5), 431–442.
- Chang, S. K., & Kunil, T. (1981). Pictorial data-base systems. *Computer*, 14(11), 13–21.
- Chang, S. K., Yan, C., Dimitroff, D. C., & Arndt, T. (1988). An intelligent image database system. *Software Engineering, IEEE Transactions on*, 14(5), 681–688.
- Chaudhuri, S. (1998). An overview of query optimization in relational systems (pp. 34–43). Presented at the Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM.
- Chazelle, B., & Edelsbrunner, H. (1992). An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM (JACM)*, 39(1), 1–54.
- Chen, J. Y., Bouman, C. A., & Dalton, J. C. (2000). Hierarchical browsing and search of large image databases. *IEEE Transactions on Image Processing*, 9(3), 442–455.
- Chen, X., Zhang, C., Chen, S. C., & Chen, M. (2005). A latent semantic indexing based method for solving multiple instance learning problem in region-based image retrieval (p. 8–pp). Presented at the Seventh IEEE International Symposium on Multimedia, IEEE.
- Cheng, H., Hua, K. A., & Yu, N. (2010). An automatic feature generation approach to multiple instance learning and its applications to image databases. *Multimedia Tools and Applications*, 47(3), 507–524.

- Cheung, S. C. S., & Kamath, C. (2004). Robust techniques for background subtraction in urban traffic video (Vol. 5308, pp. 881–892). Presented at the Proceedings of SPIE.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. MIT press.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273–297.
- Cremers, D., Rousson, M., & Deriche, R. (2007). A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape. *International journal of computer vision*, 72(2), 195–215.
- Cucchiara, R., Grana, C., Piccardi, M., & Prati, A. (2003). Detecting moving objects, ghosts, and shadows in video streams. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10), 1337–1342.
- Danielson, P. (2002). Video surveillance for the rest of us: Proliferation, privacy, and ethics education (pp. 162–167). Presented at the International Symposium on Technology and Society, IEEE.
- Date, C. J. (1977). *An Introduction to Database Systems* (2nd ed.). Addison-Wesley Publishing Company, Inc.
- De La Briandais, R. (1959). File searching using variable length keys (pp. 295–298). Presented at the Papers presented at the the March 3-5, 1959, western joint computer conference, ACM.
- Dietterich, T. G., Lathrop, R. H., & Lozano-Perez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2), 31–71.
- Dimitrova, N., Zhang, H. J., Shahraray, B., Sezan, I., Huang, T., & Zakhor, A. (2002). Applications of video-content analysis and retrieval. *MultiMedia, IEEE*, 9(3), 42–55.
- Dimopoulos, S., & Landsberg, G. (2001). Black holes at the large hadron collider. *Physical Review Letters*, 87(16), 161602.
- Du, W., & Piater, J. (2007). Multi-camera people tracking by collaborative particle filters and principal axis-based integration (pp. 365–374). Presented at the Proceedings of the 8th Asian conference on Computer vision-Volume Part I, Springer-Verlag.
- Duda, R. O., Hart, P. E., & Stork, D. G. (1995). *Pattern Classification and Scene Analysis* 2nd ed.
- Dufaux, F., & Ebrahimi, T. (2008). Scrambling for privacy protection in video surveillance systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8), 1168–1174.

- Durkee, D. (2010). Why cloud computing will never be free. *Queue*, 8(4), 20.
- Dwork, C. (2008). Differential privacy: A survey of results. *Theory and Applications of Models of Computation*, 1–19.
- Eastman, C., & Zemankova, M. (1982). Partially specified nearest neighbor searches using kd trees. *Information Processing Letters*, 15(2), 53–56.
- Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 167–181.
- Feng, H., Fang, W., Liu, S., & Fang, Y. (2005). A new general framework for shot boundary detection and key-frame extraction (pp. 121–126). Presented at the Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval, ACM.
- Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1), 1–9.
- Finkelstein, S. (1982). Common expression analysis in database applications (pp. 235–245). Presented at the Proceedings of the 1982 ACM SIGMOD international conference on Management of data, ACM.
- Fisher, C. W., Lauria, E. J. M., & Matheus, C. C. (2009). An accuracy metric: Percentages, randomness, and probabilities. *Journal of Data and Information Quality (JDIQ)*, 1(3), 16.
- Fisher, R. (2011, November 12). CAVIAR: Context Aware Vision using Image-based Active Recognition. Retrieved November 12, 2011, from <http://homepages.inf.ed.ac.uk/rbf/CAVIAR/>
- Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., ... Petkovic, D. (1995). Query by image and video content: The QBIC system. *Computer*, 28(9), 23–32.
- Fodor, I. K. (2002). A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 9, 1–18.
- Foote, J. (1997). Content-based retrieval of music and audio (Vol. 3229, pp. 138–147). Presented at the Proc. SPIE.
- Foote, J. (1999). An overview of audio information retrieval. *Multimedia Systems*, 7(1), 2–10.
- Ford, R. M., Robson, C., Temple, D., & Gerlach, M. (2000). Metrics for shot boundary detection in digital video sequences. *Multimedia Systems*, 8(1), 37–46.
- Fu, K. S., & Mui, J. (1981). A survey on image segmentation. *Pattern recognition*, 13(1), 3–16.
- Fukunaga, K., & Narendra, P. M. (1975). A branch and bound algorithm for computing k-nearest neighbors. *Computers, IEEE Transactions on*, 100(7), 750–753.

- Fung, B., Wang, K., Chen, R., & Yu, P. S. (2010). Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4), 14.
- Gartner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels (pp. 179–186). Presented at the Proceedings of the 19th International Conference on Machine Learning.
- Geetha, P., & Narayanan, V. (2008). A Survey of Content-Based Video Retrieval. *Journal of Computer Science*, 4(6), 474–486.
- Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing (pp. 518–529). Presented at the Proceedings of the International Conference on Very Large Data Bases.
- Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4), 132–133.
- Grant, J., & Minker, J. (1981). Optimization in deductive and conventional relational database systems. *Advances in Data Base Theory*, 1, 195–234.
- Grimes, J., & Potel, M. (1991). What is multimedia? *Computer Graphics and Applications, IEEE*, 11(1), 49–52.
- Guting, R. H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., & Vazirgiannis, M. (2000). A foundation for representing and querying moving objects. *ACM Transactions on Database Systems (TODS)*, 25(1), 1–42.
- Guttman, A. (1984). *R-trees: a dynamic index structure for spatial searching* (Vol. 14). ACM.
- Hall, P. A. V. (1974). *Common Subexpression Identification in General: Algebraic Systems*. UK Scientific Centre, IBM United Kingdom Limited.
- Hampapur, A., Brown, L., Connell, J., Ekin, A., Haas, N., Lu, M., ... Pankanti, S. (2005). Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. *Signal Processing Magazine, IEEE*, 22(2), 38–51.
- Harris, C., & Stephens, M. (1988). A combined corner and edge detector (Vol. 15, p. 50). Presented at the Alvey vision conference, Manchester, UK.
- Hatano, H. (1996). Image processing and database system in the National Museum of Western Art; an integrated system for art research. *INSPEL*, 30, 259–267.
- Hemayed, E. E. (2003). A survey of camera self-calibration (pp. 351–357). Presented at the IEEE Conference on Advanced Video and Signal Based Surveillance, IEEE.
- Hinneburg, A., Aggarwal, C. C., & Keim, D. A. (2000). *What is the nearest neighbor in high dimensional spaces?* Morgan Kaufmann Publishers Inc.

- Hoberman, S. (2005). Data modeling made simple: A practical guide for business & information technology professionals.
- Hristescu, G., & Farach-Colton, M. (1999). *Cluster-preserving embedding of proteins*. Technical Report 99-50, Computer Science Department, Rutgers University.
- Hu, M. K. (1962). Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2), 179–187.
- Huang, T., Mehrotra, S., & Ramchandran, K. (1997). Multimedia analysis and retrieval system (MARS) project.
- Huttenlocher, D. P., Klanderman, G. A., & Rucklidge, W. J. (1993). Comparing images using the Hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9), 850–863.
- Ide, I., Yamamoto, K., & Tanaka, H. (1999). Automatic video indexing based on shot classification. *Advanced Multimedia Content Processing*, 87–102.
- Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality (pp. 604–613). Presented at the Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM.
- Jackson, J. E. (1991). *A user's guide to principal components* (Vol. 244). Wiley-Interscience.
- Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8), 651–666.
- Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.
- Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1), 4–37.
- Jain, R., & Hampapur, A. (1994). Metadata in video databases. *ACM Sigmod Record*, 23(4), 27–33.
- Javed, O., Rasheed, Z., Shafique, K., & Shah, M. (2003). Tracking across multiple cameras with disjoint views.
- Javed, O., & Shah, M. (2008). *Automated multi-camera surveillance: algorithms and practice* (Vol. 10). Springer.
- Jerri, A. J. (1977). The Shannon sampling theorem—Its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11), 1565–1596.



- Jiang, H., Montesi, D., & Elmagarmid, A. K. (1997). VideoText database systems (pp. 344–351). Presented at the IEEE International Conference on Multimedia Computing and Systems, IEEE.
- Jolliffe, I. (2005). *Principal component analysis*. Wiley Online Library.
- Jungnickel, D. (2004). *Graphs, networks and algorithms* (Vol. 5). Springer.
- Kamgar-Parsi, B., & Kanal, L. N. (1985). An improved branch and bound algorithm for computing k-nearest neighbors. *Pattern recognition letters*, 3(1), 7–12.
- Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7), 881–892.
- Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. *International journal of computer vision*, 1(4), 321–331.
- Kim, K., Chalidabhongse, T. H., Harwood, D., & Davis, L. (2005). Real-time foreground–background segmentation using codebook model. *Real-time imaging*, 11(3), 172–185.
- Kim, W. (1984). Global optimization of relational queries: A first step. *Query processing in database systems*, 206–216.
- Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2), 83–97.
- Lau, T., & King, I. (1997). Montage: An image database for the fashion, textile, and clothing industry in Hong Kong. *Computer Vision—ACCV’98*, 410–417.
- Lee, M. S., Yang, Y. M., & Lee, S. W. (2001). Automatic video parsing using shot boundary detection and camera operation analysis. *Pattern Recognition*, 34(3), 711–719.
- Lew, M. S., Sebe, N., Djeraba, C., & Jain, R. (2006). Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 2(1), 1–19.
- Lienhart, R. (1999). Comparison of automatic shot boundary detection algorithms (Vol. 3656, pp. 290–301). Presented at the Proc. SPIE.
- Linial, N., London, E., & Rabinovich, Y. (1995). The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2), 215–245.
- Liu, X., & Chen, T. (2002). Shot boundary detection using temporal statistics modeling (Vol. 4, pp. IV–3389). Presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE.

- Liu, Y., Zhang, D., Lu, G., & Ma, W. Y. (2007). A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40(1), 262–282.
- Liu, Z., & Huang, Q. (2000). Content-based indexing and retrieval-by-example in audio (Vol. 2, pp. 877–880). Presented at the IEEE International Conference on Multimedia and Expo, IEEE.
- Lo, B., & Velastin, S. (2001). Automatic congestion detection system for underground platforms (pp. 158–161). Presented at the International Symposium on Intelligent Multimedia, Video and Speech Processing, IEEE.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features (Vol. 2, pp. 1150–1157). Presented at the Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, IEEE.
- Makhoul, J., Kubala, F., Leek, T., Liu, D., Nguyen, L., Schwartz, R., & Srivastava, A. (2000). Speech and language technologies for audio indexing and retrieval. *Proceedings of the IEEE*, 88(8), 1338–1353.
- Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N., & Theodoridis, Y. (2005). *R-trees: Theory and Applications*. Springer.
- Mardia, K. V., Kent, J. T., & Bibby, J. M. (1980). Multivariate analysis.
- Maron, O., & Lozano-Perez, T. (1998). A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*.
- Maron, O., & Ratan, A. L. (1998). Multiple-instance learning for natural scene classification (Vol. 15). Presented at the Proceedings of the Fifteenth International Conference on Machine Learning.
- Mehrotra, S., Rui, Y., Ortega-Binderberger, M., & Huang, T. S. (1997). Supporting content-based queries over images in MARS (pp. 632–633). Presented at the IEEE International Conference on Multimedia Computing and Systems, IEEE.
- Mills-Tettey, G. A., Stentz, A., & Dias, M. B. (2007). *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs* (No. CMU-RI-TR-07-27). Pittsburgh, PA: Robotics Institute.
- Mojsilovic, A., & Rogowitz, B. (2001). Capturing image semantics with low-level descriptors (Vol. 1, pp. 18–21). Presented at the Image Processing, 2001. Proceedings. 2001 International Conference on, IEEE.
- Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1), 32–38.

- Nagasaka, A., & Tanaka, Y. (1992). Automatic video indexing and full-video search for object appearances.
- Nievergelt, J., Hinterberger, H., & Sevcik, K. C. (1984). The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1), 38–71.
- Nixon, M., & Aguado, A. S. (2012). *Feature Extraction & Image Processing for Computer Vision*. Academic Press.
- Norusis, M. J. (1990). *SPSS advanced statistics user's guide*. SPSS Chicago.
- Nwosu, K. C., Thuraisingham, B. M., & Berra, P. B. (1996). *Multimedia Database Systems: design and implementation strategies*. Springer.
- Oh, J. H., Hua, K. A., & Liang, N. (2000). A content-based scene change detection and classification technique using background tracking (pp. 254–265). Presented at the SPIE Conf. on Multimedia Computing and Networking.
- Pal, N. R., & Pal, S. K. (1993). A review on image segmentation techniques. *Pattern recognition*, 26(9), 1277–1294.
- Parks, D. H., & Fels, S. S. (2008). Evaluation of background subtraction algorithms with post-processing (pp. 192–199). Presented at the IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance, IEEE.
- Peng, B., Zhang, L., & Zhang, D. (2013). A Survey of Graph Theoretical Approaches to Image Segmentation. *Pattern Recognition*, 46(3), 1020–1038. doi:10.1016/j.patcog.2012.09.015
- Peng, R., Aved, A. J., & Hua, K. A. (2010). Real-Time Query Processing on Live Videos in Networks of Distributed Cameras. *International Journal of Interdisciplinary Telecommunications and Networking (IJITN)*, 2(1), 27–48.
- Pentland, A., Picard, R. W., & Sclaroff, S. (1996). Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3), 233–254.
- Piccardi, M. (2004). Background subtraction techniques: a review (Vol. 4, pp. 3099–3104). Presented at the Systems, Man and Cybernetics, 2004 IEEE International Conference on, IEEE.
- Pieprzyk, J., & Sadeghiyan, B. (2001). *Design of hashing algorithms*. Springer-Verlag New York, Inc.
- Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., & Seltzer, M. (2006). Network-aware operator placement for stream-processing systems (p. 49). Presented at the 22nd International Conference on Data Engineering, IEEE.

- Pirsiavash, H., Ramanan, D., & Fowlkes, C. C. (2011). Globally-optimal greedy algorithms for tracking a variable number of objects (pp. 1201–1208). Presented at the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE.
- Radke, R. J. (2010). A survey of distributed computer vision algorithms. *Handbook of Ambient Intelligence and Smart Environments*, 35–55.
- Ray, S., & Craven, M. (2005). Supervised versus multiple instance learning: An empirical comparison (pp. 697–704). Presented at the Proceedings of the 22nd international conference on Machine learning, ACM.
- Rekleitis, I., & Dudek, G. (2005). Automated calibration of a camera sensor network (pp. 3384–3389). Presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE.
- Rekleitis, I., Meger, D., & Dudek, G. (2006). Simultaneous planning, localization, and mapping in a camera sensor network. *Robotics and Autonomous Systems*, 54(11), 921–932.
- Rish, I. (2001). An empirical study of the naive Bayes classifier (Vol. 3, pp. 41–46). Presented at the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence.
- Rosin, P. L. (1997). Edges: saliency measures and automatic thresholding. *Machine Vision and Applications*, 9(4), 139–159.
- Rui, Y., Huang, T. S., & Chang, S. F. (1999). Image retrieval: Current techniques, promising directions, and open issues. *Journal of visual communication and image representation*, 10(1), 39–62.
- Rui, Y., Huang, T. S., & Mehrotra, S. (1997). Content-based image retrieval with relevance feedback in MARS (Vol. 2, pp. 815–818). Presented at the International Conference on Image Processing, IEEE.
- Saini, M., Atrey, P. K., Mehrotra, S., Emmanuel, S., & Kankanhalli, M. (2010). Privacy modeling for video data publication (pp. 60–65). Presented at the IEEE International Conference on Multimedia and Expo, IEEE.
- Samet, H. (1990). *The design and analysis of spatial data structures* (Vol. 85). Addison-Wesley Reading MA.
- Samet, H. (1995). Spatial data structures. *Modern Database Systems, The Object Model, Interoperability and Beyond*, 361–385.
- Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

- Sato, T., Kanade, T., Hughes, E. K., Smith, M. A., & Satoh, S. (1999). Video OCR: indexing digital news libraries by recognition of superimposed captions. *Multimedia Systems*, 7(5), 385–395.
- Scheuermann, P., & Ouksel, M. (1982). Multidimensional B-trees for associative searching in database systems. *Information systems*, 7(2), 123–137.
- Schneiderman, H., & Kanade, T. (1998). Probabilistic modeling of local appearance and spatial relationships for object recognition (pp. 45–51). Presented at the Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on, IEEE.
- Sellis, T. K. (1988). Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1), 23–52.
- Senior, A., Pankanti, S., Hampapur, A., Brown, L., Tian, Y. L., Ekin, A., ... Lu, M. (2005). Enabling video privacy through computer vision. *Security & Privacy, IEEE*, 3(3), 50–57.
- Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*.
- Siddiqi, K., & Kimia, B. B. (1995). Parts of visual form: Computational aspects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(3), 239–251.
- Skopal, T. (2004). Pivoting M-tree: A metric access method for efficient similarity search (pp. 27–37). Presented at the Proceedings of the Annual International Workshop on Databases, Texts, Specifications and Objects, Citeseer.
- Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., & Jain, R. (2000). Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12), 1349–1380.
- Smith, R., Self, M., & Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. *Autonomous robot vehicles*, 1, 167–193.
- Snoek, C. G. M., & Smeulders, A. W. M. (2010). Visual-Concept Search Solved? *Computer*, 76–78.
- Snoek, C. G. M., & Worring, M. (2005). Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools and Applications*, 25(1), 5–35.
- Snoek, C. G. M., Worring, M., Van Gemert, J. C., Geusebroek, J. M., & Smeulders, A. W. M. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia (pp. 421–430). Presented at the Proceedings of the 14th annual ACM international conference on Multimedia, ACM.
- Song, B., & Roy-Chowdhury, A. K. (2007). Stochastic adaptive tracking in a camera network.

- Stauffer, C., & Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking (Vol. 2). Presented at the Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., IEEE.
- Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer-Verlag New York, Inc.
- Tagare, H., Vos, F. M., Jaffe, C. C., & Duncan, J. S. (1995). Arrangement: A spatial relation between parts for evaluating similarity of tomographic section. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(9), 880–893.
- Tantaoui, M. A., Hua, K. A., & Do, T. T. (2004). BroadCatch: a periodic broadcast technique for heterogeneous video-on-demand. *Broadcasting, IEEE Transactions on*, 50(3), 289–301.
- Taylor, S. (2007). *Optimizing Applications for Multi-Core Processors, Using the Intel Integrated Performance Primitives*. Intel Press.
- Tieu, K., Dalley, G., & Grimson, W. E. L. (2005). Inference of non-overlapping camera network topology by measuring statistical dependence.
- Treisman, A., Cavanagh, P., Fischer, B., Ramachandran, V. S., & Von der Heydt, R. (1990). Form perception and attention: Striate cortex and beyond.
- Turner, J. (1990). Representing and accessing information in the stockshot database at the National Film Board of Canada. *Canadian Journal of Information Science*, 15(4), 1–22.
- Uhlmann, J. K. (1991). Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4), 175–179.
- Ullman, J. D. (1981). Principles of database systems. *Computer Software Engineering Series, Rockville: Computer Science Press, 1981, 1*.
- Velipasalar, S., Brown, L. M., & Hampapur, A. (2010). Detection of user-defined, semantically high-level, composite events, and retrieval of event queries. *Multimedia Tools and Applications*, 50(1), 249–278.
- Wang, J. T. L., Wang, X., Lin, K. I., Shasha, D., Shapiro, B. A., & Zhang, K. (1999). Evaluating a class of distance-mapping algorithms for data mining and clustering (pp. 307–311). Presented at the Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.
- Wang, Y., Liu, Z., & Huang, J. C. (2000). Multimedia content analysis-using both audio and visual clues. *Signal Processing Magazine, IEEE*, 17(6), 12–36.
- Weber, R., Schek, H. J., & Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces (pp. 194–205). Presented at the Proceedings of the International Conference on Very Large Data Bases, IEEE.

- Weinberger, K. Q., Blitzer, J., & Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. Presented at the In NIPS, Citeseer.
- White, T. (2012). *Hadoop: The definitive guide*. O'Reilly Media.
- Wold, E., Blum, T., Keislar, D., & Wheaton, J. (1996). Content-based classification, search, and retrieval of audio. *MultiMedia, IEEE*, 3(3), 27–36.
- Wren, C. R., Azarbayejani, A., Darrell, T., & Pentland, A. P. (1997). Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7), 780–785.
- Xu, C., Yezzi Jr, A., & Prince, J. L. (2000). On the relationship between parametric and geometric active contours (Vol. 1, pp. 483–489). Presented at the Conference Record of the Thirty-Fourth Asilomar Signals, Systems and Computers, IEEE.
- Yang, J., & Hauptmann, A. G. (2008). (Un) Reliability of video concept detection (pp. 85–94). Presented at the Proceedings of the 2008 international conference on Content-based image and video retrieval, ACM.
- Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces (pp. 311–321). Presented at the Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, Society for Industrial and Applied Mathematics.
- Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 38(4), 13.
- Zha, H., He, X., Ding, C., Simon, H., & Gu, M. (2001). Bipartite graph partitioning and data clustering (pp. 25–32). Presented at the Proceedings of the tenth international conference on Information and knowledge management, ACM.
- Zhang, D., Qi, W., & Zhang, H. (2001). A new shot boundary detection algorithm. *Advances in Multimedia Information Processing—PCM 2001*, 63–70.
- Zhang, H. J., Kankanhalli, A., & Smoliar, S. W. (1993). Automatic partitioning of full-motion video. *Multimedia systems*, 1(1), 10–28.
- Zhang, Z., & Zhang, R. (2008). *Multimedia data mining: a systematic introduction to concepts and theory*. Chapman & Hall/CRC.
- Zhao, R., & Grosky, W. I. (2002). Negotiating the semantic gap: from feature maps to semantic landscapes. *Pattern Recognition*, 35(3), 593–600.
- Zheng, W., Yuan, J., Wang, H., Lin, F., & Zhang, B. (2005). A novel shot boundary detection framework (Vol. 5960, pp. 596018–1). Presented at the Proc. of SPIE Vol.

- Zhou, J., & Zhang, X. P. (2005). Video shot boundary detection using independent component analysis (pp. 541–544). Presented at the Proc. Int. Conf. Acoustics, Speech and Signal Processing.
- Zhou, X. S., & Huang, T. S. (2000). CBIR: from low-level features to high-level semantics (Vol. 3974, p. 426). Presented at the Proceedings of SPIE.
- Zhou, Y. H., Cao, Y. D., Zhang, L. F., & Zhang, H. X. (2005). An SVM-based soccer video shot classification (Vol. 9, pp. 5398–5403). Presented at the Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on, IEEE.
- Zou, X., Bhanu, B., Song, B., & Roy-Chowdhury, A. K. (2007). Determining topology in a distributed camera network (Vol. 5, pp. V–133). Presented at the IEEE International Conference on Image Processing, IEEE.