An Introduction to Multimedia Data Indexing and Retrieval

Michael Kropfberger
michael.kropfberger@gmx.net

28th November 1999

Contents

Abstract In this paper, I will describe todays general problems with video indexing. Furthermore, two possible approaches, non-semantic vs. semantic indexing, are introduced. We will show different implementations and thoughts about indexing and retrieval. Finally, we will show a more or less simple way to efficiently include self-explaining indexing data directly into the video stream.

Key words video indexing and retrieval - non-semantic indexing - semantic indexing - QBIC - AVIS - OVID - Java - MPEG - query processing - multimedia data

1 Introduction

Since the time, when computers were powerful enough to handle video and audio streams (*aka multimedia streams*) in a reasonable quality, many efforts were taken to invent application areas in this field. This does not include only local editing of multimedia data, but also – as network connections became faster and faster – distributed video conferencing (with enhanced blackboard facilities), distant learning, video on demand (*VoD*), or large video/image libraries. All over the world, these huge amounts of data are stored in different binary raw formats like MPEG-1[1], which is a standardized video and audio compression format for approximately video quality.

Even very specialized video[1] libraries like medical operations databases or the “Alfred Hitchcock Fan-club Video Library” [3] may consist of many hard disk and CD-ROM filling videos and it is hard to find a specific video at a glance. It is easy to maintain a list of video titles corresponding to a CD-ROM/file identifier, but there are many ad-hoc queries which may be of interest to the video database user.

The most wanted approach to video indexing would be a completly automated procedure without human involvement. We are looking for a computer program, which is intelligent enough to gaze through a video, and understands the video's contents and events. Let's take a movie. The program should recognize that two humans are walking along the beach, there are sailing boats in the background. It should also “listen”[2] to the conversation and understand that one human is called Tom, so apparently is male and the other human is called Jackie. Jackie could be Jacob or Jacqueline, so our intelligent program should check the high pitch of the yet asexual human's voice. Those detailed specifics should be automatically generalized because we also want to ask queries like:

1. Find all video frames where Jackie meets another human being.
2. Find all video frames where boats occur.
3. Find all video frames where the location is a beach, and two people are strolling from the left to the right.
4. Find all video frames where a dog jumps up and down.[3]

[1]For simplicity, the reader may substitute future occurrences of the term *video* with *video/audio* throughout this document. To be more general, *video* could be every temporal multimedia data stream, which can be indexed.

[2]This forces the inclusion of the audio tracks.

[3]Was there a dog in the above mentioned movie scene? :)

Of course, this has to be possible for multiple videos.

Unfortunately, this intelligent program does not exist yet. Perceptual organization (the process of grouping image features into meaningful objects, and attaching semantic descriptions to scenes through model matching[2]) by computers is a field of artificial intelligence. This problem is far from being solved.

Now let us look on possible problems for queries concerning a single video: Although it is pretty easy to look for a certain string pattern in a text file, there is absolutely no quickly accessible[4] information stored in video streams except some meta data like video size, audio quality, length, or creation time. Accepting this handicap, let's multiply the above mentioned problems for searching in multiple videos stored in a huge database.

There are two big approaches to video indexing, which both have their advantages and disadvantages:

- non-semantic indexing
- semantic indexing

In the following two sections we will outline the different methods of video indexing in more detail.

2 Non-Semantic Indexing

Imagine a fashion designer needing a cloth with a special mixture of colors or pattern, or a museum cataloger looking for different vase-shaped objects in a catalog. All this information can be calculated/extracted out of a video/image database without the need to know what is really shown in it.

2.1 Query by Image Content (QBIC)[2]

A project called QBIC was introduced by the IBM Almaden Research Center[5]. It implements the following content-based retrieval methods on large image and video databases:

- example images,
- user-constructed sketches and drawings,
- selected color and texture patterns,
- camera and object motion, and
- other graphical information.

So when you are looking for a vase-shaped object based on a sketch, you may find vases, fish, and pictures of snowmen in your query results. Since the computer does not know which semantics are associated with a vase, it is your task to refine your query by adding eg. a clayish gray-brown pattern or a handle to your sketch.

[4]Even color information has to be calculated as color histograms (see Section 2) for each picture.

[5]To run an interactive query, visit the QBIC Web server at http://wwwqbic.almaden.ibm.com

QBIC tries to add a simple semantic indexing method by giving the user the possibility to describe a picture with a line of text. To make the queries more powerful, these descriptions may not only be associated with the whole picture but with objects in that picture. QBIC uses an object identification algorithm based on a foreground/background model. It works reasonably good only on museum and retail catalogs, which have a small number of foreground objects on a generally separable background. This works without any user intervention, but there is also the possibility to help the computer by setting the foreground/background threshold, helping with outlining tools like self-fitting curves, or by clicking on non-connected object-parts to join two objects into one. An example for connection would be a swarm of birds. Since this swarm of birds is based on many different birds, every bird additionally could be a single object in this scene.

Video data needs more features like

- shot detection,
- representative frame creation for each shot, and
- derivation of a layered representation of coherently moving structures/objects.

A shot is a sequence of frames, which is taken with one camera from one position. Multiple coherent shots are called a scene[6] (eg. an interview, where a reporter and a politician is shown alternatingly).

Shot detection means the connection of frames belonging together and is basically done by detecting scene changes or cuts. Problems arise with softly changing effects like fades or dissolves. The frames of one shot have many things in common, such as showing the same background and containing the same (maybe moving) objects.

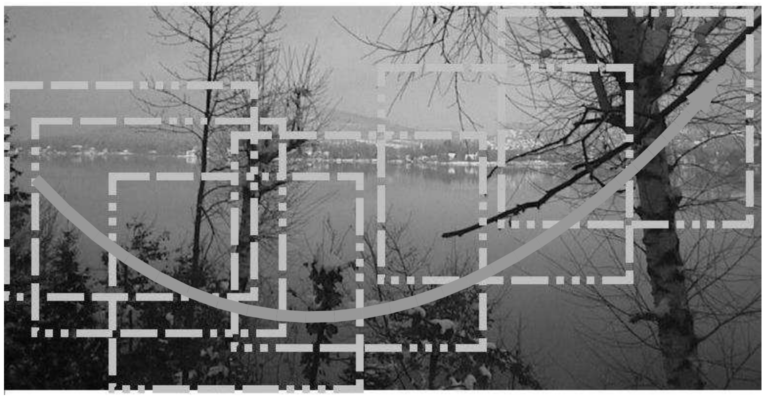


Figure 1: Stitched image from multiple images

This shot detection allows newly generated still images, eg. panning movie-scenes may be stitched together, so we have a very wide image. Assuming an image size of $w \times h$ and a panning (eg. camera moves from the top left to bottom right) over a shot of N frames. After overlaying the frames, every frame i (with $1 \leq i \leq N$) has an upper-left corner point $P_i(X_i, Y_i)$. So the stitched image size is

$$Size_{all} = \Big(|\max(X_i) - \min(X_i)| + w\Big) \times \Big(|\max(Y_i) - \min(Y_i)| + h\Big)$$

Additionally to these thoughts, we have to add spherical effects because of the fish-eye effect of camera lenses.

Another example is to remove a moving object out of a shot by using temporal median filtering on the frames of the shot. Also different layers of a panning scene could be extracted: eg. a shot is taken out of a moving car. There is a house in the background with some trees closer to the camera. Apparently the trees will move faster and a wide image of the house in the back could be generated, removing all trees in front.

3 Semantic Indexing

Think about the movie industry. There are thousands of movies available (maybe not all digitally, I guess). In the future it will be important to rent them over new media channels like Internet-2 (a new high-bandwidth world-spanning IP network).

Now think about news stations gathering hundreds of video/audio/text sniplets[6] over the years. Reporters need them to find historic events similar or dependent to today´s news.

To gather this kind of information, you will need more semantic knowledge about the video data. You are not looking for something, which looks like a black hat and has a circle under it, if you are looking for Charlie Chaplin. So, we need to store names to objects in a movie. This will allow to search for names, but not general things like "give me all frames where Charlie Chaplin dances" or "give me all frames where somebody is dancing".

3.1 Advanced Video Information System (AVIS)[3]

A simple but fast approach is to map a video upon several entities. We need *objects* like "Charlie Chaplin", a car, or a tree, but also *activities* like fishing, dancing etc. Then we need *events*, which are combinations of objects and activities like "Charlie Chaplin dances around a tree", and "The Marshall shoots Billy the Kid". Events being of the same activity types, are differentiated from each other by the set of objects involved in them. To realize a query like "find all frames where a Marshall shoots a murderer", we have to add *roles*, which describe the different objects in an event. Thus, we do not only have the role of the murderer Billy the Kid, but also a role of the murder weapon, eg. a colt. So our *objectArray* does not really distinguish between living persons and normal objects.

The Example Movie In our movie ("The Killing Game"), Billy the Kid stands at the bar and starts arguing with the bartender without any reason. The Marshall Mr. DoJ sits at a table in the back and watches the scene all the time. Then he stands up and helps the poor bartender, Billy gets angry, draws his gun, the Marshall does so either, and – since we have a happy end – the Marshall shoots evil Billy. The Marshall blows the smoke of his gun and leaves the saloon, jumps on his horse and rides out of town to look for other evil desperadoes.

Figure 2 shows the appearance of the different objects and events throughout the movie. For internal storage, a *segment tree* is generated. The numbers in brackets symbolize the shot interval (eg. $[1, 11)$ means from shot 1 to 10, excluding 11). Figure 3 shows the storage arrays for the objects with a linked list to the nodes of the segment tree. So $o5$, the Marshall's colt links to node 2 and 4. It is seen during the shots $[3, 6)$ and $[6, 11)$, so the overall interval melts down to $[3, 11)$, which fits the association map. Events are also stored with links to nodes,

[6]No! Sniplets are not a newly invented $Java^{TM}$ extension!

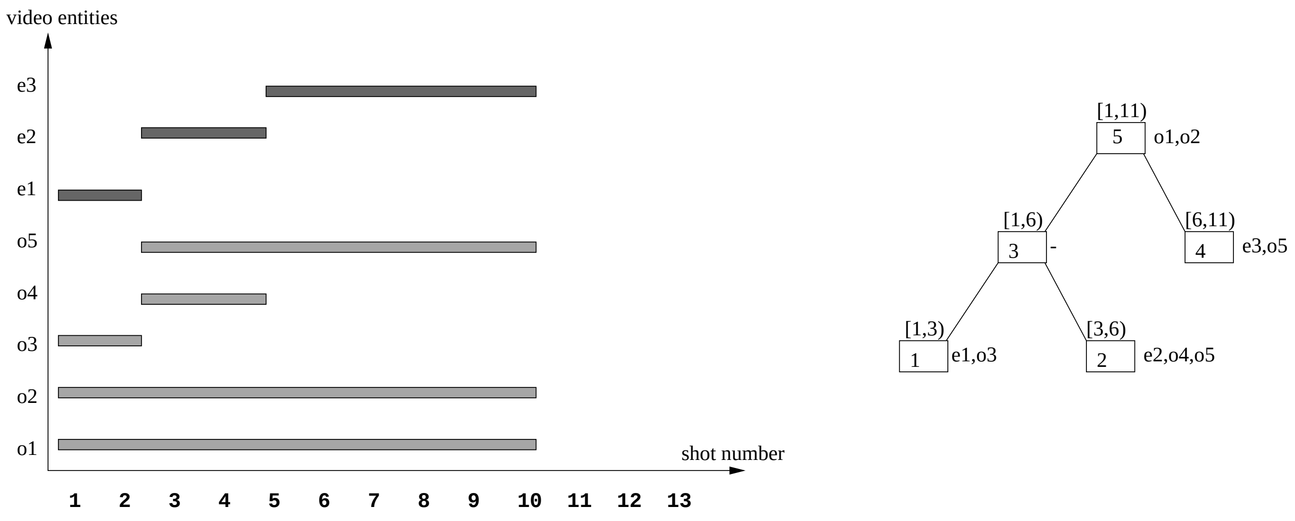
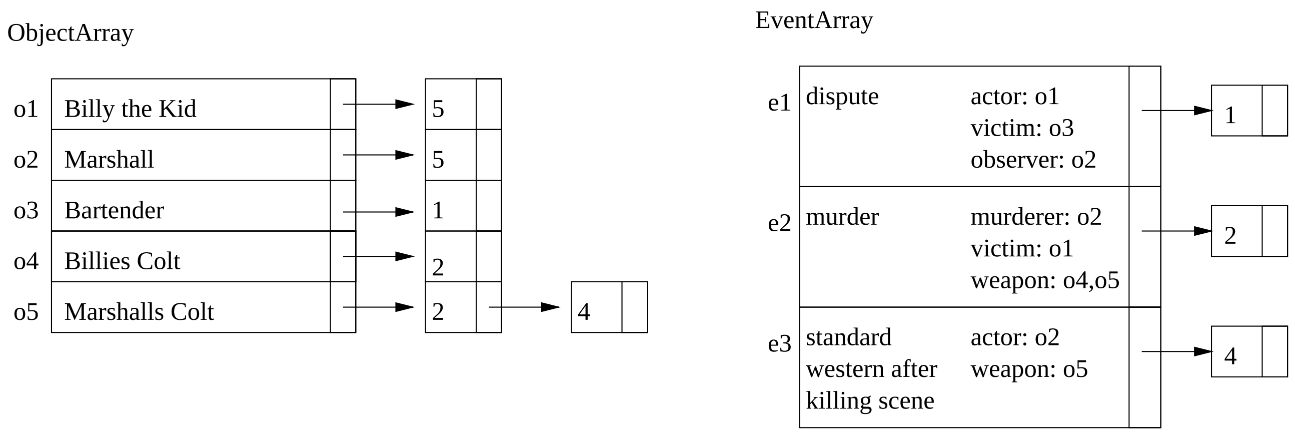


Figure 2: Our short western: association map and object-array-linked segment tree



ObjectArray
o1 Billy the Kid 5
o2 Marshall 5
o3 Bartender 1
o4 Billies Colt 2
o5 Marshalls Colt 2 4
EventArray
e1 dispute actor: o1 victim: o3 observer: o2 1
e2 murder murderer: o2 victim: o1 weapon: o4,o5 2
e3 standard western after killing scene actor: o2 weapon: o5 4

Figure 3: Objects, events (with roles) pointing to tree nodes

and they store role names to certain objects. In [3] various queries/operations based on this data structures are introduced and it is shown, that, in most cases, the complexity of these algorithms is linear.

This “flat” approach – thus very efficient – lacks support for more generalized queries like “find all frames where a male person is threatened by a gun”.

3.2 Object-Oriented Video Information Database (OVID)[5]

To do this, we need a hierarchically organized tree, where we can categorize objects like “Billy The Kid” as persons and actions like “shooting” as cruel activities. To be more generic, we build up a hierarchy tree (like in figure 4), where every child node in the generalization hierarchy tree defines an *is-a* relationship ($\succeq$) to its parent node. So $BillytheKid \succeq cowboy \succeq person$.

For this subsection, a video-object is a triple (oid, I, v) where oid is the unique object ID, I is a set of shot regions, and v is a n-tuple of attribute/value pairs $[a_1 : v1, \ldots, a_n : v_n]$, where each value v_i $(1 \leq i \leq n)$ may be another oid, another interval I', any arbitrary node out of the generalization hierarchy tree, or a set of other values v_j $(1 \leq j \leq m)$.

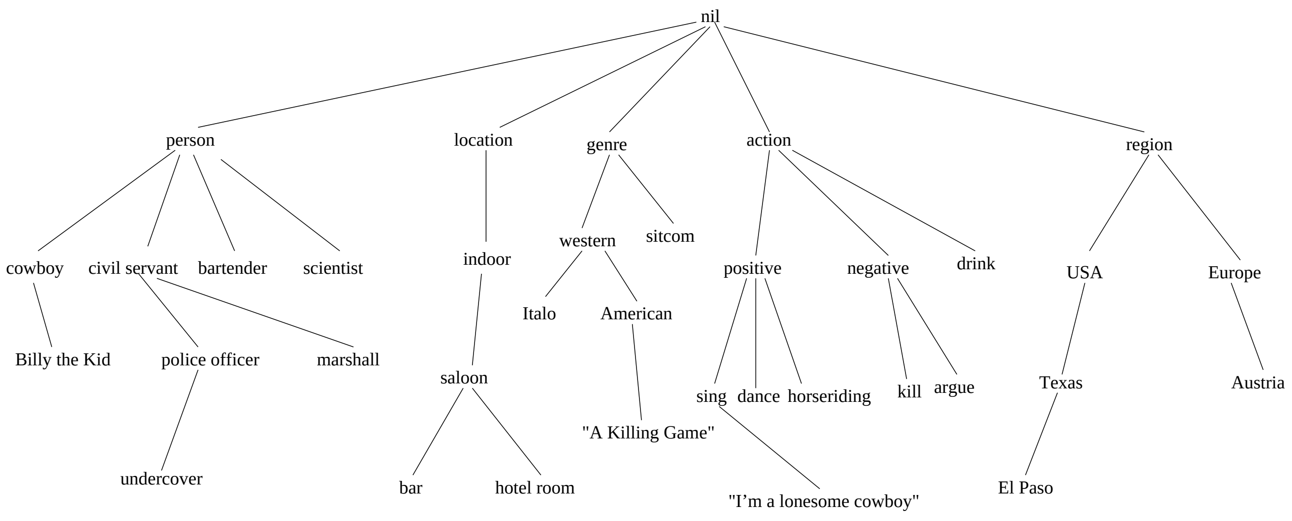


Figure 4: generalization hierarchy tree of atomic values

Inheritance In figure 5, we see *inheritance* as a very important feature of this approach. When an object covers a subregion of another object, some of the attributes should be inherited. The inheritable attributes should be defined in advance by the database creators. In our example, o_1 defines the whole movie, where $o_2..o_n$ are objects covering subregions of this movie. It makes sense to inherit the attribute *title* to all subsequent objects, but there is no need for a reference to all *blood_scenes* from each object.

Based on this database, we can easily ask queries like

1. Give me all frames where a person is in Texas and sings (ignoring attribute names).

2. Give me all frames with *blood_scenes* from various genres.

The more formal queries would be

1. Find all *oid*s where $\exists\ v_i \succeq person$ and $\exists\ v_j \succeq Texas$ and $\exists\ v_k \succeq sing$; return all frame intervals.

2. Find all *oid*s where $\exists\ v_i \succeq genre$ and $\exists\ a_j = blood\_scenes$; return all frame intervals of all objects stored in a_j for each *oid*.

Query (1) will find our $o5$ since

$$marshall \succeq civilservant \succeq person$$

and

$$\text{``}I'm\ a\ lonesome\ cowboy'' \succeq sing$$

Query (2) will match with

$$''AKillingGame'' \succeq \ldots \succeq genre$$

and all frames of $o4$ will be returned.

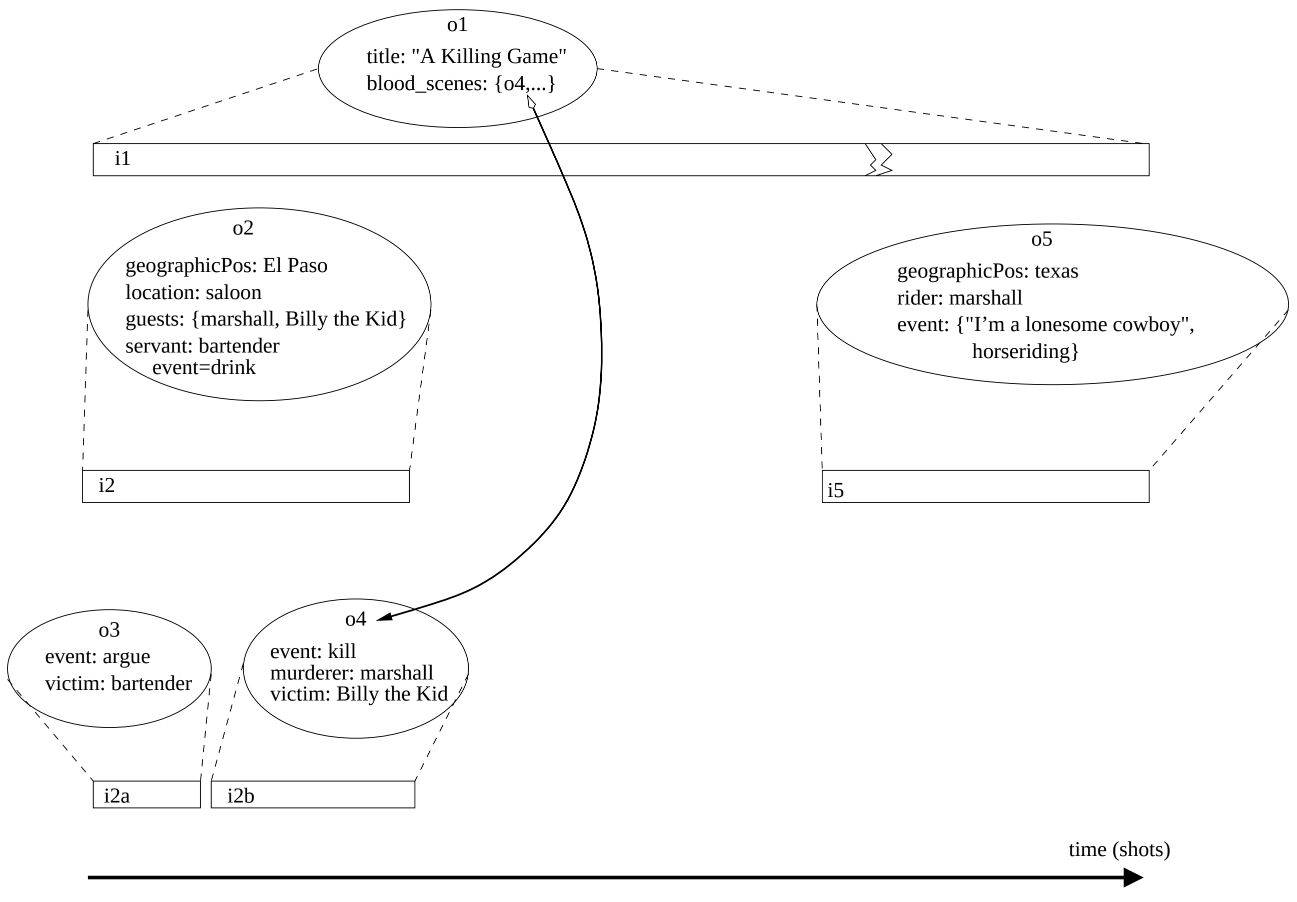


Figure 5: Video Objects in our short western

Composition Operations When a new object o_i is created, it may concern a sub-interval of another object o_j $(i \neq j)$. So every inheritable attribute of o' will be inherited to o. Furthermore, there are two composition operations for video-objects introduced: *merge* and *overlap*.

Generally speaking, *merge* creates a new object o from two existing objects o_i and o_j $(i \neq j)$ such that some descriptional data common to both of o_i and o_j is inherited by o, and that o's interval set is the union (merge) of the interval sets of o_i and o_j[5]. When o_i and o_j have an equal attribute name a_m with different values $(o_i.a_m \neq o_j.a_m)$, the first shared general value found in the hierarchical tree is taken for $o.a_m$. If the two a_m have different video-objects, these are merged together recursively. In our example, if we merge $o2$ and $o5$ to a new object $o6$, we have two equal attributes *geographicPos* and *event* which both hold values. Because of the given hierarchy, the attribute *geographicPos* is set to *Texas* and the attribute *event* is set to *action*. When the equal attribute name stores other objects, then all objects go into the merged attribute.

Any newly generated object using the *overlap* operator is the result of taking the intersection of two video-objects o_i and o_j $(i \neq j)$. So only really overlapping intervals out of both interval sets go into the new object. Using our example, we may have an object $o6$ being already merged from $o2$ and $o5$ with the resulting attribute vector $\{geographicPos = Texas, event = action\}$. Now we want to overlap $o6$ and $o4$. The resulting interval set is $\{i2b\}$ and the attribute vector is $\{geographicPos = Texas, event = \{action, kill\}, murderer = marshall, victim = Billy\ the\ Kid\}$.

4 Intelligent Multimedia Data

All the above introduced systems [2][3][5], doesn't matter if indexed semantically or not, rely on a server-centric model. They store their generated and/or manually added data in a proprietary format with a proprietary program into a proprietary database. Therefore, it is impossible to share multimedia data independently with various applications and machines.

In [7], Kumar and Babu emphasize to include not only indices directly into the video stream, but also the software, which knows how to use such indices. The increase of the file size for software and indices is not an issue, since the raw data of multimedia streams (eg. a movie) is many times bigger.

In their demo implementation, [7] use MPEG-1 and the object oriented programming language $Java^{TM}$. MPEG-1 is a well-accepted standard[1] for compressed video/audio streams. Fortunately, MPEG-1's internal data structures allow the storage of user-data in the main header, in every GOP[7] and even in every frame. The pre-compiled platform-independent Java byte code is easily inserted into the user data areas and can be loaded on the fly by the Java class loader. Readers are referred to the Java language tutorial[9] for more information. For inserting and reading the classes and indices out of the MPEG stream, you need a modified encoder and decoder software. [7] uses the omnipotent Berkeley encoder[8] and decoder, since it is free software and all (well documented) sources are available. All other decoders (free and commercial ones like xing) ignore all data stored in the user fields.

The proposed video indexing software is heavily based on AVIS' data structures[3]. So we have *objects* O like {Billy the Kid, colt}, and *actions* A like {sing, kill}. The representation of an object's action in exactly one frame is called an *event*. An *activity* lasts over more frames, with a start event and an end event. Additionally indexing based on color histograms is implemented.

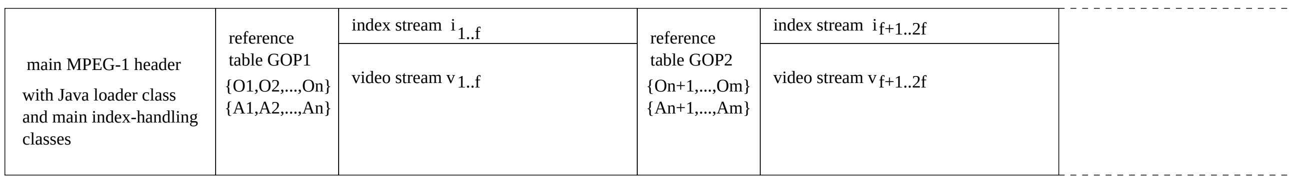


Figure 6: Storage format for indexed video with reference table

In figure 6 such an indexed MPEG-1 stream is shown. The main Java classes are stored in the main header and every GOP[8] does not only contain a list of objects and actions but also, if necessary, new Java classes to handle this data. To save space, one could only store the difference of object and action lists (persons appear and leave a scene). The index i_p stores the occurance of objects and actions for the frame v_p. Since these indices only refer to objects introduced in the preceeding reference table of this GOP, the object list and action list is highly dynamic. This allows small lists, because only few objects and actions will be needed in a GOP. On the other hand, using this approach of incremental introduction of objects and actions, an overall video search needs to sequentially look at each and every reference table. So for fast querying, only one reference table in the front is much more efficient. Also for searching in video databases, it is unfeasable to sequentially look through all streams for keywords. Again, we will need really quick databases for keyword search. Maybe it would

[7]Group of Pictures (GOP) is a data structure in MPEG-1 containing full video information data for synchronizing and a fixed number of frames.

[8]As mentioned above, this could also be for every single frame

make sense to store all indices into the stream but also maintain a database apart from the raw data.

4.1 Video Content Authorization

Another usage of this in-line-data could be authorization. Different encrypted passwords could be inserted throughout the video and by providing[9] a player class for the stream, a user would be authorized by a password. So one could easily block scenes showing brutality for kids. The embedded Java classes may also modify their own stream on the fly to keep track of usage, pay-per-view, or download location tracking. This also enables the intelligent adaption to different users' knowledge levels on certain topics. In [7], Kumar and Babu did not seem to realize a little malfunction in this approach: as mentioned above, every non-adapted MPEG-decoder will ignore the user-specific data. A simple solution would be to scramble/encrypt the whole video (or parts out of it), so it can only be watched by the modified decoders using the embedded Java classes and verification algorithms.

5 Conclusion

In this paper, we have seen the need for video indexing for efficient indexing, and the problem of indexing the right things to meet the users requirements for their queries. There is non-semantic indexing with IBMs QIBC[2] for color, motion detection, etc. This information can be extracted (semi)automatically by an indexing software. For semantic indexing, somebody has to index manually shot by shot of a video stream. To do this, there are different approaches like AVIS[3] and OVID[5]. The most powerful approach is to mix non-semantic and semantic indexing to provide wide-spread query types by different users. To get rid of server-centric video indices, in [7] self-describing in-line indices with interpreter classes are introduced. To show that industry has accepted the need for indexing, the (not yet finished) MPEG-7 standard will include definitions for in-line indices and interpreter information. In the already finished MPEG-4 standard, different object layers allow rudimentary search on keywords for objects and include programmable interactivity in the stream.

[9]This player class is directly packed into the stream.

References

[1] MPEG-1 Standard (ISO/IEC International Standard 11172-2)

[2] Flickner M, Sawhney H, Niblack W, Ashley J, Hunag Q, Dom B, Gorkani M, Hafner J, Lee D, Petkovic D, Steele D, Yanker P, “Query by Image and Video Content: The QBIC System”, IEEE Computer Sept 1995, pp24-32

[3] Adali S, Candan K, Chen S, Erol K, Subrahmanian VS, “The Advanced Video Information System (AVIS): Data Structures and Query Processing”, IEEE Multimedia Systems (1996) 4, pp172-186

[4] Gong K, Rowe L, “Parallel MPEG-1 Video Encoding”, 1994 Computer Science Division - EECS University of California, Berkeley

[5] Oomoto E, Tanaka K, “OVID: Design and Implementation of a Video-Object Database System”, IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No 4, Aug 1993, pp629-643

[6] Steinmetz R, “Multimedia-Technologie: Grundlagen, Komponenten und Systeme” 2. Auflage, ISBN 3-540-62060-5 Springer Verlag Berlin Heidelberg New York

[7] Kumar P, Babu G, “Intelligent Multimedia Data: Data + Indices + Inference”, IEEE Multimedia Systems (1998) 6, pp395-407

[8] Berkeley MPEG-1 Video Encoder “User’s Guide”, 1995 Plateau Research Group, University of California, Berkeley

[9] Gosling J, McGilton H, “The Java Language Environment: A White Paper”, 1995 Sun Microsystems, Mountain View, California

[10] Roland Tusch, “Content-based Indexing, Exact Search and Smooth Presentation of Abstracted Video Streams”, 1999 Thesis for the degree of Diplomingegieur, University of Klagenfurt