A Video Browsing Application based on visual MPEG-7 Descriptors and Self-Organising Maps

Horst Eidenberger
Vienna University of Technology, Interactive Media Systems Group
Vienna, 1040 Austria

Abstract

The paper introduces a novel approach for interactive video browsing that makes video content fully transparent to the user. Video clips are analysed and indexed by two tree structures: a content index tree representing the content of automatically segmented video shots and a time index tree representing the temporal structure. The index top levels give an overview over the entire content. Subsequent levels illustrate content relationships more detailed. Every level of both trees is a two-dimensional self-organising map organising media objects by two degrees of freedom. Media objects are represented by content-based visual MPEG-7 descriptions. The implemented navigation scheme allows the user for switching between content index tree and time index tree without loosing the overview. Context information (position in the tree, content of next lower level, etc.) is permanently shown in auxiliary panels. The implementation is based on the scalable vector graphics standard (visualisation) and the MPEG-7 reference implementation. First evaluation results show that the proposed approach facilitates accessing video content in a novel way.
Keywords : Video Browsing, Video Segmentation, Self-Organising Map, MPEG-7.

1. INTRODUCTION

This paper describes a novel video browsing approach that is based on a neural network clustering technique. Interactive video browsing aims at making video content transparently accessible. Application scenarios include editing, post production and metadata annotation. Generally, video browsing problems are investigated in visual information retrieval research (VIR) [14, 17, 2]. Like the majority of VIR approaches, our approach is based on media representation by visual descriptions (e.g. colour histograms, edge maps). We employ the visual MPEG-7 descriptors [16, 15, 1, 8] to index video content and make it accessible for browsing in a web-based user interface. Indexing is performed using self-organising maps (see Subsection 2.2) [10, 9].

In our approach, video data is hierarchically indexed by two criteria: by shot content and by time. For the content index tree, video streams are segmented into shots (using automatic shot boundary detection). Shots are represented by average descriptions and visualised by representative key-frames (see Subsection 3.3 for details). Indexing is performed on multiple levels: from an overview level (coarse selection of representatives from all shots) to multiple detail levels (fine selection of representatives from similar shots). This is similarly true for time index tree. The difference is that for the time index tree, frames are selected at certain time intervals. On each level every *n*-th frame is used for indexing. *n*, the step width, is set to a large value for the overview level and to smaller values for the detail levels (see Subsection 3.2 for details). Hence, the time index tree represents a content-independent top-down view on video data while the content index is constructed bottom-up based on shot boundaries. Since content index tree and time index tree are based on the same data, the user is enabled to switch between the two views at any time during the browsing process.

Our video browsing approach differs from related approaches in the point that it employs both browsing and retrieval techniques: Visual descriptors are used to identify shot boundaries and to describe media objects. A similarity-based clustering algorithm is employed to cluster video segments. Similar video frames are located close to each other. Since we use a two-dimensional clustering technique, two degrees of freedom are available for clustering. Content-based and time-based selection and similarity-based clustering in hierarchically organised index trees result in a structured transparent view of video data. Technically, a major novelty is that the implementation is exclusively based on free software. For example, the user interfaces are based on the scalable vector graphics standard (SVG) [21]. Description extraction is based on the free reference implementation of the MPEG-7 standard. Shot detection is based on state-of-the-art VIR procedures.

The paper is organised as follows. Section 2 sketches relevant related work including the visual MPEG-7 descriptors, the clustering technique used, automatic video segmentation and recent video browsing approaches. Section 3 describes idea and design of the video browsing application and the implemented navigation paradigm. Section 4 deals with implementation issues: descriptor selection for video segmentation, description clustering and user interface implementation. Finally, Section 5 presents experimental evaluation results.

2. RELATED WORK

2.1. Visual MPEG-7 descriptors

In the video browsing application, we use visual MPEG-7 descriptors for media description and video segmentation. The

Corresponding Author: Horst Eidenberger is with the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Favoritenstrasse 9-11, Vienna, 1040, Austria. FAX: +43-58801-18898
Email: eidenberger@ims.tuwien.ac.at

visual part of the MPEG-7 standard defines several descriptors [16, 15, 1, 8]. Not all of them are actually descriptors in the sense that they extract properties of media content. Some of them are just structures for descriptor aggregation and localisation. The basic colour descriptors are *Color Layout* (first DCT coefficients of YCrCb averages of major image/frame regions), *Color Structure* (histogram of colour usage in colour regions), *Dominant Color* (colour value and percentage of eight most used colours) and *Scalable Color* (classic, scalable colour histogram). Texture descriptors are *Edge Histogram* (edge orientation histograms for 4x4 sub-regions), *Homogeneous Texture* (energy values and distributions for 40 Gabor filters) and *Texture Browsing* (average coarseness and directionality of textures). Shape descriptors are *Region-based Shape* (35 ART coefficients for Y channel) and *Contour-based Shape* (contour descriptions of segmented objects). Motion descriptors are *Camera Motion* (based on optical flow), *Parametric Motion* (motion of predefined objects) and *Motion Activity* (motion vector-based frame by frame motion).

Other descriptors are based on these low-level descriptors or on additional semantic information: *Group-of-Frames*/*Group-of-Pictures* (aggregation of *Scalable Color* descriptions), *Shape 3D* (based on 3D mesh information), *Motion Trajectory* (based on object segmentation) and *Face Recognition* (major face parameters like eye to eye distance, etc.; based on face extraction). Finally, supplementary (textual) structures exist for colour spaces, colour quantisation and multiple 2D views of 3D objects. Since our application is dealing with individual key-frames, only the listed colour, texture and shape descriptors are considered below.

2.2. Self-organising maps

The self-organising map (SOM) [10, 9, 11] is a two-layer fully connected neural network that uses feed-forward learning. SOMs are mainly intended for clustering of high-dimensional data (see [7] for a survey). The input layer is interpreted as a one-dimensional data vector. The output layer is interpreted as a two-dimensional map of clusters. The clusters of the output map may have rectangular or hexagonal shape. Each cluster of the output map is described by a weight vector pointing to its center (codebook vector). In training and application, input data vectors are mapped to the codebook vector with minimum Euclidean distance (best matching unit, BMU). SOM learning is based on a predefined map size and randomly selected codebook vectors. The map is adapted by iteratively applying input vectors, selecting the codebook vector with minimum distance and changing its location by a fraction of the distance (weighted by learning rate α).

One major innovation of SOMs over other clustering methods is the introduction of neighbourhood kernels. These two-dimensional functions define the fraction, to which the BMU is adapted but also, to which extent neighbouring codebook vectors are adapted. Thus, SOM learning means learning of cluster neighbourhoods. A typical neighbourhood kernel is the two-dimensional Gaussian density function. Using neighbourhood kernels results in somewhat 'natural' cluster structures that intuitively fit with humans' similarity perception. This property is the major reason why we are using SOMs for clustering in the video browsing application.

The tree-structured SOM [12] is a further developed SOM that allows for constructing hierarchical cluster trees. Tree-structured SOMs are related to our approach. The major difference is that tree-structured SOMs cluster the entire data on every level while in our approach every SOM consists only of a small, carefully selected fraction of the entire data (video frames). Hence, it would not have been possible to achieve the effect desired by the proposed video browsing application by using tree-structured SOMs.

2.3. Temporal video segmentation

Automatic temporal video segmentation aims at identifying shot boundaries in video streams without user involvement. In recent years, a significant number of approaches have been proposed [2]. Today, state-of-the-art automatic video segmentation procedures identify more than 90% of all transitions (including fades and wipes) in video streams at minimal numbers of false positive detections. Generally, shot transitions can be distinguished in sharp cuts and effect transitions (fades and wipes). Sharp cuts are, for example, used in news videos. Effect transitions are regularly used in sports programs.

Methods for detection of sharp cuts are either based on uncompressed media data or compressed media data. The simplest approach that uses uncompressed data is the frame difference approach: Consecutive video frames are spatially pixel-wise compared. If the sum of differences exceeds a certain predefined threshold, a cut is assumed. This approach is easy to implement but has several drawbacks: it is computation power-demanding, not robust against global changes in the video data (e.g. changed lighting conditions) and sensitive for camera movement (e.g. zooming, panning, etc.). More sophisticated approaches use visual features to summarise frames. Examples are colour histograms (global features) or edge maps (local features). Shot boundaries are assumed where the distance of feature vectors exceeds a threshold. Obviously, feature-based approaches do not suffer from lacking robustness against photographing conditions and camera movement. Furthermore, if features can be computed in advance, the cut detection process is less computation power-demanding than the frame difference approach. Recently, since the visual part of the MPEG-7 standard for multimedia content description has been released, more and more feature-based approaches employ MPEG-7 descriptors for cut detection (e.g. *Scalable Color* in [5]). In Subsection 4.2 we try to identify the best MPEG-7 descriptors for cut detection.

Most methods for sharp cut detection that are based on compressed media data make use of motion vectors (e.g. [24]). If the optical flow changes significantly from frame to frame (again, significance implemented by a threshold), a shot boundary is assumed. The major advantage of compressed data-based approaches is that they require less computation power than approaches working on the uncompressed domain. Methods for detection of effects are usually based on feature-based approaches. Twin-comparison [23] employs two thresholds: All inter-frame distances exceeding a first threshold are summed up. If the sum exceeds the threshold for sharp cuts, an effect transitions is assumed. This approach works

excellently for gradual transitions as fades and wipes. The production model-based approach [6] analyses effects top-down. Models for location (wipes) and intensity (fades) changes are derived. Frame sequences fitting to the models are assumed being effect transitions.

2.4. Video visualisation for browsing

The crucial user interface issue that has to be solved in video browsing systems is the visualisation of the temporal dimension of video. The spatial content of video changes over time. Since the view does not, there is no 'natural' way to visualise video content entirely on the spatial domain. In general, there are three solutions to present video information. Firstly, integration of the full video with player controls into the environment. This approach is CPU power- and network bandwidth-consuming. Secondly, creation and usage of animated iconic structures. Even though being less demanding in terms of network bandwidth, this approach is still computation power-consuming. Thirdly, creation of two- or three-dimensional models that represent the video content.

Examples for animated iconic structures are the hierarchical video magnifier [2] and the scene transition graph [22]. The approach followed by the hierarchical video magnifier is similar to the time index tree proposed in this paper. It provides a simple hierarchical structure of key-frames: Key-frames selected from the entire video content are shown on the top level. On subsequent levels, key-frames selected from parts of the video (but at smaller intervals) are shown. Layers are simply rows of key-frames ordered by time. The user can select detail views by clicking on key-frames on higher levels. Scene transition graphs give a graph representation of video content: Shots with similar content are clustered in nodes. Nodes are connected by arcs depending on their temporal relationships.

Model-based representation is the most widely applied video visualisation method. In the simplest form an image matrix of all key-frames in a video clip is used. A more sophisticated approach is the Micon [4], an image showing the first frame of a video clip as well as the first line and the last column of all consecutive frames in a cube-like view. Micons are easy to compute and give good indication on video motion for many types of content. The main shortcoming of Micons is that perspective cannot be changed easily. The video X-ray approach provides a fully three-dimensional model of video. Video X-rays are visualised as Micons but, since the model is three-dimensional, perspective can be adapted arbitrarily. Furthermore, video X-rays allow for editing of video clips (e.g. spatio-temporal cutting, compression, etc.). Another approach from a similar direction as the Micon is mosaic visualisation [2]. In a mosaic visualisation, the frames of a video clip are glued together to a panorama-like view. Stitching is based on object motion. See [4, 2] for comprehensive introductions to these and other video visualisation techniques.

3. VIDEO BROWSING APPLICATION

This section describes the design of the proposed video browsing application. Subsection 3.1 illustrates the novel ideas implemented in the approach. Subsections 3.2 and 3.3 describe, how the two types of browsing criteria (time and content) are used. Subsection 3.4 sketches navigation in the browsing process and switching between time index tree and content index tree.

3.1. Idea and motivation

In our video browsing application, video streams are organised in tree structures. The top level gives an overview over the entire video content. Subsequent levels show detail information (on groups of shots, shots, temporal fractions of the video stream). Leaves of the tree are shots (content index tree) or single frames (time index tree), respectively. The user browses through the tree structures from top to bottom. Selecting a cluster from a map causes him stepping one level down in the index tree and seeing more details on the selected fraction of the video stream. The route taken through the index tree is visualised in the user interface by auxiliary panels (see Subsection 3.4). Generally, this paradigm is similar to the hierarchical video browser (as described in [4]). Two aspects are responsible for making video perception through the proposed video browsing application a completely new experience:

- Two organisation criteria are used: time and content
- Tree layers are maps of elements clustered by content similarity

Mostly, existing video browsing approaches offer only a single view of video: a temporal view of key-frames selected at predefined intervals (independently of the content) or a content-based view of selected representative frames. In the author's opinion this is unsatisfactory, since many applications require having both types of index available simultaneously. For example, in video archival and browsing-based retrieval, the user might – depending on event characteristics – in some cases remember the time, when something happened and in other cases, in which context something happened: Regular viewers of soccer matches can easily remember *when* a goal was shot, if it was scored in overtime and decided the match, but hardly when a free kick was executed that did not have a major impact on the game. On the other hand, the free kick can easily be remembered, if it was the result of a brutal foul by a hated defender on a beloved star of the preferred team.

Our video browser offers both views in independently organised index structures. Key-frames for time-based and content-based indexing are selected independently and clustered hierarchically using the same procedure (see Subsection 4.3). Additionally, a matching procedure is provided that allows for switching between the two views. Subsections 3.2 and 3.3 describe the two index types and Subsection 3.4 sketches the matching procedure.

The second innovation in the proposed video browsing application is making use of content-based visual information retrieval for layer organisation to support the user's visual similarity perception. Key-frames are described by content-based visual MPEG-7 descriptors (see Subsection 2.1). These media descriptions (technically, high-dimensional data vectors) are clustered using self-organising maps (SOMs, see Subsection 2.2). The result is a two-dimensional map of clusters, in which similar media objects are located closely to each other. Since we use MPEG-7 descriptors, similarity is defined on the basis of generally perceived (un-recognised)

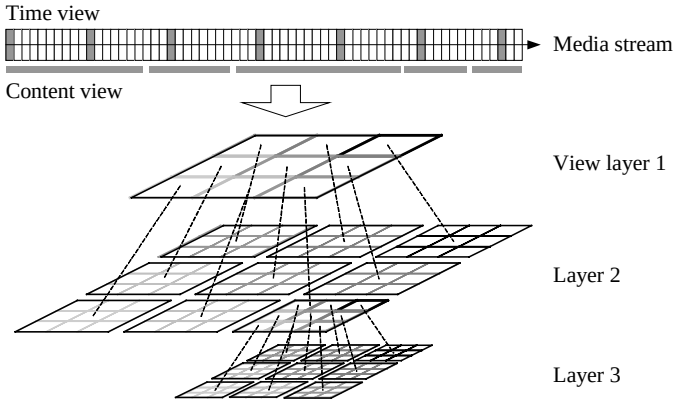

Figure 1. Video index trees. Every index tree consists of multiple layers. The number of layers depends on the media stream size. Indexes are constructed from the temporal video view (by selecting every *n*-th frame) and the content view (by selecting representative frames of shots).

image properties (e.g. colour distributions). The major advantage of this approach is that it supports human visual similarity perception. Similarity-based clustered key-frame images allow the user to judge the content of a particular layer more quickly and to uncover implicit similarities in the content of video streams. This allows, for example, to understand colour codes applied in advertisements better (e.g. bright colours for product properties that should bet perceived positively, etc.).

We use two-dimensional clustering in the video browser, because it supports human spatial perception. Additionally, it offers an additional degree of freedom in comparison to hierarchical clustering. Furthermore, maps can easily be illustrated in any type of user interface. In the past, we have also experimented with three-dimensional clustering based on Sammon mapping [19] and visualisation in virtual worlds using VRML [20]. We found that three-dimensional maps are more difficult to understand and navigation, overlapping and clipping can soon become confusing for the user. Therefore, we decided on two-dimensional clustering. SOMs were selected, because – as pointed out in Subsection 2.2 – by employing neighbourhood kernels for learning they provide a human perception-like cluster structure.

Figure 1 illustrates the resulting type of index: Layers are derived from the video stream by time and content criteria. Every layer is a two-dimensional map clustering elements by content-based features (e.g. colour, structure). Top levels give overview information. Subsequent layers give detail information. The entire video browsing application comprises two independently organised index structures inter-connected on the frame level. See Figures 8, 9 for examples.

3.2. Time index tree

For the time index tree, key-frames are selected from the video stream in a way that preserves the temporal order. Even though content-based access is an important issue in visual information browsing, the temporal structure must not be neglected. Humans have an excellent memory for the temporal order of events. The time index tree is responsible for providing a hierarchical temporal view on the media.

Key-frames are selected as follows (see Figure 1 for illustration): Every *n*-th frame of the video stream is selected. *n* (the step width) depends on *l*, the layer number (starting with '1' (top layer)). *n(l)* is defined by equation 1. Map dimensions are given by *r* (rows), *c* (columns). *round_up(X)* replaces *X* with the next higher cardinal number.

$$n(l) = round\_up\left(\frac{length(video\ clip)}{(rows.columns)^l}\right) \quad (1)$$

Thus, the step width for a map on a particular layer *l* decreases proportionally to the position of the layer in the time index tree. At most, one frame per map entry (cluster) is selected. Maps on layers below the top level are mapped to clusters on the next higher level by an offset function *o(x, y)* (see equation 2): The offset function defines the starting offset for key-frame selection from the video stream.

$$o(x,y) = (y.r.c^2 + x.r.c).n(l) + O_{l-1} \quad (2)$$

x, y (cardinal numbers starting with zero) identify a cluster on layer *l-1* (that is elaborated on level *l*). O_{l-1} is the offset of the map on layer *l-1*. (Remark on navigation: It is important to notice that, since temporal order is lost in the content-based map clustering process, the pair *<x, y>* does not simply identify the *(y*rows+x)*-th cluster of the map on layer *l-1*. The corresponding cluster has to be located by establishing a link from map elements on layer *l* back to map elements on layer *l-1* using the input video stream.)

In conclusion, the content of maps of the time index tree is determined by *<n(l), o(x, y)>* pairs. On the top level, just one map exists. On subsequent levels, exactly one map exists for every cluster on the preceding level. Consequently, the time index tree is always a balanced tree. Leaves are single frames. If *rows=columns=2*, the time index tree is a quad-tree structured by visual content.

3.3. Content index tree

The content index tree is an iconic shot index. While the time index tree is constructed top-down, the content index tree is built bottom-up based on shots. Shot boundaries are detected using automatic video segmentation (see Subsections 2.3, 4.2). Even though automatic shot detection does not provide full accuracy, it is sufficiently good for our purpose.

In the indexing process, shots are represented by average media descriptions. Media descriptions are extracted from frames using the content-based visual MPEG-7 descriptors. These descriptions are averaged for the relatively coherent content of single shots. Generally, using a simple mean should be sufficient as an averaging method. The averaged descriptions are clustered using self-organising maps. In contrast to the time index tree, where only a fraction of frames are employed for clustering, all averaged descriptions are considered for clustering on the top level. Then, in a recursive process all clusters containing a number of elements that exceeds a predefined threshold are clustered again and mapped to clusters on the next higher level as detail levels. For practical reasons the threshold should be set larger than map size. Smaller threshold values would result in unnecessarily deep index structures.

Shots are the leaves of the content index tree. Since it is not

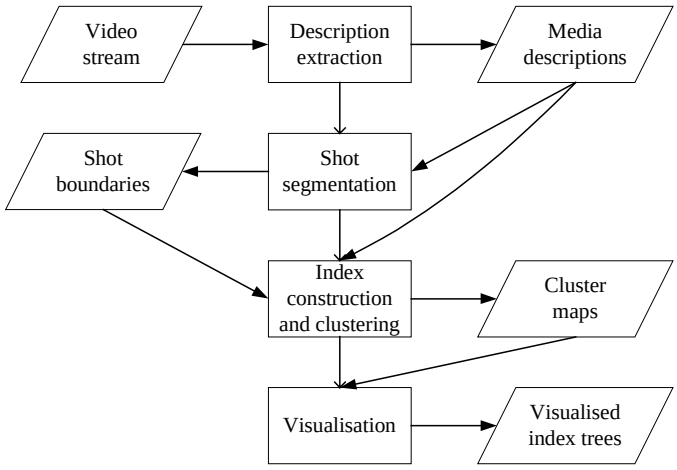

Figure 2. Workflow and data flow in the data preparation process of the video browsing application.

predictable, which content-based relations exist among shots, the content index tree is generally unbalanced containing deep, highly differentiated structures for frequently appearing content (e.g. shots of leading actors in movies) and less differentiated structures for less frequently appearing content (e.g. extras). This is desired by the approach as it supplements the context-free view provided by the time index tree elegantly.

The major design issue connected to the content index tree is selecting representative media objects for map clusters. Since, in contrast to the time index tree, clustered media descriptions are artificial, we cannot simply employ the cluster medians as representatives. A two-step procedure is required: Firstly, we identify the median average description vector (the one with minimum Euclidean distance to the codebook vector; see Subsection 2.2). Then, we identify the frame with the most similar description to the average vector (again, by Euclidean distance). This frame is selected as cluster representative and visualised in the map.

3.4. Tree matching and navigation

Above, it was mentioned that the video browsing application allows the user to switch from content index tree to time index tree and back during browsing. The implementation of this feature requires matching between the index trees. Starting from a selected cluster in one index tree, two parameters have to be determined for switching: map cluster correspondence and layer correspondence. A content index cluster and a time index cluster are defined as corresponding, if they use the same representative media object for cluster visualisation:

$$Median_{\text{content index tree cluster}} \equiv Best_{\text{time index tree cluster}} \quad (3)$$

This is a one-to-many relationship: multiple clusters in the second tree may correspond to the selected cluster. In order to reduce the number of candidates to one, we use layer correspondence: The cluster is selected as switching target that is located on the layer with minimum hierarchical distance to the switching source. Formally:

$$\text{select } m_t \text{ with } d\left(\frac{layer(m_s)}{|source\ tree|}, \frac{layer(m_t)}{|target\ tree|}\right) \rightarrow min \quad (4)$$

where m_s, m_t are the maps in source and target index trees that contain the corresponding cluster representatives, *layer(X)* gives the layer number of map *X*, $|X|$ is the span (number of layers) of index tree *X* and *d()* measures Euclidean distance. These two conditions define a unique mapping between content index tree and time index tree.

However, there is still one problem that needs to be solved. The time index tree contains all frames of an indexed video clip. Therefore, for any cluster representative in the content index tree it is possible to identify a corresponding cluster element in the time index tree. The other way around, this is not the case: In the content index tree entire shots are represented by a single frame. We suggest the following solution to overcome this problem: If, for a particular switching source in the time index tree, no corresponding frame exists in the content index tree, then the leaf map and cluster are chosen as switching target, that refer to the video shot containing the corresponding frame. In this case, the layer condition cannot be satisfied. To avoid a confusing effect on the user, she is notified by a system message.

Generally, browsing through and switching between trees can easily become confusing. We have implemented several user interface components to avoid such an effect. These components will be described in detail in Subsection 4.4. The major guidelines are: Trees are never shown entirely (information overload). Instead, we display the active layer, the preceding layer (with the selected cluster highlighted) and a preview of the next layer that corresponds to the cluster that is highlighted in the active layer. Moreover, in an additional panel the corresponding map in the non-active index tree is shown. Besides avoidance of information overload this scheme has the advantage that it can be implemented without complex and resource-consuming three-dimensional tree visualisations.

4. IMPLEMENTATION

Below, we describe relevant implementation issues of the video browsing application. Subsection 4.1 gives an overview over workflow and data flow in the index preparation process. Subsection 4.2 describes descriptor selection for automatic shot boundary detection. Subsection 4.3 gives details on the clustering process used. Finally, Subsection 4.4 sketches important visualisation and user interface aspects.

4.1. Overview

Figure 2 illustrates the index tree preparation workflow in the video browsing application. Starting from the input video stream, media descriptions are extracted. We apply the MPEG-7 image descriptors and describe each frame of a video clip by colour content, textures and general shape properties. Subsection 5.1 gives detailed information on the descriptors and parameters used. The media descriptions are the input for the automatic shot segmentation procedure. It employs description-based comparison (for sharp cuts) and twin comparison (for fades and wipes) on optimised MPEG-7 descriptions to identify shot boundaries (see Subsection 4.2 for details).

Shot boundary information and visual descriptions are fed into the index tree construction and clustering process. In the first step, averaged shot descriptions are computed. Then,

Figure 3. Example frames from advertisement, cartoon, documentary, movie and news clips employed in the evaluation (captured from German satellite television).

independently for time index tree and content index tree, frames are selected and the top views of both indexes are computed using self-organising maps (see Subsection 4.3 for details and Subsection 5.1 for parameters). Based on the top view clustering, SOM calculation is recursively repeated for the content index tree. The time index tree is computed by top-down selection of step widths and offsets (as described in Subsection 3.2). The resulting index trees are stored in a simple XML format that marks the endpoint of the pre-processing steps.

The XML document describing the two index trees is used as input for the visualisation process. Visualisation target is the web browser. Hence, we employ web-based standards for visualisation. Specifically, index tree components are visualised by scalable vector graphics (SVG). Visualisation is supplemented by event-based interaction: SVG supports ECMAScript, which is used for handling user requests. The visualisation part of the video browsing application is described in detail in Subsection 4.4.

4.2. Shot boundary detection

In the video browser we require a procedure for shot boundary detection. We use description-based cut detection in combination with the twin comparison approach for effect detection (see Subsection 2.3). Since we use visual MPEG-7 descriptors for media description, we want to use the same descriptions for cut detection. Below, we aim at identifying the best application domain-independent MPEG-7 description scheme for automatic cut detection. Optimising the performance of shot boundary detection is crucial for the quality of the video browser index trees. To reach this goal we apply the majority of content-based visual MPEG-7 descriptors on video clips of varying content and compare the results of automatic detection to ground truth information provided by human users.

Experimental Setup: We split the process of identifying the best description scheme for MPEG-7-based cut detection into two steps: First, we compute the individual performance of descriptors. Then, we try to identify combinations of descriptors that improve the individual results. This section describes the media sets used for evaluation, the visual descriptors we apply and the methods we apply for cut detection (including threshold optimisation), performance evaluation, descriptor combination and ranking.

The test data comprises media clips from five different genres: advertisements, cartoons, documentaries, movies and news (see also Subsection 5.1). Figure 3 shows example frames. The genres differ widely in cut rate and transition types used. In advertisements clips cuts occur after at least 2,5 percent of frames, in cartoons after about one percent and in documentaries, films and news after less than 0,5 percent. News programs and advertisements mostly apply sharp cuts while cartoons and documentaries often use transitions (mainly, fades) over twenty or more frames.

We apply the visual MPEG-7 descriptors (using the eXperimentation Model version 5.6) on all frames of the test videos. All colour descriptors are used: *Color Layout*, *Color Structure*, *Dominant Color*, *Scalable Color*, two texture descriptors: *Edge Histogram*, *Homogeneous Texture*, and the *Region-based Shape* descriptor. All descriptors are applied with maximum resolution. The feature vector elements are normalised to the interval [0, 1]. In total, every frame is described by a feature vector of 306 elements. For dissimilarity measurement we employ the distance measures and parameters suggested by the MPEG-7 authors (mostly, city block distance without weights).

For cut detection we define a threshold t_b (individually for each descriptor). Additionally, we use the twin comparison approach [2] to detect fades and wipes: A second threshold t_s is defined for gradual changes ($t_s << t_b$). Two indicators are computed to evaluate the performance of a descriptor: the number of correct hits and the number of false positives. Calculation of indicators is based on ground truth information provided by test users. Since we want to measure the best possible performance for each descriptor it is crucial guaranteeing that no descriptor is discriminated by false threshold values. Therefore, the thresholds are iteratively optimised in an automated procedure

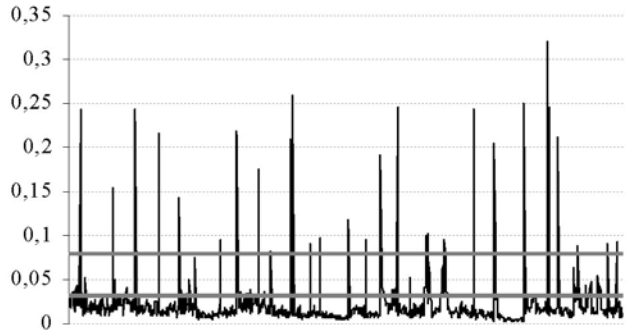

Figure 4. Frame difference signature of *Color Structure* descriptor applied to advertisement clips (X axis: time, Y axis: distance, grey lines: thresholds).

based on ground truth information. This optimisation is performed per genre. Then, the best identified thresholds are used to compute the hits and false positives indicators.

In this process we have to deal with two optimisation criteria. As the numbers of *hits* (correct / present) and *false positives* (false / present) cannot be easily combined, it is almost impossible to define a single goal function. Giving preference to one of them depends heavily on the considered type of application. Hence, we decided to base the ranking procedure on a superiority principle: One descriptor is considered being superior to another if it leads to better results for one indicator (more hits, less false positives) while being at least as good for the other indicator. Two descriptors, for which superiority cannot be clearly identified, are given the same rank (independently of the size of the performance gaps).

Based on the ranking of individual MPEG-7 descriptors we aim at identifying the best overall description scheme by combining descriptors using logical operators. Generally, cut detection results of two descriptors can be combined in two ways: Either, all cuts are assumed correct that are detected by both descriptors (*AND*) or all cuts that are detected by one of them (*OR*). An *AND* combination of two descriptors reduces the hits indicator to the value of the worse descriptor. The false positives indicator is reduced to a value in the interval [0, $min(FP_1, FP_2)$] where FP_1 and FP_2 are the false positive indicators for the first and second descriptor, respectively. In the best case, all false positives are eliminated. If two descriptors are combined by the *OR* operator the number of hits equals the hits indicator of the better descriptor. The false positives indicator becomes a value of [$max(FP_1, FP_2)$, $FP_1 + FP_2$]. In the worst case, all false positives are part of the combined analysis. Obviously, *OR*-combined descriptors can never be superior to the involved descriptor with the higher correct hits rate. In consequence, the *OR* operator is not further considered in this study.

Results: In the first step the performance of individual MPEG-7 descriptors is analysed. For example, Figure 4 shows the distance signature of the *Color Structure* descriptor over time (frames). This feature is highly discriminant for sharp cuts and leaves enough space between cuts, fades and wipes, and object and camera movement to define the thresholds for twin comparison clearly. Actually, *Color Structure* showed the best performance of all evaluated MPEG-7 descriptors.

Descriptor	t_b	t_s	*Hits*	*FP*	*Rank*
Color Layout	0,465	0,027	95,4%	6,5%	1
Color Structure	0,096	0,036	97,2%	10,2%	1
Dominant Color	0,429	0,230	57,4%	61,1%	4
Edge Histogram	0,191	0,071	85,2%	1,9%	1
Homog. Texture	0,074	0,015	76,9%	5,6%	1
Region-based Shape	0,173	0,022	87,0%	15,7%	2
Scalable Color	0,078	0,015	68,5%	19,4%	3

Table 1. Shot detection thresholds and performance indicators for visual MPEG-7 descriptors.

Description Scheme	*Hits*	*FP*
Color Structure	97,2%	10,2%
Color Layout, Color Structure	95,4%	1,9%
Col. Layout, Col. Struct., Edge Hist.	85,2%	0,0%

Table 2. Shot boundary detection performance of best MPEG-7 description schemes.

Table 1 summarises optimal threshold values and performance indicators for all descriptors. *Color Structure* and *Color Layout* retrieve most cuts correctly, while the texture features *Edge Histogram* and *Homogeneous Texture* minimise the number of false positives. This may be the case because edge information is more robust against camera operation than colour information. Characteristics of descriptors and distance measures can be seen from the threshold values. For some descriptors (especially, *Scalable Color*) it is highly difficult to set the threshold for gradual transitions. In consequence, the hit rate is significantly less than for the best descriptors. The first rank (in terms of superiority) is shared between *Color Layout*, *Color Structure*, *Edge Histogram* and *Homogeneous Texture*. Only these descriptors were considered for combination.

Computing the performance for all *AND*-combined description schemes reveals three description schemes being superior over all others (see Table 2). The highest hit rate is achieved by the *Color Structure* descriptor alone. Using *Color Layout* and *Color Structure* in combination leads to a high hit rate and few false positives. If *Color Structure* is used in combination with *Color Layout* and *Edge Histogram*, the number of false positives drops to zero. This description scheme may be considered optimal for most applications. Consequently, it is used for shot boundary detection in the video browsing application.

4.3. Description clustering

The data clustering procedure of the video browsing application is responsible for visual similarity-based organisation of index tree layers. It takes its input from the key-frame selection procedure (as described in Subsections 3.2, 3.3). Key-frames are described by visual MPEG-7 descriptions, i.e., basically, high dimensional vectors of floating point numbers (in our case normalised to interval [0, 1]). The descriptions of key-frames are clustered by self-organising maps. Subsection 2.2 describes the learning process in self-organising maps and their specific advantages. SOMs have been used in visual information retrieval and browsing before: The PicSOM system of the Helsinki University of Technology

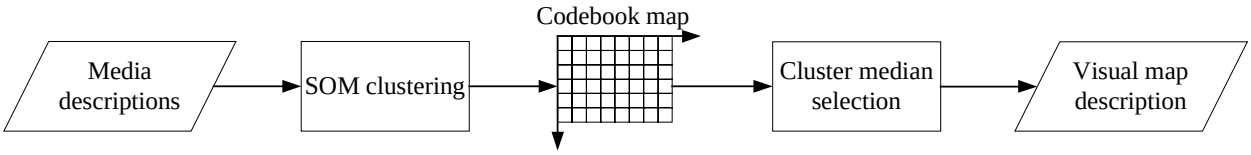

Figure 5. Workflow in MPEG-7 description clustering process. Every layer of the time index tree and the content index tree is clustered based on visual similarity criteria.

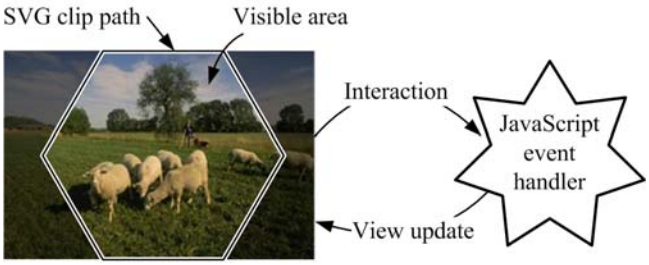

Figure 6. The SVG cell is the basic building block of the video browser user interface. A clip path is used to create the hexagonal shape. Event handling is implemented using W3C DOM event types and JavaScript listener procedures.

[13] is a successful content-based image retrieval system that employs SOMs for data clustering and incorporates iterative refinement by relevance feedback [18] into the retrieval process (based on tree-structured SOMs; see Subsection 2.2).

Self-organising maps offer two degrees of freedom for similarity-based media organisation. The SOM decides implicitly, which properties it selects for spatial organisation. Generally, the description elements with the highest variance have the most significant influence on the cluster structure. In visual information retrieval usually the strongest stimuli are colour and structure (textures, shapes) properties. Therefore, it is likely that SOMs trained from key-frames described by MPEG-7 descriptors are spatially organised by colour and structure appearance.

Figure 5 illustrates the workflow in the clustering process. Media descriptions are repeatedly fed into the SOM learning process. The output map has a predefined size. Every cluster is described by a vector pointing to the cluster center (so-called codebook vector). The codebook vectors are adapted in the learning process until the quantisation error is minimal. To compute the quantisation error, every input vector is fed into the SOM once and mapped to the codebook vector that has minimum Euclidean distance (best matching unit, BMU). The sum of distances over all vectors (normalised by the number of input vectors) defines the quantisation error: the average displacement, if input vectors would be replaced by their BMUs.

The set of codebook vectors completely defines a SOM but it does not explicitly express, to which clusters input vectors belong. Identifying the cluster structure requires locating the BMU for every input vector in an additional iteration. In some cases multiple input vectors are mapped to the same BMU and other BMUs are not associated with any input vectors. In our application, this behaviour is acceptable for the content index tree: similar shots are clustered together. Holes in the map may exist. See Figure 9 for examples. It is not acceptable for the time index tree. In the time index tree every cluster should consist of exactly one frame (time interval) that is detailed by a map on the subsequent layer. See Figure 8 for an example. To implement such a behaviour based on SOMs, we require an algorithm that identifies the best combination (e.g. in terms of quantisation error) of input vectors and codebook vectors. Since, generally this is a problem of order $O(n)=n!$, we use a simple heuristics to identify a sufficiently good *1:1* association of input and codebook vectors: For every randomly chosen codebook vector (map entry) we identify the best matching input vector (frame). Then, this input vector is removed and the procedure is repeated until all codebook vectors are mapped to input vectors. Experimental results show that this mapping procedure generates acceptable results.

After finished SOM training and identification of the BMU for every input key-frame, cluster coordinates and frame IDs of the key-frames representing clusters (see Subsections 3.2, 3.3 for the selection procedures) are stored in a simple XML document. The XML descriptions are used in the visualisation process described in the next subsection.

4.4. User interface design

User interface design for the video browsing application comprises two activities: visualisation of index tree layers and visualisation of the navigation system. As described in Subsection 3.4, we decided not to visualise entire index trees. Instead, the user interface displays the active map layer, the preceding layer and a preview of the subsequent layer (for the active cluster).

The basic building block of each layer is the cluster cell. Figure 6 describes its shape and functionality. Since we are using self-organising maps with hexagonal layout (every non-border codebook vector has six neighbours), the cluster cell is also of hexagonal shape. The cell is implemented by a scalable vector graphics (SVG) document. Every layer map consists of one cell per cluster. Hence, every map is a collection of SVGs that can easily be displayed and manipulated in a web browser window.

The SVG cell is based on the key-frame representing a cluster. A polygon of hexagonal shape is laid over this image. A copy of this hexagon is used as a clip-path to cut off those parts of the image that should not be visible in the cluster map. The resulting image is associated with an ECMAScript event listener for handling of mouse events. If the mouse cursor is moved over the cell, a listener method changes the border colour and triggers a user-defined event handler. This event handler can, for example, be responsible for displaying the preview of the map on the next lower level. The entire user interface of the video browsing application is based on this simple active SVG cell.

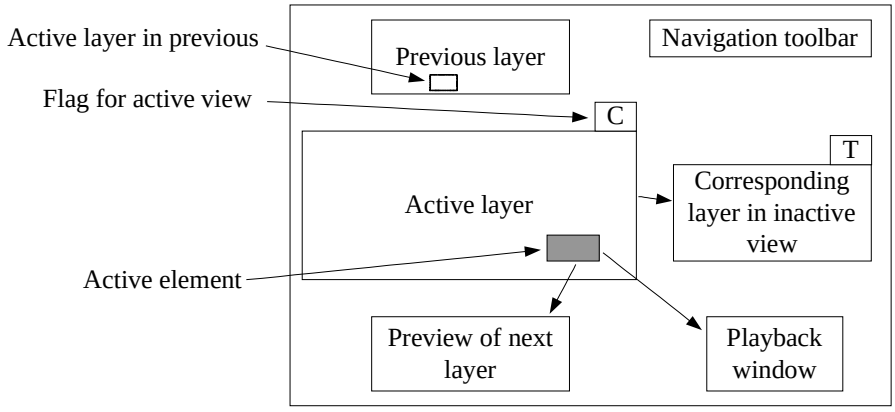

Figure 7. Navigation layout of the video browser user interface (see Subsection 4.4 for details).

Figure 7 illustrates the user interface layout. Central element is the selected layer of the active index tree ("C" for content index tree, "T" for time index tree). The selected cluster is shown highlighted. Above the active layer a smaller panel shows the next higher layer. In this window, the cluster is highlighted that is associated with the active layer. A window below the active layer shows a preview of the layer associated with the selected cluster in the active layer. If the active layer points to a leaf of the index tree (key-frames or shots, respectively), the associated video clip can be viewed in a playback window. Next to the three layers of the active index tree, the corresponding layer of the second index tree (see Subsection 3.4) is rendered in a smaller panel. Finally, on the top right a panel with navigation tools is shown (back button, history, etc.).

This user interface allows for browsing through the video content without having to visualise the entire three-dimensional index trees. In earlier experiments we found that two-dimensional user interfaces are easier to handle for non-expert users, if sufficient context information is given. Furthermore, this user interface can be implemented at a minimum demand of resources. All panels are based on the SVG cell. Interaction is exclusively based on ECMAScript and mostly executed locally. Remote access is only required if the user switches to a layer that has not been used before. The next section gives first evaluation results of the proposed video browsing application.

5. EVALUATION

5.1. Test environment

The following components were used for the prototype presented in this section. Firstly, clips with the following content were used: advertisements clips (short shots, fast changes, high quality images), cartoons (reduced colour palette, few colour gradations, slow scene changes, low motion activity), documentaries (alternating videos and animations, slow scene changes), movie clips (average image quality, average motion activity) and news clips (low motion activity, sometimes bad image quality). The media clips were captured from German satellite programs and stored in PAL format (720 by 576 pixels, 25 fps). Figure 3 shows examples.

Frames were described by seven visual MPEG-7 descriptors: *Color Layout*, *Color Structure*, *Dominant Color*, *Edge Histogram*, *Homogeneous Texture*, *Region-based Shape* and *Scalable Color*. Descriptor extraction was performed using the MPEG-7 eXperimentation model. After extraction, descriptions elements were normalised to identical intervals ([0, 1]).

Indexing was performed using self-organising maps (SOM; see Subsection 2.2). SOMs were computed with a hexagonal layout (every non-border cluster has six neighbours), six rows and eight columns. For learning, a Gaussian neighbourhood kernel was used. Maps were initialised randomly. Learning was performed in two iterations. In the first iteration 10000 learning steps were performed with learning rate α=0,05 and radius 5 (clusters). In the second iteration (fine tuning) 100000 learning steps were performed with learning rate α=0,02 and radius 3. For every dataset 15 separate SOMs were computed and the best map was chosen by the minimum quantisation error (as suggested in [11]).

The entire video browsing prototype is based on free software. Media access is implemented using Java and the Java Media Framework. Descriptions are extracted by the MPEG-7 reference implementation from the eXperimentation Model. SOMs are computed using the C-implementation provided by the Helsinki University of Technology [11]. Visualisation of maps is based on scalable vector graphics [21]. Visualisation of maps is implemented in Perl scripts and the SVG output is rendered by the Adobe SVG Viewer plug-in (tested for Netscape Navigator and Microsoft Internet Explorer). Finally, event-based interaction is implemented in ECMAScript scripts.

5.2. Experimental results

This subsection summarises our experiences with the video browser prototype. So far, we have not conducted a user study. Therefore, all presented results are preliminary based the authors' observations. In the first part of this section we will investigate the look-and-feel of the video browser. The second part discusses quantitative criteria, advantages and disadvantages as well as usage types.

Figures 8 and 9 illustrate hierarchical layer dependencies of time index tree and content index tree. The time index tree shows the top layer and two detail layers. Time-code values of key-frames depicted in clusters act as an additional source of information to the user. Since all elements are required for

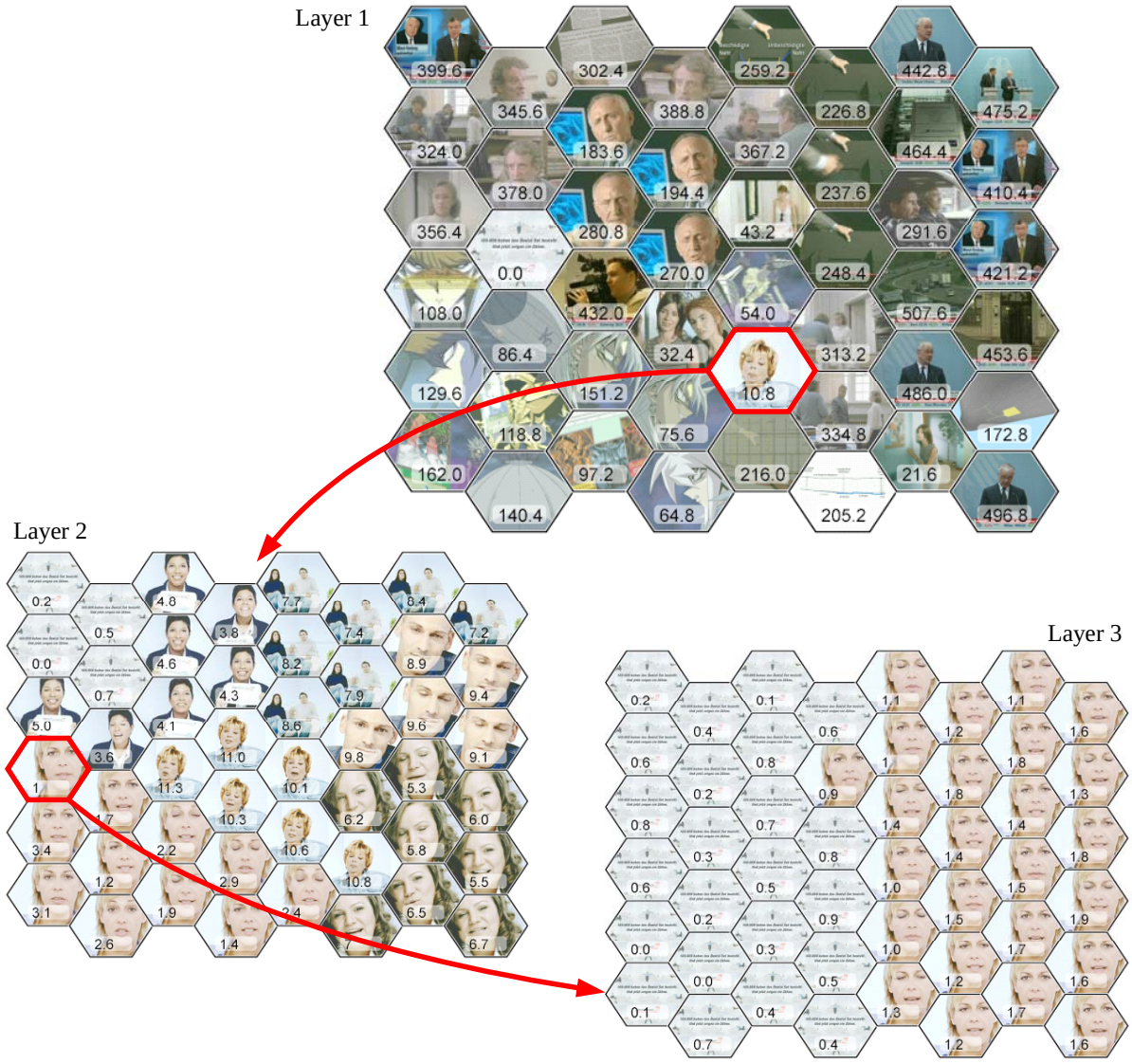

Figure 8. Example screenshot of time index view. The figure shows maps on three layers. Layer 1 is the top layer computed from the test videos used in the evaluation.

browsing, no holes are allowed in the SOMs. The algorithm describes in Subsection 4.3 solves this problem sufficiently. Some artefacts (e.g. some non-circular clusters) are due to its heuristic nature. Still, clustering of similar content is semantically understandable (especially on detail levels). The major clustering criteria seem to be colour distributions and edge layouts. This is similarly true for the content index tree (Figure 9). The figure illustrates the top layer for the test data used and one detail layer. If shots have similar content, they are clustered together. Hence, content index tree SOMs have holes and varying numbers of detail layers. Shot-content is visualised spatially. For example, Layer 2 organises the content of an animation sequence in a looped path (starting from bottom right; see time-code values). Interestingly, colour information is not the dominating clustering criterion. For example, the third and fourth cluster in the fifth row of Layer 1 of the content index tree have similar structures but different colours. In conclusion, since colour and structure are the two dominating clustering criteria, similarity is spatially perceivable in the two-dimensional SOMs.

Generally, the layer map size determines the capacity of the video browser index trees. For the example, we use maps with six rows and eight columns per row. Therefore, every map layer has 48 elements and a time index tree with three layers has a capacity of $48^3 = 110592$ frames. For a frame rate of 25 frames per second (PAL, SECAM), this number equals to 73 minutes of video: Three layers are sufficient to browse through 73 minutes of content. A map size of 48 elements was chosen, because humans are able to perceive between 50 and 100 icons spatially by one look. Therefore, 48 is a very convenient number of items. Additionally, smaller maps can be computed faster and be visualised easier.

Next, we investigate major differences (in terms of practical usage) of content index tree and time index tree. The content index tree clusters dependencies in the content: Scenes that have no temporal relationship. Scenes with similar colour and

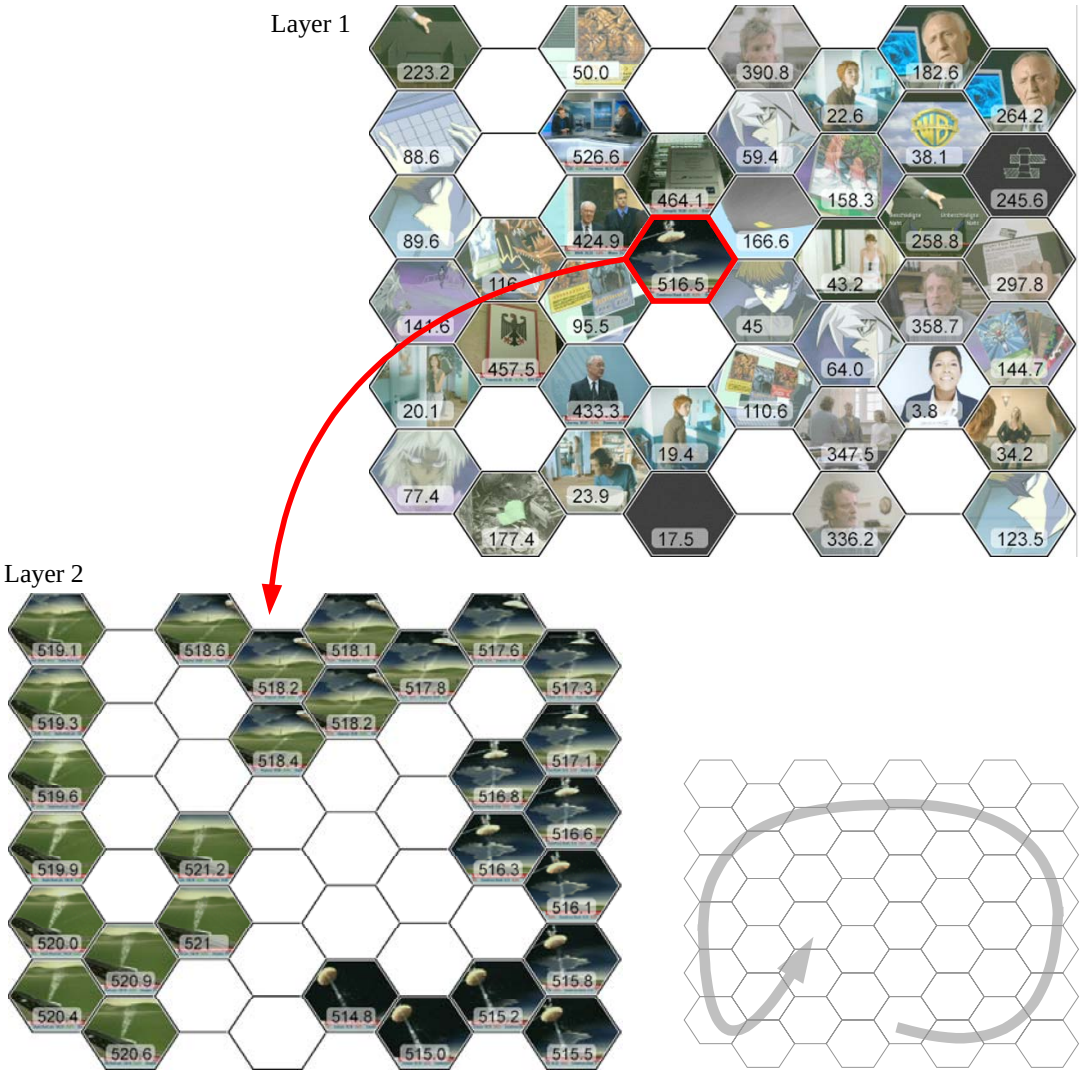

Figure 9. Example screenshot of content index view. The top layer visualises shots by representative frames. Layer 2 shows an animation shot in detail. Clustered spatially by similarity, the frames of the animation shot follow a loop.

structure properties are clustered together. For very similar scenes, one representative is chosen and the others are omitted. The content index tree shows the 'assets' of a video stream: it successfully selects prototypes of all appearing types of content and presents them to the user. Furthermore, the content index tree guarantees (on the top level) that the entire content is visualised in one view. In contrast, the time index tree clusters temporal transitions. To a certain extent it preserves the story and gives 'suggestions' for more detailed analysis in temporal order.

Technically, content index tree and time index tree are not that different: the frames selected as representatives for clusters are often located in close proximity in the video stream (of course, depending on the shot structure). If shots are short (as, for example, in advertisement clips) content-index tree and time index tree use mostly similar selections of key-frames. Additionally, since SOM clustering destroys the order in the set of selected key-frames anyway, content index tree and time index tree may appear highly similar (especially, on the top levels).

Usage experience shows that the content index tree is the main browsing tool. It is employed to identify interesting areas in the video content and analyse them in greater detail. The time index tree is mainly used in the starting phase to get a first impression of the video data, for orientation during a browsing session and as a tool for associative browsing. Since it preserves the temporal order (the story) of the video, it allows for semantic browsing through the content.

From our experiments, we draw the conclusion that the proposed video browsing approach is reasonable. Its major advantages are: Firstly, the video browser makes use of content analysis techniques and similarity-based clustering. This supports human visual perception and allows fast and effective browsing. Secondly, it summarises the assets of a video stream in an easy to overlook structure. The video browser allows real content-based random access of video data. Spatially, the video browser user interface makes use of human spatial memory. Since information overload is avoided by using small maps, the user can browse through the data quickly. Furthermore, the implemented navigation style is easy to understand. It does not implement revolutionary new interaction paradigms but is based on simple click operations. Finally, the spatial layout

used in the layer maps fits to the users spatial expectations. In the video browser, video content is presented in a natural way. One major disadvantage of the proposed video browser is that temporal organisation of video is destroyed. The 'video feeling' is lost when analysing the content by the index trees. Even though illustrating the time-code together with cluster representatives allows the user to comprehend temporal organisation intellectually, the obvious visual temporal flow is lost (especially in the content index tree).

6. CONCLUSIONS

The paper describes a novel video browsing application that is based on two index structures. A time index tree visualises the temporal structure and a content index tree visualises the video stream content. The application is interactive: The user can browse through the trees and switch between the trees. Browsing is easy, because several additional panels visualise navigation-relevant context information. Furthermore, the index trees integrate visual information retrieval know-how as media objects used on index layers are clustered content-based. Media objects are described by visual MPEG-7 descriptions. Similarity-based clustering is performed using self-organising maps. From the implementation point of view, the video browser is novel as it is exclusively based on free software. Scalable vector graphics are used for index visualisation and the entire browsing application can be accessed through a web browser.
The major contribution of the video browsing application is allowing time and content-based access simultaneously. Moreover, it integrates ideas from information visualisation, information browsing and content-based information retrieval. The result is a powerful application that makes video content transparently accessible.

7. ACKNOWLEDGEMENTS

The author would like to thank Christian Breiteneder for his valuable comments and suggestions for improvement. The work presented in this paper is part of the VizIR project [3]. VizIR is funded by the Austrian Scientific Research Fund (FWF) under grant number P16111-N05.

8. REFERENCES

[1] M. Bober, "MPEG-7 Visual Shape Descriptors," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 11, No. 6, pp. 716-719, 2001.

[2] A. Del Bimbo, *Visual Information Retrieval*, San Francisco, CA: Morgan Kaufmann, 1999.

[3] H. Eidenberger, and C. Breiteneder, "VizIR – A Framework for Visual Information Retrieval," *Journal of Visual Languages and Computing*, Vol. 14, No. 5, pp. 443-469, 2003.

[4] B. Furht, S.W. Smoliar, and H. Zhang, *Video and Image Processing in Multimedia Systems*, Boston MA: Kluwer 1996.

[5] M. Höynck, C. Mayer, and J.R. Ohm, "Application of MPEG-7 Descriptors for Temporal Video Segmentation," *SPIE Proceedings*, Vol. 4676, pp. 347-358, 2002.

[6] A. Hampapur, R. Jain, and T. Weymouth, "Production Model-based Digital Video Segmentation," *Multimedia Tools and Applications*, Vol. 1, No. 1, pp. 9-46, 1995.

[7] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: a Review," *ACM Computing Surveys*, Vol. 31, No. 3, pp. 264-323, 1999.

[8] S. Jeannin, and A. Divakaran, "MPEG-7 Motion Descriptors," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 11, No. 6, pp. 720-724, 2001.

[9] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, "Engineering Applications of the Self-Organising Map," *Proceedings of IEEE*, Vol. 84, No. 10, pp. 1358-1384, 1996.

[10] T. Kohonen, "The Self-Organising Map," *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1464-1480, 1990.

[11] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen, "SOM-PAK: The Self-Organizing Map Program Package," Helsinki University of Technology, Tech. Rep., 1995.

[12] P. Koikkalainen, and E. Oja, "Self-Organising Hierarchical Feature Maps," Proc. Neural Networks Conference, pp. 279-284, 1990.

[13] J. Laaksonen, M. Koskela, S. Laakso, and E. Oja, "PicSOM – Content-based Image Retrieval with Self-Organising Maps," *Pattern Recognition Letters*, Vol. 21, No. 13-14, pp. 1199-1207, 2001.

[14] M.S. Lew (ed.), *Principles of Visual Information Retrieval*, Heidelberg, Germany: Springer, 2001.

[15] B.S. Manjunath, J.R. Ohm, V.V. Vasudevan, and A. Yamada, "Color and Texture Descriptors," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 11, No. 6, pp. 703-715, 2001.

[16] B.S. Manjunath, P. Salembier, and T. Sikora, *Introduction to MPEG-7*, San Francisco CA: Wiley, 2002.

[17] O. Marques, and B. Furht, *Content-Based Image and Video Retrieval*, Boston MA: Kluwer, 2002.

[18] Y. Rui, and T.S. Huang, "Relevance Feedback Techniques in Image Retrieval," in M.S. Lew (ed.), *Principles of Visual Information Retrieval*, Heidelberg, Germany: Springer, 2003).

[19] J.W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. on Computers*, Vol. 18, No. 5, pp. 401-409, 1969.

[20] J.R. Vacca, *VRML* (second edition), Boston MA: Academic Press, 1998.

[21] World Wide Web Consortium, Scalable Vector Graphics standard website, http://www.w3c.org/Graphics/SVG/, last visited 2004-08-12.

[22] M. Yeung, B.L. Yeo, and B. Liu, "Extracting Story Units from long Programs for Video Browsing and Navigation," Proc. IEEE Multimedia Computing and Systems Conference, pp. 296-305, 1996.

[23] H.J. Zhang, A. Kankanhalli, and S. Smoliar, "Automatic Partitioning of Video," *ACM Springer Multimedia Systems*, Vol. 1, No. 1, pp. 10-28, 1993.

[24] H.J. Zhang, C.Y. Low, and S. Smoliar, "Video Parsing and Browsing using compressed Data," *Multimedia Tools and Applications*, Vol. 1, No. 1, pp. 89-111, 1995.