An Extended Framework Structure in MUVIS for Content-based Multimedia Indexing and Retrieval


*Tampere University of Technology, Tampere, FINLAND
{serkan.kiranyaz, esin.guldogan, mari.partio, murat.birinci, stefan.uhlmann, moncef.gabbouj}@tut.fi

**Nokia Mobile Phones, Tampere, FINLAND
{olcay.guldogan, ahmad.iftikhar}@nokia.com

Abstract: In this paper, we propose an extended framework structure designed for MUVIS multimedia indexing and retrieval scheme in order to achieve the dynamic integration and run-time execution for the following operations within the context of multimedia indexing and retrieval: Visual and aural feature extraction, shot boundary detection and spatial segmentation. We introduce a general structure of the extended framework, and describe its well-defined interface, the general-purpose instructions and the conditions for integrating new algorithms into MUVIS. Finally, the limitations and the advantages of the integration scheme are discussed whilst providing examples from some algorithms already implemented.

Keywords: Content-based indexing and retrieval, feature extraction, shot boundary detection, spatial segmentation, MUVIS.

INTRODUCTION

It is a known fact that recent technological hardware and network improvements along with the presence of Internet have caused a rapid increase in the size of digital audio-visual information that is used, handled and stored via several applications. Besides several benefits and usages, such massive collection of information has brought storage and especially database management problems. In order to realize the full potential of these databases, tools for automated indexing and intelligent search engines are urgently needed. Indeed, image and video cataloguing, indexing and retrieval are the subject of active research in industry and academia across the world. This reflects the commercial importance of such technology and thus several content-based indexing and retrieval techniques and applications have been developed such as MUVIS system [15], Photobook [19], VisualSeek [21], Virage [22], VideoQ [1], etc. The common property of all such systems is that they all provide some kind of framework and several techniques for indexing and retrieving still images and/or audio-video files. Due to the storage problem of such digital multimedia collections, such systems have the tendency to work on compressed domain.

Among all, MUVIS aims to bring a unified and global approach to indexing, browsing and querying of various digital multimedia types such as audio/video clips and digital images. The primary motivation behind MUVIS is to achieve a unified and global framework architecture upon which a robust set of applications for capturing, recording, indexing and retrieval combined with browsing and various other visual and semantic capabilities can be implemented. Variations in formats, representations and other parameters in today’s digital multimedia world such as codec types, file formats, capture and encoding parameters, may significantly affect the efficiency and performance of the indexing and retrieval operations. Therefore, covering a wide-range of multimedia family and especially the last generation multimedia codecs, MUVIS is recently reformed to provide an extended framework structure upon which robust algorithms can be implemented, tested, configured and compared against each other. It further supports various browsing capabilities, a hierarchic video summarization scheme and an efficient retrieval (QBE) method, the so called Progressive Query (PQ) [11], which can work along with a dynamic indexing structure, so called Hierarchical Cellular Tree (HCT) [11]. HCT is mainly designed to work with PQ in order to provide the earliest possible retrievals of the most relevant items. Furthermore, HCT indexing body can be used for efficient browsing and navigation among database items.

In this paper, we propose an extended framework structure designed for MUVIS multimedia indexing and retrieval operations in order to achieve the dynamic integration and run-time execution of the following:
The contemporary visual and Aural Feature eXtraction (FeX and AFeX) framework

SBD (Shot Boundary Detection) framework

SEG (Spatial SEGmentation) framework

The main purpose of the extended framework is to use MUVIS as a common platform to develop and test novel techniques on spatial segmentation and shot boundary detection along with the existing (visual and aural) feature extraction. Using such techniques within MUVIS over the multimedia databases yields the necessary basis to perform further (multi-modal) analysis and to achieve a better indexing and retrieval performance. The rest of the paper is organized as follows: the following section presents an overview on MUVIS. Section 2 presents the contemporary FeX and AFeX frameworks for dynamic visual and Aural Feature eXtraction schemes and file formats. Section 3 presents the proposed framework structure for Shot Boundary Detection (SBD) designed for video collections in MUVIS databases. Section 4 introduces an efficient framework for spatial SEGmentation (SEG). Section 5 presents some visual and numerical experimental results and finally in Section 6, we give some conclusive remarks.

1. Muvis– An Overview

As shown in Figure 1, MUVIS framework is based upon three applications, each of which has different responsibilities and functionalities. AVDatabase is mainly responsible for real-time audio/video database creation with which audio/video clips are captured, (possibly) encoded and recorded in real-time from any peripheral audio and video devices connected to a computer. DbsEditor performs the offline operations for the multimedia databases such as dynamic database creation with appending (and removing) multimedia items, offline feature extraction and Hierarchical Cellular Tree (HCT) [11], based indexing operations, etc. MBrowser is the primary media browser and retrieval application into which the Progressive Query (PQ) technique [11] is integrated as the primary retrieval scheme. The exhaustive search based Normal Query (NQ) is the alternative query scheme within MBrowser. Both PQ and NQ can be used for retrieval of the multimedia primitives with respect to their similarity to a queried media item (an audio/video clip, a video frame or an image). In order to achieve the ultimate goal for the user point of view that is the retrieval of the most relevant items in the earliest possible time regardless of the database size, HCT [11] is integrated into MUVIS and PQ is used over HCT indexing body to speed up the queries.

![Figure 1: MUVIS Block Diagram.](image-url)
especially on large multimedia databases. Moreover, HCT can provide a basis for accomplishing an efficient browsing scheme, namely HCT Browsing. The hierarchic structure of HCT is quite appropriate to give an overview to the user about what lies under the current level so that if well supported via an user friendly Graphical User Interface (GUI), HCT Browsing can turn out to be a guided tour among the database items. The details of HCT Browsing and the necessary GUI support within MUVIS framework can be found in [11].

1.1. MUVIS Multimedia Databases

MUVIS databases are formed using the variety of multimedia types belonging to MUVIS multimedia family as given in Table 1. For instance AVDatabase allows the user to create an audio/video MUVIS database in real time via capturing or by converting into any of the specified format within MUVIS multimedia family. Since both audio and video formats are the most popular and widely used, a native clip with the supported format can directly be inserted into a MUVIS database without any conversion. This is also true for the images but if the conversion is required anyway by the user, any image can be converted into one of the “Convertible” image types presented in Table 1.

MUVIS system supports the following types of multimedia databases:
- Audio/Video databases include only audio/video clips and associated indexing information
- Image databases include only still images and associated indexing information.
- Hybrid databases include audio/video clips, images, and associated indexing information.

As illustrated in Figure 1, Audio/Video databases can be created using both AVDatabase and DbsEditor applications and image databases can only be created using DbsEditor. Hybrid databases can be created by appending images to video databases using DbsEditor, or by appending audio/video clips to image databases using either DbsEditor or AVDatabase (in real-time).

### Table 1: MUVIS Multimedia Family

<table>
<thead>
<tr>
<th>Codecs</th>
<th>Sampling Freq.</th>
<th>Channel Number</th>
<th>File Formats</th>
<th>Codescs</th>
<th>Frame Rate</th>
<th>Frame Size</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP3</td>
<td>16, 22,050,</td>
<td>Mono</td>
<td>MP3</td>
<td>H.263+</td>
<td>1.25 fps</td>
<td>AVI</td>
<td>MP4</td>
</tr>
<tr>
<td>AAC</td>
<td>24, 32, 44.1 KHz</td>
<td>Mono</td>
<td>AAC</td>
<td>MPEG-4</td>
<td></td>
<td></td>
<td>3gp</td>
</tr>
<tr>
<td>G721</td>
<td></td>
<td>Stereo</td>
<td>AVI</td>
<td>YUV 4:2:0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G723</td>
<td></td>
<td></td>
<td>MP4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCM</td>
<td>8 – 16 KHz</td>
<td>Mono</td>
<td>3gp</td>
<td>RGB 24</td>
<td></td>
<td>H.264</td>
<td></td>
</tr>
<tr>
<td>AMR</td>
<td></td>
<td></td>
<td>AMR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formats</th>
<th>MUVIS Audio</th>
<th>MUVIS Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convertible Formats</td>
<td>JPEG</td>
<td>JPEG 2K</td>
</tr>
<tr>
<td>Non-convertible Formats</td>
<td>PCX</td>
<td>GIF</td>
</tr>
</tbody>
</table>

1.2. Retrieval Scheme in MUVIS: Progressive Query

The usual approach for multimedia indexing is to map database items such as images, video and audio clips into some high dimensional vector space, the so called feature domain. The feature domain may consist of several types of features such as visual, aural, motion, etc. as long as the database contains multimedia items from which those particular features can be extracted. Among so many variations, careful selection of the feature sets allows capturing the semantics of the database items. Especially for large multimedia databases the number of features extracted from the raw data is often kept large due to the naïve expectation that it helps to capture the semantics better. Content-based similarity between two database items can then be assumed to correspond to the (dis-)similarity distance of their feature vectors. Henceforth, the retrieval of a similar database items with respect to a given query (item) can be transformed into the problem of finding the database items, which gives such feature vectors that are close to the query feature vector. This is so-called QBE, which is one of the most common retrieval schemes. The exhaustive search based QBE operation is so called NQ, and works as follows: using the available aural or visual features (or both) of the queried multimedia item (i.e. an image, a video clip, an audio clip, etc.) and all the database items, the similarity distances are calculated and then merged to obtain a unique similarity distance per database item. Ranking the items according to their similarity distances (to the queried item) over the entire database yields the query result. NQ is costly and CPU intensive especially for large-scale multimedia databases since the number of similarity distance calculations is proportional with the database size. Furthermore any abrupt stopping during the query process will cause total loss of retrieval information and essentially nothing can be saved out of query operation so far performed.
In order to eliminate the aforementioned drawbacks and provide a faster query scheme, the PQ has been integrated into MUVIS framework. It is a dynamic retrieval technique, which is mainly designed to bring an effective solution especially for queries on large-scale multimedia databases. In this way it achieves a series of sub-query results that will eventually be converging to the retrieval via full-scale search; however in a faster and with no or minimal system requirements. Basically PQ performs over a series of sub-queries each of which is a fractional query process that is performed over a sub-set of database items. The items within a sub-set can be chosen by any convenient manner such as randomly or sequentially but the size of each sub-set should be chosen via a suitable (to human perception) unit such as time (period). The sub-query time will vary due to the number of features present in the database and the speed of the computer where it is running. In order to avoid such uncertainties, the entire PQ scheme is designed over periodic sub-queries as shown in Figure 2 with a user defined period value \( t = t_p \). The details of PQ can be obtained from [11].

### 1.3. Indexing Scheme in MUVIS: Hierarchical Cellular Tree

Especially for the management of large multimedia databases, there are certain requirements that should be fulfilled in order to improve the retrieval feasibility and efficiency. Above all, such databases need to be indexed in some way and traditional methods are no longer adequate. It is clear that the nature of the search mechanism is influenced heavily by the underlying architecture and indexing system employed by the database. For this purpose HCT is designed to bring an effective solution especially for indexing multimedia databases and furthermore to provide an enhanced browsing capability, which enables user to make a guided tour within the database. A pre-emptive cell search mechanism is introduced in order to prevent the corruption of large multimedia item collections due to the misleading item insertions, which might occur otherwise. In addition to this, the similar items are focused within appropriate cellular structures, which will be the subject to mitosis operations when the dissimilarity emerges as a result of irrelevant item insertions. Mitosis operations ensure to keep the cells in a focused and compact form and yet the cells can grow into any dimension as long as the compactness prevails. HCT is particularly designed for PQ, in order to maximize the retrieval efficiency for the user point of view. It is mainly a hierarchical clustering method where the items are partitioned depending on their relative distances and stored within cells on the basis of their similarity proximity. The similarity distance function implementation is a black-box for the HCT and it is a self-organized tree, which is implemented by genetic programming principles. This basically means that the operations are not externally controlled; instead each operation such as item insertion, removal, mitosis, etc. are carried out according to some internal rules within a certain level and their outcomes may uncontrollably initiate some other operations on the other levels. More detailed information about HCT indexing scheme can be obtained from [11].

PQ over HCT is now used as the primary query scheme in MUVIS framework in order retrieve the relevant items in the earliest possible time regardless of the database size. Extensive experiments approve that the speed of the retrievals is significantly improved and the HCT indexing structure shows no sign of degradations when the database size is increased.

### 2. FeX-AFeX Framework in Muvis

Feature vectors of media items in a MUVIS database are represented by (unit) normalized array of numbers. Also to support dynamic integration of exclusive FeX/AFeX modules (DLLs), features should be represented in a common and easily supportable format, which is simply the vector representation. These (feature) vectors are extracted by dynamic and independent FeX modules and they are essentially managed and used by two applications in MUVIS, namely MBrowser and DbsEditor. MBrowser is capable of merging multiple features for querying. The merging scheme used in MBrowser requires unit-normalized feature vectors. For this purpose the value of each item in a feature vector is divided by its theoretical maximum value. DbsEditor is mainly responsible for the management of a feature extraction
operation. DbsEditor also supports the extraction of multiple sub-features, each of which is a single feature with unique parameters. MBrowser is used to query (via QBE) a media item within a MUVIS database, as long as the database contains at least one aural/visual feature and the corresponding FeX/AFeX module is available. When necessary (i.e. for an external image query), MBrowser uses the same FeX/AFeX module for extracting the features of the queried media and for obtaining the similarity measure.

2.1. FeX Framework
There are mainly 2 visual media items in MUVIS framework: video clips and images. Image features are extracted directly from 24bit RGB frame buffer by decoding the image. On the other hand the visual features for video clips are extracted from the key-frames of the clips. In real time video recording case AVDatabase may optionally store the uncompress key-frames of a video clip along with the video bit-stream. If not, DbsEditor can extract the key-frames from the video bit-stream and stores them in raw (YUV 4:2:0) format. These key-frames are the INTRA frames in MPEG-4, H.263 or H.264 bit-stream. In general, the encoders use a shot detection algorithm to select the INTRA frames but sometimes a forced-intra scheme might further be applied to prevent a possible degradation. This default key-frame selection scheme is about to change by the integration of the proposed SBD (Shot Boundary Detection) framework within the extended MUVIS framework. Upon completion, there will be several techniques to choose better key-frames than the encoders’ selections and use them during the FeX operation.

FeX interface is defined in Fex_API.h C-type header file. As mentioned before, any FeX algorithm should be implemented as a DLL using this API header file. More detailed information about FeX framework can be obtained in [9].

2.2. AFeX Framework
AFeX framework mainly supports dynamic audio feature extraction module integration for audio clips. Figure 3 shows the API functions and linkage between MUVIS applications and a sample AFeX module. Each audio feature extraction algorithm should be implemented as a DLL for audio clips. As shown in Figure 4, any implemented SBD module can therefore be used to extract the following entities from a video clip in a MUVIS database:

- The list of shot boundaries: The start and end frame numbers (indexes) of each shot in a video stream.
- The list of key-frames: One or more key-frames can be chosen in a shot.

SBD * dll

Figure 4: SBD module interaction with MUVIS-DbsEditor

SBD modules are mainly developed for and used by DbsEditor application during the indexing of the video clips in a MUVIS database and the shot boundaries with key-frames extracted are stored along with the associated video clip in the same directory. All the SBD algorithms should be implemented as a Dynamic Link Library (DLL) with respect to a specific SBD API, and stored in an appropriate folder (in the framework and its role on audio-based multimedia indexing and retrieval scheme in MUVIS can be found in [11]).
same directory with the applications or in "C:\MUVIS" directory). Also the name of a SBD DLL should have the naming convention as follows:

**SBD_[fourCC code].dll** (i.e. “SBD_YUVD.dll”)

*SBD* API provides the necessary handshaking and information flow between a MUVIS application and a SBD module. If the database contains video clips, for each video clip, again a separate SBD file per SBD module is created and stored along with the video clip. The naming convention for SBD files is as follows:

[indexed video file name]_SBD.[SBD module FourCC] (i.e. “MTV_Clip_12_SBD.YUVD”)

### 3.1. Data Structures in SBD Modules

Four structures and five API function properties (name and types) are declared in a C-type template encapsulated in *SBD_API.h*. The owner of a SBD module should implement all these API functions. There also exists a macro in order to convert a character array to *FourCC* code as described in the following binary macro:

\[
FourCC(x) = \begin{cases} 
(x >> 24) & 0xff0000 \\
(x >> 8) & 0xff00 & \text{if } x = 0 \\
(x << 24) & 0xff000000 & \text{if } x > 0 
\end{cases}
\]

*SBDParam Structure:*

Created and filled by the SBD module for handshaking operation. A MUVIS application (i.e. *DbsEditor*) calls *SBD_Bind* function once with a pointer to this structure in run-time. Therefore, SBD module fills the following members of this structure to introduce itself to the application:

- *char sbd_name[512]*: Description of the SBD module (i.e. “Shot Detection by YUV Histogram Difference”).
- *long sbd_fourcc*: SBD module fourcc code (i.e. *IntroCC(YUVD)*). Unique identification code for the SBD algorithm. Used by applications to identify each SBD module and associated files.
- *unsigned int sbd_param_no*: Number of parameters (i.e.4 for YUVD).
- *long* *sbd_paramFourcc*: Array of parameter fourcc codes (i.e. [FourCC('Ybin'), FourCC('Ubin'), FourCC('Vbin'), FourCC('Thr')]). Used for display purposes.
- *double* *sbd_param_default*: Array of parameter default values (i.e. [8,4,4, 0.5] for YUVD).

*FrameParam Structure:*

Each time a new frame is fed into SBD module via calling *SBD_NewFrame* API function with the frame stored inside the given *FrameParam* structure. The format of the frame must be YUV 4:2:0.

- *unsigned char *buffer*: Raw frame data in a single dimension array. Frame pixels are located in left-to-right top-to-down raster scan order. The frame is in YUV4:2:0 format, so first part of the buffer keeps the Y values for each pixel, then a quarter size of it array keeps the U values each per 4 pixels, and the last part keeps the V values each per 4 pixel.
- *unsigned long fr_index*: The index number of the frame. Note that these numbers may not be sequential and they are used to determine shot boundaries and key-frame indexes.
  - *int Xs, int Ys*: Width and height of the frame.

*ShotBoundary Structure:*

A simple structure to indicate the start and end frame indexes of a shot.

- *unsigned long start, end*: The start and end frame indexes of a shot

*SBD_OUT Structure:*

The primary output structure of a SBD module. The module create and fill this structure and once all the frames are fed into SBD module, then it must return this structure filled by the extracted shot boundaries and key-frames to the application via *SBD_GetOut* API function.

- *ShotBoundary* *shots*: The array of shots
- *unsigned int no_shots*: Number of shots
- *unsigned long *KFs*: The array of Key-Frames
- *unsigned int no_KFs*: Number of Key-Frames

### 3.2. API Functions in SBD Modules

*SBD* framework has five C-type API functions, which are listed as below.

- **int SBD_Bind(SBDParam*)**: Used for handshaking operation between the MUVIS application (*DbsEditor*) and the SBD module (DLL). A pointer to *SBDParam* structure is given as a parameter. The SBD module fills the structure to introduce itself to the application. This function should be called only once at the beginning, just after the application links the SBD module in run-time. This function should return 0 if a problem occurs, and a nonzero value if successfully completed.

- **int SBD_Init(double*)**: Used to initialize the SBD module. The Shot Boundary Detection parameters are given in a double array, which has the size defined in the *SBDParam* structure. The SBD module performs necessary initialization operations, i.e. memory allocation, table creation etc. This function should be called once at the beginning after the SBD operation parameters are set by the user via GUI of *DbsEditor* application. This function should return 0 if a problem occurs, and a nonzero value if successfully completed.

- **int SBD_NewFrame(FrameParam frame, int eos)**: Used to feed a new frame into the SBD module. The new frame in *FrameParam* structure is used in the shot boundary detection algorithm and the second parameter *eos* is a nonzero value if this is the last frame in video stream –when the shot boundary detection is due to terminate, otherwise 0.

- **int SBD_Exit(SBDParam*)**: Used for both resetting and terminating the SBD module operation. It deallocates the memory spaces for *SBDParam* structure allocated in *SBD_Bind* function. Also, if *SBD_Init* has been called already, calling this
function with no SBDParam pointer (set it to NULL) resets the SBD module to prepare it for further SBD operations. This function should be called at least once while the MUVIS application is closed, but usually it can be called at the end of each SBD operation with different SBD parameters.

- SBD OUT* SBD_GetOut() : This is the function to get the output of SBD module as a pointer of SBD_OUT structure. The SBD_OUT object should be created and allocated by the SBD module owner and once SBD_Exit is called, the object can then be de-allocated again by the SBD module.

3.3. Step-by-Step Shot Boundary Detection Processes in DbsEditor

The following steps are carried in DbsEditor application for shot boundary detection and key-frame extraction using a SBD module implemented as a DLL according to SBD API.

1) Whenever the DbsEditor is started, it looks for the SBD modules (DLLs) in proper directories, loads them and links all their functions. It then proceeds by calling their SBD_Bind functions to establish handshaking operation. Associated SBDParam structures are filled by the modules, so that DbsEditor has all the necessary information for proper SBD operation.

2) The adaptive DbsEditor user interface lets the user supply the SBD parameters and once the user enters the parameters to perform SBD operation over the video clips of the active database, it first calls the SBD_Init function with the given parameters to initialize the SBD module.

3) For each frame in the next video clip of the database, the SBD_NewFrame function is called to feed the frame into the module.

4) Once all the frames are fed, the last frame is to signal <end of stream> to module via setting eos=1, then the extracted shots (with their boundaries) and key-frames can be retrieved by calling SBD_GetOut function.

5) Now in order to perform the same SBD operation for the next video clip, steps 3 and 4 are simply repeated.

6) In order to perform a new SBD operation with different parameters, first SBD_Exit function is called to reset the module (with passing NULL pointer) and then steps 2 to 4 are simply repeated.

7) When DbsEditor application is about to terminate, SBD_Exit function is called for final cleanup. This is necessary to de-allocate the members inside the SBDParam structure even in case steps 2 to 4 have never been executed during the lifetime of the application.

4. SEG Framework in Muvis

Similar to FeX framework, a SEGmentation (SEG) framework is designed to dynamically integrate any spatial segmentation algorithms into MUVIS using a dedicated SEG API, which is described in this section. SEG modules are used to create Segmentation Masks (SMs) from an image or a key-frame of a video clip. Each SM contains two or more segmented regions (segments) indicated by a distinct 8 bits gray-scale value (between 0-255). Therefore, as a convenient limit there can be maximum 256 segments in a SM image. Any SEG module is responsible to assign different (and unique) gray-scale values to the pixels within each segment and the segment assigned with pixel value “0” can be used for the background segment.

SEG API provides the necessary handshaking and information flow between a MUVIS application and the SEG DLL. SEG modules are mainly developed for and used by DbsEditor application during the indexing of the video clips and images in a MUVIS database. All the SEG algorithms should be implemented as a Dynamic Link Library (DLL) with respect to the specific SEG API, and stored in an appropriate folder.

The naming convention of a SEG module must be as follows:

SEG_[fourCC code].dll (i.e. “SEG_RSST.dll”)

4.1. Data Structures in SEG Modules

Two structures, one enumeration and four API function properties (name and types) are declared in SEG_API.h. The owner of a SEG module should implement all these API functions. There also exists a macro in order to convert a character array to _FourCC code.

FrameType Enumeration:

Enumerates two frame formats to be used between the application and a SEG module. Currently RGB 24 bit and YUV 4:2:0 frame buffers are supported.

SegParam Structure:

Created and filled by the SEG module for handshaking operation. DbsEditor calls Seg_Bind function once with a pointer to this structure during the start-up phase. Therefore, SEG module fills the following members of this structure to introduce itself to the application:

- char seg_name[512] : Description of the SEG module (i.e. "Segmentation by RSST").
- long seg_fourcc : SEG module fourcc code (i.e. _FourCC('RSST')). Unique identification code for the SEG algorithm. Used by applications to identify each SEG module and associated files.
- unsigned int seg_param_no : Number of parameters (i.e.4 for RSST).
- long* seg_param_fourcc : Array of parameter fourcc codes (i.e. _FourCC('NoS'), _FourCC('ThrS'), …). Used for display purposes.
- double* seg_param_default : Array of default parameters (i.e. [8, 0.5, 4, 0.5] for RSST)
- FrameType ftype : input frame that should be given to the SEG module (i.e. _RGB24)

FrameParam Structure:

Each time a new frame is fed into SEG module via calling Seg_Extract API function with the frame stored inside the given SegParam structure. The format of the frame is the format that is specified by the SEG module in the SegParam structure.
4.3. Step-by-Step Segmentation Process in DbsEditor

The following steps are carried in DbsEditor application for spatial segmentation of any media item (a video key-frame or an image):

1) Whenever the DbsEditor is started, it looks for the SEG modules (DLLs) in proper directories, loads them and links all their functions. It then proceeds by calling their Seg_Bind functions to establish handshaking operation. Associated SegParam structures are filled by the modules, so that DbsEditor has all the necessary information for the segmentation operation.

2) The adaptive user interface of DbsEditor lets the user supply the SEG parameters and once the user enters the parameters to perform segmentation over the database, it first calls the Seg_Init function with the given parameters to initialize the SEG module.

3) For each frame in the database (either all the key-frames or images), DbsEditor calls the Seg_Extract function passing the input frame within the FrameParam structure, and the fills the buffer of the output FrameParam structure with SM indexes.

4) Once the segmentation operation with certain parameters are completed for all the media items in database, Seg_Exit function is called to reset the SEG module (with passing NULL pointer).

5) Now in order to perform the same segmentation (use the same SEG module) with different parameters (if required), steps 3 and 4 can be repeated if the application is not terminated meanwhile.

6) In order to perform a new SEG operation with different parameters, first Seg_Exit function is called to reset the module (with passing NULL pointer) and then steps 2 to 4 are simply repeated.

7) When DbsEditor application is through termination, Seg_Exit function is called the SegParam structure for final clean-up. This is necessary to de-allocate the members inside the SegParam structure in case steps 2 to 4 have never been executed during the lifetime of the application.

5. Experimental Results

In the experiments performed in this section, we used 5 sample (MUVIS) databases:

1) Corel_10K There are 10000 medium resolution (384x256 pixels) images from diverse contents such as wild life, city, buses, horses, mountains, beach, food, African natives, etc.

2) Shape Image Database: There are 1500 black and white (binary) images that mainly represent the shapes of different objects such as animals, cars, accessories, geometric objects, etc.

3) Texture Image Database: There are 1760 texture images that are obtained from Brodatz database.

4) Open Video Database: This database contains 1500 video clips, each of which is downloaded from “The Open Video Project” web site [16]. The clips are quite old (from 1960s) but contain color video with sound. The total duration of the database is around 46 hours.

5) Sports Hybrid Database: There are 200 video clips mainly carrying sports content such as Football, Tennis and Formula-1. There are also 495 images (in GIF and JPEG formats) showing instances from Football matches and other sport tournaments.

In order to present the experimental conditions, the multimedia databases and especially the test-bed platform used for experiments, Section 5.1 briefly
introduces the major MUVIS applications and their interaction over contemporary FeX-AFeX framework for content-based indexing and retrieval. Section 5.2 will then presents the SBD and SEG frameworks and particularly their major role in comparative evaluation will be introduced.

5.1. Indexing and Retrieval via FeX-AFeX Framework in MUVIS

Figure 5 shows the snapshots of two MUVIS applications, DbsEditor (left) and MBrowser (right). Open Video database is activated within DbsEditor and several features (visual and aural), SBD and SEG files were already extracted. Moreover it is now ready to perform further operations within the extended framework (FeX, AFeX, SEG or SBD) in addition the existing ones. Once a database is fully indexed with at least one sub-set of (visual or aural) features, then a query (QBE) operation can be performed over it using MBrowser application. One sample query instance performed over Sports database (over the image collection) is shown in the same figure. Note that the ranking in the query result is done from left to right, top to bottom and note further that this is an inclusive query operation since the queried image belongs to the active database and therefore, the first rank always belongs to the queried item. In this particular retrieval example, the rest of the 11 retrieved images are clearly relevant (similar) to the query image. This is the first retrieval page showing 12 items with the highest rank and using the buttons on the right-bottom side, other pages can also be browsed (for 2nd, 3rd, etc. pages). Excluding the first ranked item (the query item), we can therefore, evaluate the retrieval performance by using ground-truth methodology over the retrieved items considering a certain number per query.

As mentioned earlier DbsEditor dynamically uses all the FeX modules available during the run-time. Figure 6 shows a DbsEditor instance with Corel_10K database is loaded and the user wants to perform further FeX operations via activating the FeX dialog shown on the right side. All the available FeX modules (with their default parameters), which are binded to DbsEditor at the application start-up phase are listed in the tree-list shown in the upper side. Therefore, the user can simply choose a specific descriptor (feature), change the parameters if necessary and finally append a new sub-feature (with certain set of parameters) to the list box in the middle. See for instance in the figure, Gray Level Co-Occurrence Matrix feature with 2 parameters (i.e. GLCM, 8, 8) is already appended into the list. Once all the new sub-features, which are not already in the existing features list in the bottom (note that GLCM 8,8 is not in this list), are appended, FeX operation(s) can be started to extract all sub-features in the list. DbsEditor application will thus use the respective FeX modules to extract the sub-features requested and append them into the MUVIS database in a convenient way.

Via FeX framework, MBrowser can provide two major capabilities during querying phase: On one hand evaluation of different retrieval performances can be performed via merging several (sub-) features via (weighted) linear interpolation from the same or different feature categories. In this way a “good” blend of descriptors can be found to improve the overall retrieval performance. On the other hand, by selecting one sub-feature at a time, the retrieval performances of a set of sub-features can also be compared. Figure 7 shows a snapshot of MBrowser with FeX Query dialog is activated and 3 sub-features (GLCM 10,10, ORDC 3,4,4 and EHD7 8,11) are selected for query. Note further that the weight of EHD7 (MPEG-7 Edge Histogram Descriptor) is set as 0.5. Then the following query will be performed only over the selected sub-features with their respective weights.
In order to measure the retrieval performance analytically, we use Modified Normalized Retrieval Rank (NMRR(q)), which is defined in MPEG-7 [8] as the retrieval performance criteria. It combines both of the traditional hit-miss counters; Precision and Recall and further takes the ranking information into account as given in (1).

\[ \text{NMRR}(q) = \frac{2AVR(q) - N(q) - 1}{N(q) + 1} \leq 1 \]  

where \( N(q) \) is the minimum number of relevant (via ground-truth) items in a set of \( Q \) retrieval experiments,
R(k) is the rank of the kth relevant retrieval within a window of W retrievals, which are taken into consideration during per query, q. If there are less than N(q) relevant retrievals among W then a rank of W+1 is assigned for the remaining (missing) ones. AVR(q) is the average rank obtained from the query, q. Therefore, lower the NMRR(q) is better (more relevant) the retrieval is, for the query, q.

Accordingly, using the aforementioned capability of MBrowser for selecting one feature at a time the retrieval performances of different feature extraction techniques, each of which is encapsulated in a FeX module, can be compared. See for example NMRR plot in Figure 8 obtained from 20 queries in Shape database with the following settings: N(q)=17 and W=34. In this example the retrieval performances of four shape descriptors, 2D WAH (2D Walking Ant Histogram), MPEG-7 EHD [8] (Edge Histogram Descriptor), ARP [3] (Angular Radial Partitioning) and EPNH [2] (Edge Pixel Neighborhood Histogram) are compared with each other.

![NMRR plots of 4 shape descriptors for 20 queries over Shape database.](image1)

The following FeX/AFeX modules have been already integrated into MUVIS framework:

- HSV Color Histogram (Fex_HSV.dll).
- YUV Color Histogram (Fex_YUV.dll).
- MPEG-7 Dominant Color [8] (Fex_DoCo.dll).
- Canny Edge Direction histogram (Fex_CANN.dll).
- Angular moment histogram over closed-loop segmentation [12] (Fex_MSBF.dll).
- Texture feature using Ordinal Co-occurrence Matrices [18] (Fex_ORDC.dll).
- Texture/Shape feature via MPEG-7 Edge Direction Histogram [8] (Fex_EHD7).
- Texture/Shape feature via 2D Walking Ant Histogram [15] (Fex_2WAH.dll).

5.2. MUVIS Interaction with SBD and SEG Frameworks

Currently MUVIS is only used to test the segmentation and shot boundary detection performances of different algorithms using SEG and SBD framework. As soon as a multimedia database is indexed using different SEG and SBD modules, then MBrowser provides the necessary GUI support for the visual evaluation. Figure 9 illustrates a snapshot of MBrowser with an ongoing PQ operation over HCT. The query is a video clip with cartoon content and note that 10 relevant clips with the similar content are already retrieved. Furthermore, the segmentation masks (SMs) from different algorithms, each of which is implemented in a SEG module, are shown in the segmentation dialog, in the left side. Note that any (sub-) SEG module can be selected among the ones extracted for the active database, and thus their results (SMs) can be visually compared with each other for a particular key-frame of the video clip. Similarly, both shots and key-frames of a particular video clip can be browsed within the shot boundary dialog in the right side. The shot boundaries, the start and end frames, which are extracted by a particular SEG module are basically rendered and one can visually examine the accuracy of the shot detection by playing the clip in parallel. Therefore, a comparative evaluation of different SEG modules can be performed by examining the accuracy of both shot boundaries and the relevancy of the key-frames extracted.

The following SEG/SBD modules have already been integrated into MUVIS framework:

- Segmentation via JSEG, [5] (Seg_JSEG.dll).
- Graph Based Image Segmentation, [7] (Seg_GBIS.dll).
- Segmentation over Multi-Scale Bilateral Filtered Edge Field, [12] (Seg_MSBF.dll).
- Split and Merge Segmentation algorithm, [4] (Seg_SPMG.dll).
- Shot Boundary Detection via YUV histogram difference (SBD_YUVD.dll).
6. Conclusion

MUVIS framework is reformed to support dynamic integration of visual and aural feature extraction (FeX & AFeX), Shot Boundary Detection (SBD) and Segmentation (SEG) modules. Once the modules are implemented according to C-type template defined in the particular interface (API) then they become completely exclusive to MUVIS and the module owners can therefore develop and test them without knowing the implementation details of MUVIS applications.

The new SBD framework allows developers to build and test one or several shot boundary detection algorithms over video collections, using DbsEditor application, which stores the resultant shot boundary and key-frame information into the database for further analysis and indexing purposes. Similarly the proposed SEG framework forms a basis to implement spatial segmentation algorithms dynamically within MUVIS for images and key-frames. The resultant SMs are stored as PGM images in the database again for testing (examining and performance evaluation of each algorithm) and indexing purposes. By developing efficient FeX, SBD and SEG modules over the extended framework, performing multi-modal analysis over a MUVIS database will be feasible.

Current and planned future studies include: better visual and aural descriptors will be formed as FeX modules and integrated into MUVIS. Similarly several SBD and SEG modules will further be developed, tested and integrated accordingly.

REFERENCES


[22] Virage. URL:www.virage.com