Browsing Video Along Multiple Threads

Ork de Rooij and Marcel Worring, *Member, IEEE*

***Abstract*—This paper describes a novel method for browsing a large video collection. It links various forms of related video fragments together as threads. These threads are based on query results, the timeline as well as visual and semantic similarity. We design two interfaces which use threads as the basis for browsing. One interface shows a minimal set of threads, and the other as many as fit on the screen. To evaluate both interfaces we perform a regular user study, a study based on user simulation, and we participated in the interactive video retrieval task of the TRECVID benchmark. The results indicate that the use of threads in interactive video retrieval is beneficial. Furthermore, we found that in general the query result and the timeline are the most important threads, but having several additional threads improves the performance as it encourages people to explore new dimensions.**

***Index Terms*—Conceptual similarity, information visualization, interactive search, multidimensional browsing, semantic threads.**

I. Introduction

FINDING videos in a large collection can be done quickly if each video has a descriptive title or an extensive textual description. In that case standard text retrieval approaches suffice to find the video. In other cases the user probably has an idea what he is looking for, such as who was present, what was happening or who was talking. Such descriptions are intuitive for a user, but for a computer they are difficult to extract from video automatically, a problem known as the semantic gap [1]. This problem leads to low quality retrieval results, which requires the user to extensively browse through these results. For a content-based video retrieval system (CVBR) this browse process needs to be leveraged.

In order to understand the video retrieval process better we split the task of finding something in a video collection into three phases. The first phase is the *query-phase* where the user expresses her information need in any of the query modes provided by the system. She then examines the retrieved video results on the two-dimensional screen in the *view-phase*. If required, she is able to interact with this visualization in the *browse-phase* going through these and related results. The overall process is illustrated in Fig. 1. Note that the differentiation into phases here is on a functional level. In practice the interfaces for two or more phases can be combined.

Manuscript received August 05, 2008; revised December 22, 2008. First published November 24, 2009; current version published January 20, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Michael G. Christel.

The authors are with the Intelligent Systems Lab Amsterdam, University of Amsterdam, 1098 XG Amsterdam, The Netherlands (e-mail: orooij@uva.nl; worring@uva.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2009.2037388

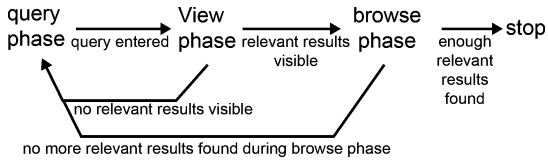

Fig. 1. Overview of the three functional phases in video retrieval. A search starts in the query phase. The user then examines the results in the view phase, followed by the browse phase in which the user looks up related results. In both the view and the browse phase, the user can choose to go back to the query phase.

Let us first look at typical web-based video search engines, which use only the textual modality for retrieval. Here the phases are instantiated as follows. In the *query-phase*, the user enters a set of keywords with optional filtering options, such as the language to search in, domain of search, or type of retrieval method to be used. The computer then provides a ranked list of results based on these parameters. The user now enters the *view-phase* where she looks at the resulting documents, represented in the form of a list. When no valid results are present in the view, she can go back to the *query-phase*, for example, to enter new keywords. This loop between query—view often leads to sufficient valid results, and the *browse-phase* is often ignored in web-based engines.

In order to do CBVR we need features as a basis, where systems typically use more than one modality. Each modality comes with a different set of features for which state of the art extraction methods exist. For example, *visual features* such as color or texture patterns present in the image frame, or *textual features* from on-screen text, or as the result of speech recognition. The video timeline defines the *temporal features*. Also *metadata features*, such as air time and date, channel, program and language, are extracted from the context of the shot. Each type of feature has a varying level of quality. On top of this, all of these *low level* features are used in building *conceptual features*, which are proven to be beneficial for CBVR [2]. Methods such as [3]–[6] allow automatic labeling of people, objects, settings or events. Conceptual features are typically based on fusion of various features in combination with supervised machine learning using large collections of labeled examples. A collection of such conceptual features, or concept detectors forms a lexicon. Examples from a representative lexicon such as LSCOM [7] may range from people like *Hu Jintao*, objects like a *truck*, settings like *indoor* and events like *people marching*. Together with the low level features this defines a wide gamut of features which can be used in CVBR systems.

Moving on to search, we observe that for a CBVR system the problem of defining a query is much harder. First, the various modalities increase the possible means of entry into the collection. Furthermore, in video retrieval instead of returning whole

documents often only fragments of videos are needed. In particular in CVBR the shot is often used as the unit of retrieval. A long video can easily contain hundreds if not thousands of shots. This increases the number of possible results enormously. Lastly, due to the semantic gap, the quality of results is much lower, so more results need to be inspected before a user is satisfied with the results. Therefore, we need new retrieval methods optimized for the specifics of CBVR.

Methods in the literature, e.g., [8]–[12], optimize the search process in different ways. In Section II we discuss those in more detail. In Section III we present our new method optimizing the browse phase, which is evaluated thoroughly in Section IV.

II. Optimizing Video Search

We have defined three phases in video retrieval, the query-phase, the view-phase and the browse-phase. In this section we discuss possible optimizations for each phase which currently exist in literature.

A. Optimizing the Query-Phase

All extracted features yield different points of entry into the collection. In the *query-phase* a combination of query by text, query by example, query by concept or query by metadata is used by various systems as the entry point into the collection. A consequence of having this many options is the increased difficulty a user experiences in setting options before results are shown. Also, since the quality of individual feature extraction methods vary, retrieval sometimes leads to low quality results. These two problems increase the number of times a user has to go back and forth between the query- and view-phases. In order to optimize this, systems combine queries [13]–[15], and re-rank results [16], [17]. All of these approaches yield optimized ranked lists of results. We postulate that not only query-phase optimization is required in order for a computer to aid in video retrieval, but view- and browse-phase optimization are equally important.

B. Optimizing the View-Phase

One way to avoid switching between phases is to combine the visualizations of the query-, view and browse phase into one. For this approach the high dimensional feature spaces need to be mapped to a 2-D visualization space, so that feature querying is possible on the 2-D space itself. This can be done both in a structured and an unstructured manner. An unstructured method allows visualization of shots with their individual distances visible. This is done in [12] where a collection is visualized as a cloud of distance-related images. Navigation in this cloud gives users a high degree of interaction freedom. More structured approaches use graphs to visualize results and their relations to each other [18], or use hyperbolic tree visualizations [19]. This allows for a structured but constrained navigation. All of these methods are limited by the available screen real estate, which limits the number of pictures that can be shown. Similarity-based visualizations have utility, but by limiting the user to view results which are already similar to each other there is a possibility that other parts of the dataset, which are relevant but not similar, are never visited.

View-phase optimization is possible by reducing the time required for the user to decide that results are not relevant. From the query phase we obtain a single ranked list of results. In its simplest form this list is visualized as a linear list. In practice most systems extend this to a grid-based representation, such as in [20], and use implicit reading order to display the ranked list. This allows for constrained and fast navigation through the list.

More advanced approaches use either clustering of results into semantic units, or enhance the speed at which the user is able to analyze the results. In this first category falls the FXPAL MediaMagic interface [21] which segments the resulting video into stories. These stories allow the user to evaluate the relevance of whole sections of results. The system in [22] clusters pages of result into visual islands, and displays these in a grid. This removes implicit reading order, adding meaning to the distance between shots in the grid. Informedia [8] falls in the second category. They use rapid serial visual presentation in the view phase to effectively explore the results at high speed. The system in [9] also uses RSVP, and combines this with a relevance feedback method on selected results to create new lists of results. However, no matter how a single list is displayed, it still constrains the quality of the results to the quality of the ranked list obtained in the query phase. Only the decision to either be done with finding results, or to go back to the query screen to retune parameters can be made faster.

C. Optimizing the Browse-Phase

The browse phase allows the user to navigate away from the initial set of results, and delve into the dataset based on the features themselves. An efficient browse-phase therefore reduces the number of times a user has to switch between phases, which improves search task efficiency. However, designing a good browse-phase in such a way that it helps and not hinders the user is not trivial. There are a number of ways to improve upon the browse phase, which we will now describe.

One method is using multiple linked interface windows, e.g., [10], [21], and [23]. Each window depicts one view on the results. When a user selects a result in one window, the other windows are updated with the chosen result as a starting point. For example [10] combines the results with a timeline and a graph of semantically related shots. The system in [23] clusters related results together using a self-organizing map. This is then displayed in a hexagonal grid together with a time bar control in separate windows, which allows the user to browse though both time and similar results. A downside of using separate windows is that there is no direct visual correlation between the contents of each window, a user has to switch from one window to another, which makes user response slower.

An even more structured approach is to display two lists in two dimensions as a rigid grid like representation. The system in [11], for example, provides a grid like representation with temporal ordering on the horizontal axis and shot similarity on the vertical axis. The system used in [24] combines this approach together with temporal zooming and additional windows showing up to ten live lists of related results.

To summarize, there are three ways to speed up video search. Optimizing the search phase increases the quality of the set of results. Optimizing the view phase decreases the time required

to determine if the set of results is adequate. Finally optimizing the browse phase reduces the number of times a user has to switch between the query and view phases. We propose a video retrieval system which primarily focuses on browse phase optimization.

III. Thread-Based Video Search

In this section we give an overview of our proposed method for searching through video using threads. First we will define thread-based video search more precisely. We then explain how threads are computed from several types of features. Finally, we describe a generic framework which uses threads to aid users in navigating video retrieval results.

Definitions

A content-based video retrieval system operates on a set of shots from a collection of videos V. We define these as follows.

Definition 1: A **video collection** $V = \{s_1, \ldots, s_n\}$ is the set of all shots.

As indicated, ranked results from the query phase form only one type of relevant information. These shots are linked because they are answers to the same query. Other forms can also be defined: the timeline of a video, or shots containing the same visual or semantic characteristics. These are all *threads*, based on the *dissimilarity spaces* induced by the features. We define such a dissimilarity space with associated threads as follows.

Definition 2: A **dissimilarity space** D^f_{pq} defines the distance between any two shots $p, q \in V$ for a specific feature f.

Definition 3: A **thread** τ_D is a linked sequence of camera shots from V in an order determined by the dissimilarity space D.

To aid the user in navigation there must both be a static structure to help the user browse the collection, and a means of dynamic querying into the collection when the user sees something interesting which he wants to explore further. For this we introduce two types of threads.

Static threads are data collection driven. They allow a structure to be placed on top of the collection beforehand.

Definition 4: A **static thread** τ_D is a thread where the order is determined by a structure induced by dissimilarity space D only.

In addition to static threads we have dynamic threads, which are instantiated at the moment the user explicitly searches for something. This instantiation is possible at several points during retrieval. Within the query-phase, the user instantiates a dynamic thread when he configures a dissimilarity function for a certain feature. Within the browse-phase, the user instantiates a dynamic thread by doing a "find related" type of query. This query takes the selected shot(s) as the basis for the dissimilarity function.

Definition 5: A **dynamic thread** τ^*_{D,s_i} is a thread where the order is determined by similarity based on D to user selected $s_i \in V$.

The timeline of a video is a special case of static threads, since no processing is required to obtain them. We define this as follows.

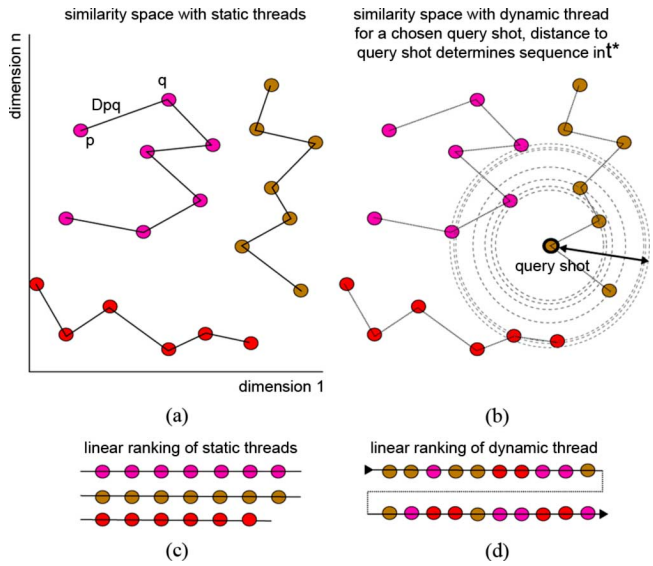

Fig. 2. Example of how shots in a dissimilarity space can be configured into either (a) multiple static threads or (b) a single dynamic thread based on a user selected shot s_i. Images (c) and (d) depict the resulting threads.

Definition 6: A **time thread** τ_{T,s_i} is a thread where the order is determined by the timeline of the video the user selected $s_i \in V$ is part of.

A. *Creating Threads*

We now present a generic framework for creating threads from features. Each type of feature, such as visual features, textual features, temporal features, auditory features or metadata features yields a multidimensional feature space. Each shot in V is a point in such a space. The dissimilarity function for each type of feature determines distances between shots within each feature space. From the dissimilarity space we create both static and dynamic threads. An example is given in Fig. 2.

Dynamic threads are created by calculating the distance in D for all shots to the given s_i. See algorithm 1. All shots are ordered lowest to highest distance in a new τ^*_{D,s_i}. Since the distance from s_i to s_i itself is 0 this shot is placed first.

Algorithm 1: Dynamic Thread Generation Given a Dissimilarity Space and a Query Shot

D_{pq}: dissimilarity space for all pairs of shots p, q

s_i: user selected query shot

τ^*_{D,s_i}: resulting thread

$\tau^*_{D,s_i} \leftarrow sort\ D_{pi}$

Static threads dissect the entire dissimilarity space D into individually meaningfull threads, where the shots within each thread still form a semantic group of data. On top of this the individual shots within each thread are placed in a linear order so that the shots next to each other are similar. This yields a two level hierarchical grouping. The static threads for a given D are calculated as shown in algorithm 2. First, we consider

the entire dissimilarity space to find shots sharing similar features, i.e., having small distances in the dissimilarity space. To find such groups we perform k-means clustering. The elements of each cluster define the elements of each static thread. The cluster already forms a group of similar shots based on its dissimilarity metric. The next step is to add a linear ordering of the shots in the cluster. We obtain an ordering by connecting the shots inside each cluster so that each shot links to its closest unconnected neighbor. We use the cheapest link algorithm on each cluster to achieve this. This algorithm connects shots with the shortest distance to each other to form a single static thread. For more details see algorithm 2. All threads together yield the set of static threads for the given dissimilarity space, with every shot in the collection linked in one such thread.

Algorithm 2: Static Thread Generation Given a Specified Dissimilarity Space. All Shots are Clustered into Clusters With Small Internal Distance. A Variant of the Cheapest Link Algorithm is Used to Project Each Cluster to a Static Thread

D_{pq}: dissimilarity space for all pairs of shots p, q

con_x: number of connections for shot x

$\tau_{D,n}$: resulting static thread n for each cluster in D

$C_1 \ldots C_n$: $\leftarrow$ K-means clustering on entire D_{pq}

for each cluster $c \in C$

  for all shots $x \in c$:

    $con_x = 0$

  $S_{sorted} \leftarrow$ **sort** $D_{pq} p \in c, q \in c$ ascending

  for each pair of shots p, q in S_{sorted}:

    if $con_p \leq 1$ **and** $con_q \leq 1$

      increment c_p **and** c_q **by** 1

      append $(p \rightarrow q)$ **to** τ^c

  until all shots in cluster are connected

until all clusters are processed

Summarizing the above, both static and dynamic threads are based on a dissimilarity space for a specific feature. As a consequence many different threads types are computable. We define the following threads from the following modalities: a textual thread τ_t containing shots with similar textual annotation, a visual thread τ_v with visually similar shots constructed from low-level visual features, a semantic thread τ_s containing semantically equivalent shots constructed from high-level textual and visual features and a query thread τ_q which is formed from the list of results provided by the initial user query.

B. Visualizing Threads

In this section we describe a method for visualizing threads in a CBVR system on a regular screen. Navigation starts with the top most shot from the initial query thread τ_q^d. This shot is the initial focal shot s_f, defined as follows.

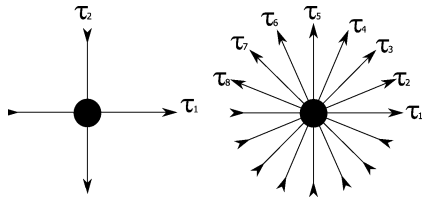

Fig. 3. Multi-thread dimensions for each browser. (Left) The CrossBrowser visualizes a fixed number of two threads. (Right) The RotorBrowser is able to visualize up to eight threads.

Definition 7: The **focal shot** s_f identifies the current position in the video collection.

Since the focal shot always identifies the current position in the video collection it stands to reason to let this shot have the most user attention in the visualization. We choose a visualization where s_f is always the largest and most central shot on the screen. From this shot we form navigation paths through related threads. Therefore, threads which contain S_f are added as navigation possibilities, where s_f is used as s_i for dynamic threads. Threads are displayed in a star formation around s_f.

For any s_f there are multiple relevant threads from various dissimilarity-spaces. An interface showing all these threads to the user might be too overwhelming. On the opposite side: an interface that only displays the bare minimum of relevant threads might miss the optimal thread. So we need to know which thread is more relevant, and how many threads we want to visualize simultaneously.

In order to ascertain the optimal number of visualized threads we have designed two browser variants. One variant—the CrossBrowser [25]—deliberately limits the number of threads shown. It shows the user the result from their initial query thread and only allows the user to also browse through the time thread which is shown to be important [11]. The other variant—the RotorBrowser—shows up to eight relevant threads, including time, for the focal shot s_f. The ordering of threads in the RotorBrowser is important because higher ordered threads require less user-effort to access. Furthermore, in order to prevent display clutter there is a fixed maximum number of threads that can be shown. To make it comparable with the CrossBrowser we choose query results and time as the most important threads. Semantic, visual and textual threads are all based on their dissimilarity to the focal shot. Since semantic dissimilarity conveys the context of a shot, we decide that semantic dissimilarity here is more important than visual dissimilarity. Moreover, since the RotorBrowser was designed for video collections where text is not always available, the textual dissimilarity thread has an even lower priority. Finally there are top-rank threads, indicating the focal shot is present in the top-n best shots for a single specific semantic concept. These top-rank threads, which can occur multiple times, are used as "filler" threads when there is still room in the visualization. We have listed these possible threads and their priorities in Table I. The difference between the browsers is visualized in Fig. 3.

C. Search Using Threads

A search session starts with the user entering a set of parameters in the *query phase*. These parameters yield a list of initial

TABLE I
TYPES OF THREADS DEFINED IN THIS PAPER, AND THEIR PRIORITIES. A HIGHER PRIORITY MEANS THAT THE THREAD IS MORE IMPORTANT TO BE INCLUDED IN A VISUALIZATION. THE TYPE COLUMN DEPICTS WHETHER A CERTAIN THREAD CAN BE STATIC (S) OR DYNAMIC (D)

Priority	Thread	Type
1	τ_q - initial query	D
2	τ_T - timeline	S
3	τ_s - semantic	D or S
4	τ_v - visual	D or S
5	τ_t - textual	D or S
6	τ_r - top rank	D

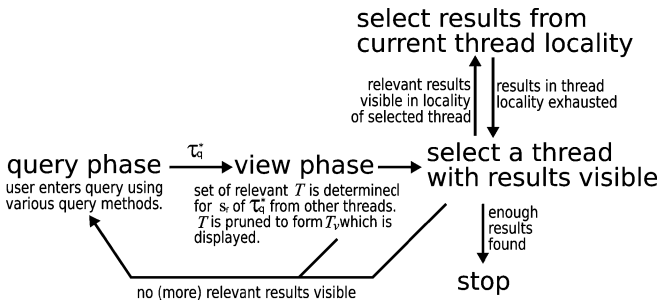

Fig. 4. Finite state machine for thread-based browsing. A search starts when the user defines the query in the query phase. He then views the results in the view phase. If initial results are satisfactory he navigates through the initial query thread. When one thread is insufficient he switches threads and browses the new thread.

results, which become the dynamic thread τ_q^*. The first item of this thread becomes the first s_f. The current s_f then determines which other threads with relevant shots are to be shown. One of these threads is then selected as the active thread through which the user navigates. To keep navigation deliberately simple, the user is limited to three options. He can accept the focal shot as relevant to his query, he can navigate through the active thread to select more results, or he can select another thread.

A more formal description of this search process is illustrated in Fig. 4. Note that in this section when we refer to the user we imply an ideal user who performs optimal search based on what she sees and knows about the dataset, but without additional world knowledge which might impact her decisions. Realistically, a user will also act on any possible own knowledge about the dataset and act on own hunches, which will alter the chosen search strategy.

Now, when a user views threads using, e.g., the CrossBrowser or RotorBrowser, he sees a number of shots spreading outward from the focal shot. We use w to describe this number. So, for example, $w = 6$ means that six shots are visible from the focal shot in either direction for all threads shown. The value w here influences the user's browse behavior. A very small value of w will lead to the user seeing almost no shots per thread, which means that possible relevant results in other threads will not be retrieved. A too large value of w shows a lot of shots per thread, increasing the time required for the user to view them all and comprehend the visualization, which therefore slows down search.

During browsing the user retrieves relevant shots by 1) making them the focal shot, and 2) selecting them. The act of making any visible shot the focal shot will update the currently

Fig. 5. This figure shows how a thread locality is defined step by step for $w = 2$ shots shown on either side of a selected shot.

visible set of shots. As a consequence, all shots within w steps distance from the focal shots along all visible threads are now shown in the browser. Among these can be new relevant shots. These are now also candidate for selection by the user, and when these are selected again new shots can appear which are also relevant. This process repeats itself. As such, individual threads can have whole *localities* of relevant shots, defined as follows.

Definition 8: A **thread locality** for an information need is a set of consecutive shots within a thread where the number of irrelevant shots between relevant shots for that information need is lower than w.

The user therefore browses through individual threads by browsing individual thread localities within these threads. We give an example of how such a locality is created in Fig. 5. Again, since we assume an ideal user, we assume here that the user does not want to browse through a thread in which he sees no relevant shots.

The search session ends when the user is satisfied with the results he found so far, or when he cannot find any new relevant results by browsing in any thread direction.

IV. EXPERIMENTAL SETUP

In order to evaluate thread-based browsing we performed three video retrieval experiments. In the first experiment we compare linear browsing to browsing using static threads based on simulated users. In the second experiment we evaluate the practical use of threads by a user study. The goal of this user study is twofold: we want to discover the benefit of having multiple threads visible, and if so: which type of query benefits most. And, secondly, which threads are best in providing results for the user? In the third experiment we compare thread-based visualization with other state of the art video retrieval systems. For this we participated in the TRECVID Interactive Search task [26], the *de facto* standard for video retrieval evaluation. A general overview of successful approaches for TRECVID is given in [27].

TABLE II
TOPICS FOR TRECVID 2006 BENCHMARK. MARKED TOPICS (*) ARE USED IN EXPERIMENT 2. THE REMAINING COLUMNS INDICATE RESULTS FROM EXPERIMENT 1, WITH THE OPTIMAL CONCEPT DETECTOR WITH THE HIGHEST RECALL AFTER 100 AND 1000 VISITS, AND WHETHER THE OPTIMAL SEARCH STRATEGY FOR THIS TOPIC IS TO SEARCH BY CONCEPT, OR TO ALSO USE TIME LOCALITIES OR SEMANTIC LOCALITIES. FOR TOPICS WHERE THERE WAS NO CLEAR OPTIMAL CHOICE, WE MERGED THE BEST TWO OPTIONS

1	emergency vehicles in motion	vehicle	truck	concept	semantic
2 *	tall buildings with top story visible	tower	building	concept	semantic
3	people leaving or entering a vehicle	pedestrian zone	ground vehicles	concept	time
4	soldiers, police, or guards escorting a prisoner	sunglasses	military	concept	time
5 *	daytime demonstration with a building visible	people marching	people marching	concept or time	
6 *	US Vice President Dick Cheney	flag USA	Bush jr	concept or time	
7 *	Saddam Hussein and another person's face	Powell	male reporter	time	concept
8	people in uniform and in formation	military	soldiers walking	concept or time	
9 *	US President George W. Bush, Jr. walking	flag USA	Bush jr.	concept	time
10	soldiers with weapons and military vehicles	road	machine guns	concept	time
11 *	water with boats or ships	waterways	waterscape/front	time	concept
12 *	people seated at a computer, display visible	computers	chair	concept	semantic
13	people reading a newspaper	newspaper	newspaper	time	
14 *	natural scene w/o buildings, roads or vehicles	tree	mountain	concept	time
15 *	helicopters in flight	aircraft	airplane flying	concept	semantic
16 *	something burning with flames visible	smoke	explosion	concept	time
17 *	seated people dressed in suits, with flag	walking	meeting	concept or time	
18 *	at least one person with at least 10 books	flag USA	monologue	concept or semantic	
19	at least one adult person and at least one child	animal	crowd	time or concept	
20 *	greeting with a kiss on the cheek	person	old people	concept or semantic	
21 *	smokestacks with smoke or vapor coming out	tower	tower	concept or time	
22 *	Condoleeza Rice	speaker at podium	speaker at podium	time	concept
23 *	soccer goalposts	stadium	grass	time	concept
24	scenes with snow	snow snow	time	concept	

A. Data Set and Task

The video collection used in all experiments is the TRECVID 2006 dataset [26], which consists of 320 h of broadcast news video from Arabic, Chinese, and American news channels. This collection is split into a training set of 160 h, used to develop the system and a test set of 160 h, which is used for evaluation and not revealed to users until the search experiment starts. The test set contains 259 video files, totalling 79 484 individual shots. TRECVID also defines a list of 24 search topics relevant to this set. See Table II for an overview of these topics.

The interactive evaluations are set up as follows. Each participant is given a list of topics for which they need to find relevant shots within a 15 min time limit per topic.

B. Threads

Textual threads τ_t are obtained by taking the text linked to a shot. This text is based on the output of automated speech recognition systems. We obtain the textual dissimilarity space by performing textual document dissimilarity analysis on each shot [25].

For visual threads τ_v we need a notion of visual dissimilarity between shots. In our system we use Wiccest features which are based on natural image statistics with related dissimilarity function from [28], and Gabor features. These two *visual dissimilarity spaces* form the basis for visual exploration of the dataset.

To determine the semantic threads τ_s we first extract a description of the semantic content of a video. We use the semantic pathfinder [6] which provides us with a measure of presence P_j for a given concept j in each shot. As lexicon we combine the LSCOM [7] and MediaMill [25] lexicons, resulting in 491 concept detectors. This results in a concept vector, generic enough to describe the semantic content of a shot. This method is similar to the method of [29] which uses semantic threads to cluster semantic concepts into semantic dimensions, though they use a manually annotated dataset with binary yes/no states for concepts, instead of automatically generated confidence scores.

Typically only a few concepts will be available in a shot, so D^s should be chosen such that it explicitly values shots sharing the same concepts while ignoring concepts present in one of the shots only. We therefore compare shots on a semantic level by comparing semantic concept vectors of the individual shots with the dissimilarity function D^s given in 1 . This metric yields high similarity when concepts are available in both shots. This yields the *semantic dissimilarity space*, which forms the basis for semantic exploration of the dataset:

$$D^s_{pq} = \sum_i \frac{\min(p_i, q_i)}{\max(p_i, q_i)} \quad p, q : \text{the feature vectors.} \qquad (1)$$

C. Evaluation Measures

TRECVID provides an incomplete ground truth on the test set for all topics. This ground Truth describes each shot as relevant for a topic, irrelevant for a topic or "not judged" for a topic based on the answers given by official TRECVID participants. This set therefore does not include the results found by the user study participants. Therefore, we extended the ground truth manually to also cover all results from the user evaluation, so that there were no unjudged shots in the results found.

For the overall performance we use average precision [26], which is a single valued measure related to precision and recall, favoring good results at the beginning at the list. The perfect score of 1.0 is achieved by returning all correct results within the dataset as the first results. Finally, the mean average precision is obtained by averaging the average precision scores for all topics.

The average precision allows us to evaluate overall performance. However, looking at the final result only does not show how a user achieves this. Therefore, in addition we measure the

Fig. 6. Screenshots of the (left) CrossBrowser and (right) RotorBrowser both searching for the same topic. The CrossBrowser displays τ_q in the vertical direction, with τ_t in the horizontal direction, while the RotorBrowser visualizes more relevant threads.

recall as a function of the number of keyframes viewed by the user.

D. Implementation and Parameter Settings

We implemented both browsers within the MediaMill semantic video search engine [25]. See Fig. 6 for screenshots of both browsers.

Both browsers are designed to allow for keyboard-only interaction. The CrossBrowser has keyboard controls to move the focal shot up, down, left or right. The RotorBrowser only allows to move the focal shot left or right, and has a special rotate operation to rotate the browser clockwise to the next-active thread. By repeating this action the user is able to cycle through the visualized threads. Futhermore, both browsers are able to playback up to 16 individual frames of the focal shot by holding a key.

The CrossBrowser was configured to show only τ_q^* and τ_t^*. The RotorBrowser was configured to show up to eight threads. We determined this number by gradually increasing the number of visible threads, until there was too much thread overlap on screen to make the threads useful. Thread relevancy is determined by 1) the focal shot being present in the thread and 2) the thread importance hierarchy listed in Table I.

1) Experiment 1: Evaluate the Benefit of Static Threads: In this experiment we evaluate the benefit of browsing static threads over browsing one linear list of results. We do this by evaluating a list of shots on a specific topic, and we measure the number of shots viewed, i.e., the position in the list, against the number of relevant shots found so far. To measure the benefit of static threads we simulate user search with the finite state machine depicted in Fig. 4 where we assume the user recognizes a relevant shot when he sees it, and always chooses to select it.

For each topic we determine the thread localities for the timeline and for the set of static semantic threads τ_s for various sizes of w. This yields lists of shots which would be browsed through if the user were to encounter them. We then use these localities as follows. First we select the best concept detector available for the topic. This is defined as the concept detector with the most relevant results in the first 1000 shots. We assume that this is the concept the user would have chosen to find results for this topic, although in practice this might not be the intuitive choice. Table II shows a list of the selected detectors for each topic. From this detector and the generated localities the system determines the list of shots the user would encounter during his browsing session. This is done as follows:

- Method 1: browsing through a linear list
  1) take the most optimal concept detector for the topic;
  2) add all shots to the shots-viewed list.
- Method 2: browsing using static threads
  1) start with most optimal concept detector for the topic;
  2) Iterate over each shot. If the shot is embedded in a locality for a specified w, add the entire locality to the shots-viewed list. Otherwise only add the shot to the shots-viewed list.

This yields several one dimensional lists of shots, one list based on linear browsing, one list based on browsing with additional static semantic threads, and one based on static time threads. Each list is truncated after 4000 results, and then validated using the ground truth as described in Section IV-D. This yields recall vs shots visited statistics for both methods. We redo this experiment for several values of w—thereby simulating various browser sizes. Finally, in order to assess the benefit of localities we also measure the oracle-based recall based on a simulated user visiting all semantic localities in order from the largest locality, i.e., with the most positive results, to the smallest locality which still has more than one result. This gives us the theoretical maximum achievable gain with locality-based browsing and an indication of how well the semantic threads are organized for this topic.

2) Experiment 2: Evaluate the Benefit of Extra Threads: In this experiment we evaluate if there is a benefit of having multiple threads visible, and if so, which threads are optimal. We evaluate this with a user study with 32 participants. The participants ranged from 21 to 26 years of age. Only 29% indicated they have experience going beyond browsing with Google Video and Youtube. Each participant was given two hours of training time with both browsers using a different dataset one week beforehand.

The user evaluation was set up in a similar fashion to the TRECVID Evaluation guidelines [26]. The users were given questionnaires before and after the evaluation. Due to a time-limit 16 topics were chosen from the selection, see Table II. We divided the 16 topics across the 32 participants. The topics were divided in such a way that each topic was completed 4 times, twice using each browser. Also, each participant completed two topics using the RotorBrowser and two topics using the CrossBrowser.

3) Experiment 3: Evaluate the Efficiency of Threads: We compared the system with the current state of the art in video retrieval systems by participating in the TRECVID Interactive Search task [26]. Note that in this experiment we rely exclusively on the official TRECVID ground truth as provided by NIST.

V. Experimental Results

A. Experiment 1: Evaluate the Benefit of Static Threads

Results for experiment 1 are visible in Table II. For each topic we have also given the optimal concept to be used. As can be

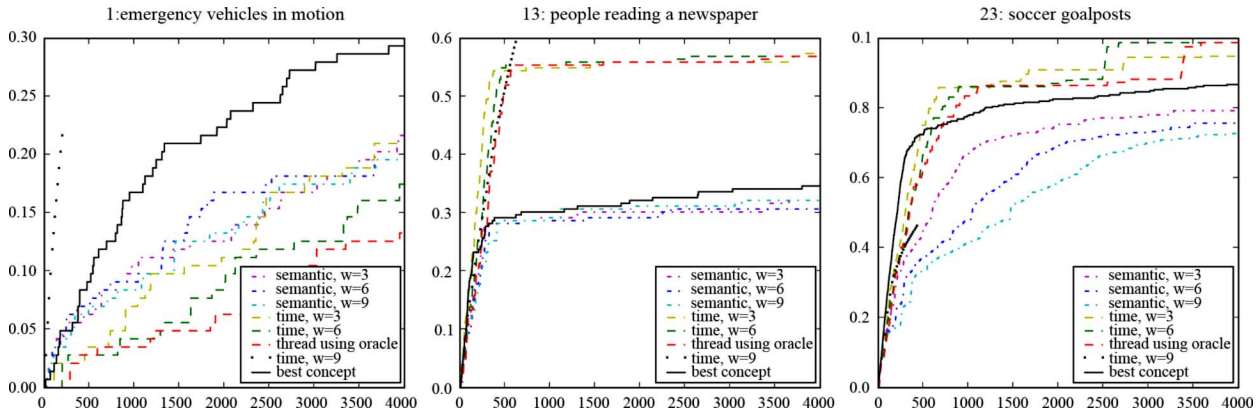

Fig. 7. Detailed results for experiment 1: these graphs plot the number of viewed results against the achieved recall at that moment for three topics. The left graph shows results for the topic "find shots of emergency vehicles". The results indicate that using the "truck" concept detector results in better performance than using either of the static threads. The right graph, "find shots of soccer goalposts", shows that browsing using the timeline static threads yields a higher recall compared to browsing with the "grass" concept alone. The middle graph, "find people reading a newspaper", is an odd example where time localities significantly help retrieval when compared to linear concept browsing only. After 500 visits a recall of 0.55 with 110 shots was achieved, where for linear browsing this was 0.29 with 60 shots.

seen, using concepts detection results alone without other dimensions often yields the best results with respect to the number of shots viewed, though it must be said that the difference with time localities is often marginal.

We highlight a more detailed selection of topics in Fig. 7. Each graph shows the number of images found against the achieved recall at that point for several locality widths. Also, results with linear browsing using the optimal concept after 1000 views are shown. Lastly, the optimal performance using only static semantic thread localities with $w = 6$ is shown. From these results we observe the following trends.

Not surprisingly, topics which have a clear best-concept benefit most from single list concept browsing. For example using concept *grass* for topic 23: *find shots of soccer goalposts*, shown in Fig. 7(c), yields a recall of 0.8 after 1500 keyframes viewed. The timeline boosts this to 0.85 recall at the same number of images viewed. Also, the results are significantly boosted to a recall of almost 1.00 after 2500 images viewed for a locality width of 3. This boost indicates that the simulation found a yet-undiscovered set of soccer videos with many positive results. This pattern is repeated in several topics in Table II. For example, the optimal concepts for topic 2: *tall buildings* are *tower* or *building*.

It was also expected that timeline locality-based browsing yields worst performance for topics which have the subject in view for only a fraction of a second, such as topic 1: *find shots of emergency vehicles*, as shown in Fig. 7(a). Browsing with semantic localities yields better but still limited results for all values of w. This can be explained by looking at the semantic locality oracle. There are only 12 localities which would yield a recall of 0.22 if they are all found. This did not happen however, the graph indicates that a user browsing the optimal concept did not find all of these localities.

Topics where no clear best-concept is available benefit most from static semantic threads. For example, for topic 20: *find shots of a kiss on the cheek* the best concept detector is *old people*, but this only yields a recall of 0.14 after 1000 keyframes viewed. Semantic threads do achieve the same result at that point, but without forcing the user to find this not so obvious optimal concept.

B. Experiment 2: Evaluate the Benefit of Extra Threads

The user questionnaires indicated that 60% of the users preferred the CrossBrowser, citing that it was easier to use (42%), faster (32%) and that the RotorBrowser was too complex for them (16%), though the same users indicated that this might change if they have more time to learn from the system. About 35% of the users indicated their preference for the RotorBrowser, citing that this browser was better for more difficult topics (45%), and allowed for more interaction possibilities (55%).

The results of the user study in terms of AP show that performance is very dependent on the user. Per series of topics the MAP per user varies between 0.03 and 0.28, with the scores of the expert users far above that.

Looking at the number of unique shots found per topic, together with the originating threads gives us an insight into which threads are used most for which kinds of topic. Fig. 8(b) shows the number of retrieved shots per thread type for each topic. This figure shows that, though a large portion of the results found can be obtained by various threads, most of the shots were found using only one specific type of thread, with τ_q and τ_T being the threads used most. This indicates that each topic requires a different combination of threads to achieve the result; there is no single-best thread. When we compare RotorBrowser vs CrossBrowser performance in Fig. 9 we see that both browsers find different results, and that the CrossBrowser is generally able to find more results than the RotorBrowser. This is consistent with the experiences of the users themselves, who indicated that they found the RotorBrowser more difficult to work with. We found that the reason for the difference between results has has to do with the difference in browsing. The CrossBrowser allows a user to browse through large portions of a single thread quickly, without distractions from other visible threads. A RotorBrowser user selects results from this thread, and also from all localities

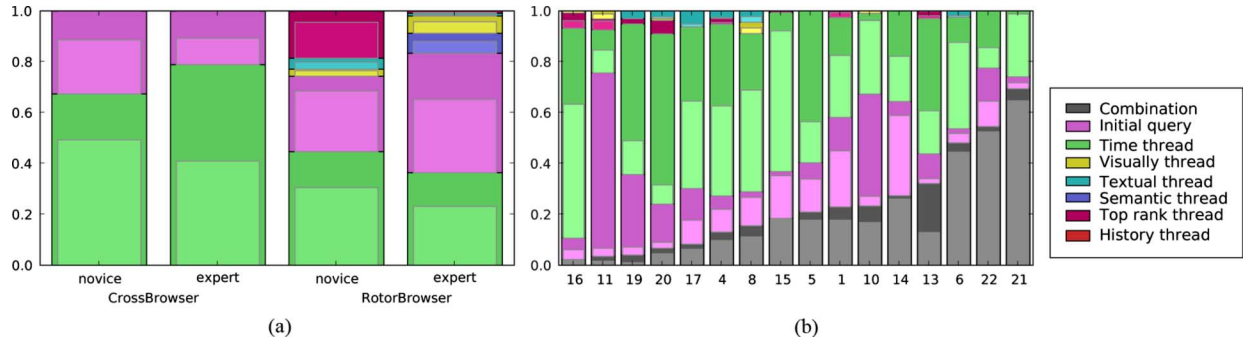

Fig. 8. These graphs give an overview of thread usage and relevance on a global and a detailed level. Each stacked bar shows the relative usage of each thread. The inner, lighter colored bar in each part denotes the relative percentage of results of that thread which was judged relevant. (Left) Difference in thread usage between expert and the group of novice users for both browsers. (Right) Results from experiment 2. Each stacked bar shows which threads have been used to gather the results for that topic, gray denotes that multiple threads where able to find the same results. (a) Average thread usage per user type. (b) Detailed thread usage per topic for the RotorBrowser.

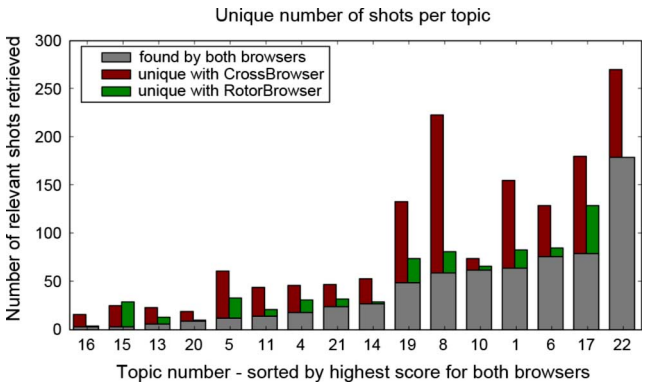

Fig. 9. Unique user evaluation results per browser, aggregated over all users. This graph shows that both browsers are able to both find a portion of the available relevant results, but that each individual browser also allows the user to find a unique set of results.

of visible relevant threads. As a consequence both browsers display different subsets of the dataset. This, in itself is interesting: browsing a single thread extensively will not result in the same results as found by browsing multiple threads.

C. Experiment 3: Evaluate the Efficiency of Threads

Overall our browsers perform well in the TRECVID evaluation. The CrossBrowser placed 2nd and the RotorBrowser placed 6th in the overall results. Due to the setup of the TRECVID experiment a different expert user is used for each browser, and this might explain the difference between results of the RotorBrowser and the CrossBrowser. As a consequence, we cannot compare the results directly. The experience and capabilities of each user influences the results. We can, however, deduce some interesting facts.

Fig. 8(a) shows which thread was used to select the individual results. The largest portion of results is generated from the initial query thread and the timeline. From the other threads the semantic concept thread and the visual thread were used most. The text thread was seldom used because these were only visible when the user explicitly performed a text search. This was done only in rare instances, because this required the users to switch back to the query phase, which in itself took too much time. Moreover, the users found that results obtained by textual retrieval in itself were of limited use. The top-rank threads are also rarely used by the expert users, even though they take up the most screen real estate. Hence, showing these threads is unwise as the user has to process all visible top-rank threads mentally even though they are probably less usable.

VI. Conclusion

This paper presented a method for browsing large collections of video using threads. Two types of threads are identified: static threads and dynamic threads. We have evaluated both the benefit of having these threads available during search, and the benefit of having static threads available in addition to dynamic threads. We have implemented both thread types in two browsers for video retrieval. We evaluate these browsers with a simulated user analysis, a user study, and participation in the TRECVID benchmark.

The overall evaluation results indicate that browsing through multiple threads is beneficial for video search. The user evaluation indicated that users had difficulty understanding and using the RotorBrowser when many threads were displayed. When fewer threads were shown video retrieval performance improved. The difference between expert users and novice users is also clear. The relative number of user interaction steps for both browsers remains stable for both the user study and the TRECVID experiment, however the expert users achieved a much higher average precision for all topics. This indicates that trained users are able to find the same number of results using less interaction with the search interface. Results also indicate that the version of the RotorBrowser as used at TRECVID showed many irrelevant threads. For example: top-rank threads were shown, which were not useful for search.

The experiments with simulated users indicate that additional static threads do improve search results for some topics, but that this is dependent on the topic. The same holds for the type of static thread used. This concurs with the results from the user study, which also show that the type of threads used are dependent on the topic, though the time and initial query threads are always beneficial, followed by visual similarity.

If we look at individual users we however also see that every user employs a different combination of threads to answer the same topic. All our results indicate there is no best thread.

Therefore, it is important to offer the user the opportunity to interactively select the threads deemed useful for a certain search task. By doing so, thread-based browsing provides intuitive and flexible video retrieval.

ACKNOWLEDGMENT

The authors would like to thank the 32 students for their participation in the user study, and C. Snoek for his help in many things.

REFERENCES

[1] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content based image retrieval at the end of the early years," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 12, pp. 1349–1380, Dec. 2000.

[2] A. Hauptmann, R. Yan, W.-H. Lin, M. Christel, and H. Wactlar, "Can high-level concepts fill the semantic gap in video retrieval? A case study with broadcast news," *IEEE Trans. Multimedia*, vol. 9, no. 5, pp. 958–966, Aug. 2007.

[3] M. Naphade and T. Huang, "A probabilistic framework for semantic video indexing, filtering, and retrieval," *IEEE Trans. Multimedia*, vol. 3, no. 1, pp. 141–151, Mar. 2001.

[4] A. P. Natsev, M. R. Naphade, and J. Tevšić, "Learning the semantics of multimedia queries and concepts from a small number of examples," in *Proc. 13th Annu. ACM Int. Conf. Multimedia*, New York, 2005, pp. 598–607.

[5] D. Wang, X. Liu, L. Luo, J. Li, and B. Zhang, "Video diver: Generic video indexing with diverse features," in *Proc. Int.Workshop Multimedia Information Retrieval*, New York, 2007, pp. 61–70.

[6] C. G. M. Snoek, M. Worring, J.-M. Geusebroek, D. C. Koelma, F. J. Seinstra, and A. W. M. Smeulders, "The semantic pathfinder: Using an authoring metaphor for generic multimedia indexing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1678–1689, Oct. 2006.

[7] M. Naphade, J. R. Smith, J. Tevšić, S.-F. Chang, W. Hsu, L. Kennedy, A. Hauptmann, and J. Curtis, "Large-scale concept ontology for multimedia," *IEEE Multimedia*, vol. 13, no. 3, pp. 86–91, Jul.–Sep. 2006.

[8] A. G. Hauptmann, W.-H. Lin, R. Yan, J. Yang, and M.-Y. Chen, "Extreme video retrieval: Joint maximization of human and computer performance," in *Proc. 14th Annu. ACM Int. Conf. Multimedia*, New York, 2006, pp. 385–394.

[9] H.-B. Luan, S.-Y. Neo, H.-K. Goh, Y.-D. Zhang, S.-X. Lin, and T.-S. Chua, "Segregated feedback with performance-based adaptive sampling for interactive news video retrieval," in *Proc. 15th Int. Conf. Multimedia*, New York, 2007, pp. 293–296.

[10] D. Heesch, P. Howarth, J. Magalhaes, A. May, M. Pickering, A. Yavlinksy, and S. Ruger, "Video retrieval using search and browsing," in *Proc. NIST TRECVID*, Gaithersburg, MD, 2004.

[11] M. Rautianen, T. Ojala, and T. Seppanen, "Cluster-temporal browsing of large news video databases," in *Proc. IEEE ICME*, Taipei, Taiwan, 2004, pp. 2:751–754.

[12] G. P. Nguyen and M. Worring, "Optimization of interactive visual similarity based search," in *Proc. ACM TOMCCAP*, Feb. 2008, vol. 4.

[13] L. Kennedy, S.-F. Chang, and A. Natsev, "Query-adaptive fusion for multimodal search," *Proc. IEEE*, vol. 96, no. 4, pp. 567–588, Apr. 2008.

[14] S.-Y. Neo, J. Zhao, M.-Y. Kan, and T.-S. Chua, H. Sundaram, Ed. *et al.*, "Video retrieval using high level features: Exploiting query matching and confidence-based weighting," in *Proc. 2006 Int. Conf. Content-Based Image and Video Retrieval (CIVR '06)*, Heidelberg, Germany, 2006, vol. 4071, Lecture Notes in Computer Science, pp. 143–152.

[15] K. McDonald and A. Smeaton, W.-K. Leow, M. S. Lew, T. Chua, W.-Y. Ma, L. Chaisorn, and E. M. Bakker, Eds., "A comparison of score, rank and probability-based fusion methods for video shot retrieval," in *Proc. 2005 Int. Conf. Content-Based Image and Video Retrieval of LNCS (CIVR '05)*, Heidelberg, Germany, 2005, vol. 3568/2006, pp. 61–70.

[16] W. Hsu, L. Kennedy, and S.-F. Chang, "Reranking methods for visual search," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 14–22, Jul.–Sep. 2007.

[17] J. Liu, W. Lai, X. Hua, Y. Huang, and S. Li, "Video search re-ranking via multi-graph propagation," in *Proc. ACM Int. Conf. Multimedia*, Augsburg, Germany, 2007, pp. 208–217.

[18] C. Chen, "Bridging the gap: The use of pathfinder networks in visual navigation," *J. Vis. Lang. Comput.*, vol. 9, no. 3, pp. 267–286, 1998.

[19] H. Luo, J. Fan, J. Yang, W. Ribarsky, and S. Satoh, "Analyzing large-scale news video databases to support knowledge visualization and intuitive retrieval," in *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST 2007)*, 2007, pp. 107–114.

[20] M. G. Christel, C. Huang, N. Moraveji, and N. Papernick, "Exploiting multiple modalities for interactive video retrieval," in *Proc. IEEE Int. Conf. Acoustic, Speech and Signal Processing*, Montreal, QC, Canada, 2004, vol. 3, pp. 1032–1035.

[21] J. Adcock, M. Cooper, A. Girgensohn, and L. Wilcox, "Interactive video search using multilevel indexing," in *Proc. ACM Int. Conf. Image and Video Retrieval*, 2005, vol. 3568, Lecture Notes in Computer Science, pp. 205–214.

[22] E. Zavesky, S.-F. Chang, and C.-C. Yang, "Visual islands: Intuitive browsing of visual search results," in *Proc. 2008 Int. Conf. Content-Based Image and Video Retrieval (CIVR '08)*, New York, 2008, pp. 617–626.

[23] T. Barecke, E. Kijak, Nurnberger, and M. Detyniecki, "Video navigation based on self-organizing maps," in *Proc. 2006 Int. Conf. Content-Based Image and Video Retrieval (CIVR '06)*, 2006, pp. 340–349.

[24] J. Philbin, O. Chum, J. Sivic, V. Ferrari, M. Marín-Jiménez, A. Bosch, N. Apostolof, and A. Zisserman, "Oxford trecvid 2007 notebook paper," in *Proc. NIST TRECVID Workshop 2007*, Nov. 2007.

[25] C. G. M. Snoek, M. Worring, D. C. Koelma, and A. W. M. Smeulders, "A learned lexicon-driven paradigm for interactive video retrieval," *IEEE Trans. Multimedia*, vol. 9, no. 2, pp. 280–292, Feb. 2007.

[26] A. F. Smeaton, P. Over, and W. Kraaij, "Evaluation campaigns and trecvid," in *Proc. 8th ACM Int. Workshop Multimedia Information Retrieval*, New York, 2006, pp. 321–330.

[27] A. G. Hauptmann and M. G. Christel, "Successful approaches in the trec video retrieval evaluations," in *Proc. 12th Annu. ACM Int. Conf. Multimedia*, New York, 2004, pp. 668–675.

[28] J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, C. G. M. Snoek, and A. W. M. Smeulders, "Robust scene categorization by learning image statistics in context," in *Proc. SLAM Workshop, in Conjunction With CVPR'06*, New York, Jun. 2006.

[29] J. R. Kender and M. R. Naphade, "Ontology design for video using semantic threads," in *Proc. IEEE ICME*, Amsterdam, The Netherlands, 2005.

Ork de Rooij is pursuing the Ph.D. degree in computer science at the University of Amsterdam, Amsterdam, The Netherlands.

His research interests cover information visualization, user computer interaction, and large-scale content-based video retrieval systems.

Marcel Worring (M'03) is an Associate Professor in computer science at the University of Amsterdam, Amsterdam, The Netherlands. His research interests are in interactive video retrieval, in particular the integration of computer vision, machine learning, and information visualization.