VTApi: an Efficient Computer Vision Data Management Framework

Petr Chmelar[1,2], Martin Pesek[2], Tomas Volf[2], and Jaroslav Zendulka[1,2]

[1] IT4Innovations Centre of Excellence,
[2] Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
{chmelarp,ipesek,ivolf,zendulka}@fit.vutbr.cz

Abstract. VTApi is an application programming interface designed to fulfill the needs of specific distributed computer vision systems and to unify and accelerate their development. It is oriented towards processing and efficient management of image and video data and related metadata for their retrieval, characterization and intelligent analysis with the special emphasis on their spatio-temporal nature in real-world conditions. VTApi is a free extensible framework based on progressive and scalable open source software as OpenCV for high-performance computer vision and pattern recognition, PostgreSQL for efficient data management, indexing and retrieval extended by similarity search and geography/spatio-temporal data manipulation.

Keywords: VTApi, computer vision, data management, similarity search, API, methodology, OpenCV, PostgreSQL, spatio-temporal

1 Introduction

Ever expanding multimedia content necessitates the research of new technologies for content understanding and the development of a wide variety of academic, commerce and government applications [10]. For that purpose, we have submitted a project (referred to as VideoTerror) to the Ministry of the Interior research programme, whose objectives are to increase national security using new technology, knowledge and other outcomes of applied research in the field of identification, prevention and protection against illegal activities affecting citizens, organizations or infrastructure and against natural or industrial disasters.

The main objective of the VideoTerror (VT) project is to define, explore and create a prototype of a system warehousing image and video accomplished with computer vision and analytics based on a computer cluster. The basic requirements include image and video feature extraction, storage and indexing to enable (content-based) retrieval, summarization and characterization together with video analytics in the meaning of object detection and recognition in an interactive and iterative process.

In addition to the technology, we also target usual aspects of the research – to unify and accelerate it by choosing an appropriate design methodology and

architectural framework for the composition of domain and application specific tools focusing on open source software. In particular, we propose a solution that will enable the development and adaptation of a complex computer vision application at a reduced cost in terms of time and money. We target this goal by (re)using and integrating tool chains of (CV) methods and (multimedia) data and metadata in an arbitrary combination as simple and versatile as possible.

The VT methodology is based on the fact, that most methods of the same purpose have similar types of inputs and outputs, so there may be a base class for each of them. Moreover, the input of a process (a running instance of a method) can be seen as another process's output (e.g., annotation, feature extraction, classification) including media data creation, which has the null input. Methods are generally not included in the API self, they are created by developers using the API. In this way, methods and their chains can be created and reused on multimedia data and metadata for specific applications by providing a unified platform (API) for processing and management of both video and still image sets using custom methods. This is illustrated in Fig. 1. The VT project is not limited to a specific kind of data as Internet archives of image and video or closed-circuit television, so the framework can be used to support any CV evaluation campaign.

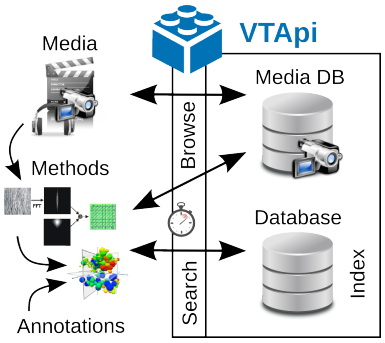

Fig. 1. The illustration of a position of the VTApi and a concept of methods' chaining.

In this paper, we present the most general part of the system and metodology – VTApi[1] (VideoTerror API), open source in C++ and Python. At the moment, VTApi is technologically based on a (remote) file-system media storage (with multimedia scraping capability) and PostgreSQL[2] database for metadata management extended by our vector-based simililarity search (distance) metrics, originaly developed for efficient local (invariant) features search (pgDistance).

[1] http://gitorious.org/vtapi/pages/Home
[2] http://www.postgresql.org/

We have integrated GEOS[3] and PostGIS[4] to be able of multi-dimensional indexing of the geography/spatio-temporal nature of real-world multimedia data acquired by (phones' and surveillance) cameras and appearing objects (trajectories). OpenCV[5] is used as the primary vision framework. In the future, we plan to integrate other technologies.

2 State of the Art

In the past decade multimedia technology has become ubiquitous. There is an instantly growing tendency of multimedia data produced by many applications in today's world. It requires to organize and manage this data and to provide support for its processing. First, image processing and data management have been a great challenge for researchers. So far, OpenCV supports only (XML/YAML) file storages, which are flexible, but not really efficient. Content-based image retrieval (CBIR) emerged as an important area in computer vision and information retrieval. Later, the support for video data management systems (VDBMS) and processing have attracted a great attention.

The need to store multimedia data in databases was reflected in SQL/MM standard. Its Part 5 Still Image provides structured user-defined types both for still images and their features that allow to store images into a database, retrieve them, modify them and to locate them by applying various "visual" predicates [9]. These data types are implemented in several commercial database products, e.g., in Oracle Multimedia and IBM DB2 Image Extender. There are also some extensions to open source database products to facilitate CBIR system development. For example, PostgreSQL-IE [6] extends the architecture of PostgeSQL. It includes more flexibility with respect to creation and definition of new feature descriptors. Such support can be understood as an elementary support for CBIR system development.

Many CBIR systems have been developed in last years[6]. One of them is Cortina [5]. Besides large scale image search, retrieval, classification and duplicate detection, it also offers the face detection, image annotation and segmentation tools and relevance feedback. Image database of the system contains images collected from the web. MySQL database is used here to store some metadata of images.

MPEG-7 standard (Multimedia content Description Interface [8]) published in 2002 has brought a standard model of multimedia content. However, most of XML-enabled and native XML databases treat simple elements as text, although their interpretation should be, e.g., time interval, vector and matrix, which the MPEG-7 defines as the extent to the XML. The MPEG-7 model has been adopted or supported by several multimedia database management systems.

[3] http://trac.osgeo.org/geos/
[4] http://www.postgis.org/
[5] http://opencv.willowgarage.com/wiki/
[6] e.g., our TrecvidSearch http://www.fit.vutbr.cz/research/view_product.php.en?id=73

For example, BilVideo-7 is an MPEG-7 compatible system to support multimodal queries in a video indexing and retrieval framework [1]. BilVideo is a representative of a video database management system, which was designed to provide full support for spatio-temporal queries. The query can contain any combination of spatial, temporal, object-appearance, trajectory-projection and similarity-based object-trajectory conditions.

MPEG-7 Multimedia Database System (MPEG-7 MMDB) [4] is another example. It is based on extensibility services of Oracle 10g. It maps MPEG-7 schema types to database types and introduces new indexing and querying system, a query optimizer, and libraries that simplify application development. The core of the system includes Multimedia Indexing Framework, which provides various index structures as SR- and SS-trees and LPC-files for fast execution of similarity and exact search.

The above system adopts the Generalized Search Tree (GiST) framework [7] originally developed by Hellerstein et al. It is an index structure supporting an extensible set of queries and data types to be indexed in a manner supporting queries (operations) natural to the types. It is unifying structures such as B+ trees and R-trees in a single piece of code and opening the application of search trees to the general extensibility. It has been adopted by PostgreSQL and our project too.

3 Concepts and Specifications

The VTApi is an API to unify and accelerate the development and evaluation of various computer vision applications. For that purpose, we have introduced a set of terms based on best practices both in computer vision and data management:

- *Dataset* is a named set of (multimedia) data along with metadata (descriptive data). Datasets can be organized hierarchically, i.e., one may be based on several others. Each dataset contains sequences.
- *Sequence* is a named ordered set of frames referred to as *Video* or *Images*. The ordering of frames in video is time-based. There may be their intervals defined for a sequence.
- *Interval* is any subsequence of *Video* or *Images* whose elements share the same metadata. For example, it can be a video shot or any sequence of frames containing the monitored object in the video or scene. Metadata of an interval are created by a process.
- *Process* (task or operation) is a named run of *Method*. *Method* defines the custom algorithm and the structure of metadata consumed and produced by a *Processes* – a running instance of this type. *Process* then defines (inserts and modifies) data according to its inputs (created by other processes, media data) and it represents all activities of the proposed framework. Implementation of a specific method is generally not included in the API, it is created by developers using the API.
- *Tag* is an indexing term representing an ontology class (in hierarchy). Tags are assigned to the multimedia data as description or annotation of a scene, object or action.

– *Selection* is a subset of logically related metadata, appropriately chosen, so that operations (processes) are effective and allow the natural chaining of processes (input, output of a process or media data). Common examples of selections are *Interval* and *Tag*. This concept is related to the effective implementation and access to the metadata in the database.

In VTApi, all these concepts are mapped to classes as it is shown in Fig. 2. Our approach is not based directly on MPEG-7 XML descriptors and description schemes, because they do not provide the flexibility of an efficient streaming and database storage and they are tree structured. Thus, we focus more on the BiM (Binary Format for MPEG-7). Assuming data in the binary form, we generally support all structures of descriptors including operations as the first order temporal interpolation, spatial transformation and coordinate mapping, including their indexing using GiST and GIN [7]. The same states about MPEG-A (Multimedia application format) Part 10: Surveillance application format.

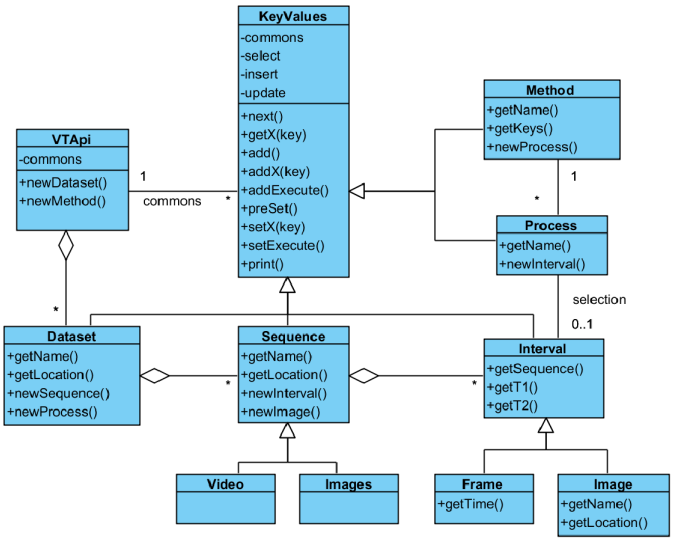

Fig. 2. The simplified class diagram of VTApi.

3.1 Data Model

The very simplified class diagram of VTApi is illustrated in Fig. 2. It follows the concepts given in the previous section and operations that logically belong to. Most classes inherit from *KeyValues* that provides the basic operations needed to manage key-value pairs (associative array), on which the VTApi model is based. *KeyValues* class is the crucial to ensure the functionality and generality

of the API by the main function *next*(), which performs most operations of the API, except constructors. It performs the database queries similarly to JDBC. It also allows to change the values of object's variables and commits the values changed using setters and it inserts values using our adders methods. Moreover, it uses the lazy approach, hiding the functionality necessary, doing it efficiently when needed by using caches and batches if possible.

The VTApi is strongly typed, the following description uses notation of X referring to any data type implemented (integers, floating points, strings, 4D geometry points, lines and polygons and their structures, vectors, arrays and (OpenCV) matrices). For instance, *getX*(k) or *setX*(k,v) operates key k and its value v of type X.

The entry point to the application is the *VTApi* class based on the config file and command-line arguments[7]. All other classes, denoted as C, are derived from *KeyValues*, and they inherit its operations and attributes:

- *Commons* class provides a very basic functions such as loading configuration file and command line parameters (using GNU Gengetopt), it provides a connection to the database (PostgreSQL), a data storage (remote file system) and it uniformly manages error reports and other statements (log). *Commons* is a shared object, usually created by the *VTApi* class.
- *Select* class is used to construct queries that after the first call of the function *next*() retrieve information from the database. There are special functions to simplify the construction of queries, so that it is basically mostly satisfactory to use the constructor for most of them. Other functions simplify the work with selections, keys and their values to filter queries, use functions and indexes.
- *Insert* class provides insertion of defined data where possible using the function *addX*(k,v). There are 2 general ways of inserting – immediate (*addExecute*()) or batch (implicitly) by calling *next*().
- *Update* class similarly allows the modification of values of the current element using the the typed family of functions *setX*(k,v).

Classes derived from *KeyValues* contain only a minimum of functionality programmed, so it is easy to create a new derived class C if needed. Above all, they take care of consistency of data and provide simple accelerator functions. For instance, *getName*() for their identifier or *getLocation*() for the physical data location (e.g., a dataset or a directory with pictures) and *newC*() methods, that show the developer the natural flow of the program (represented as aggregations in Fig. 2). For example, invoking *newSequence*() method of the *Dataset* class object creates a new object of class *Sequence*, with all necessary parameters, in which we can call the *next*() to access all the current dataset's sequences identified by *getName*(). This is illustrated in the sample code in Sect. 4.1.

The database model presented in Fig. 3 is here for completeness. Only attributes (columns) that are necessary for the VTApi functionality are shown there. During the development, the schema is changed using the *Method*'s data

[7] See the Wiki at http://gitorious.org/vtapi

definition capabilities. When running a *Method*, a new process is created and column(s) are added according to the keys required. One can decide to create a new *Selection* or to use an existing one (*Intervals* by default). One *Method* can be run (e.g., with different parameters) on various data as a different process. A process stores its metadata in the same format, so chaining is simplified to specifying input process name.

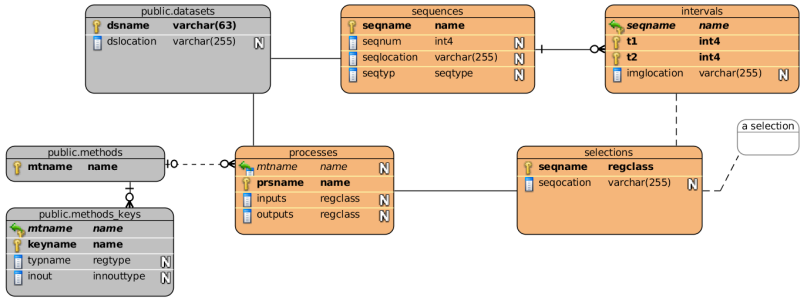

Fig. 3. Illustration of the minimum logical databases ER model. Orange tables are specific to a distinct dataset (database schemes), while public tables (grey) are shared across all datasets. The white table denotes a selection, which can be used either as the storage in non-SQL databases (using modified *KeyValues* class) or for storing different data, as trajectories or tags.

4 Use Cases

We have chosen two simple use-cases demonstrating the use of VTApi for some common tasks based on TRECVid evaluations and an object trajectory extraction and transformation experiment. The first use-case is a simple CBIR and the second presents trajectory clustering followed by the performance experiment.

4.1 Content-Based Image Retrieval

The OpenCV library provides powerful feature extraction and classification techniques. However, it doesn't have capabilities to store the data to be efficiently searched and processed further. This is especially useful for tools like Google Image Search or developed within TRECVid [10], where the retrieval is based on multiple types of (low to high-level, local and global) features related to any object (mask) together with annotation (tags) based on the image description.

Thus, we have implemented various similarity-based distance functions. The pgDistance extension (included in the VTApi code) performs the similarity queries measuring distances of feature vectors in PostgreSQL database, e.g., cosine or Euclidean distance. So that we can employ the feature-based similarity

search supported by efficient indexing techniques (GiST and GIN [7]) as filter methods – Heap and Bitmap indexes, R-Tree, Inverted Document Index and other general indexing structures, supporting containment and nearest neighbor search on vectors (using `@>` and `<->` operators). However, the example in Fig. 4 is quite simple. Assume you have a large dataset of images called "search" already populated in the database, and you want to perform MPEG-7 Color layout descriptor based CBIR.

```
// VTApi entry point, using Dataset "search"
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("search");
// code of the ColorLayout Method, using Selection "image"
Image* image = new Image(&dataset, "image");
while (image->next()) {
    vector color = colorLayout(image->getDataLocation());
    image->setIntA("color", color);
}
// retrieve Image(s) according to their similarity to "Q.jpg"
Image* nearest = new Image(&dataset);
nearest->select->from("image", "distance_square_int4(color, "
    + toString(colorLayout("Q.jpg")) +")");
nearest->select->orderby("distance_square_int4");
nearest->next();   // is the most similar image
```

Fig. 4. A simple CBIR code example.

4.2 Object tracking, trajectory querying and analysis

In the surveillance video, it is important to be able to track moving objects and to extract their visual and spatio-temporal features. Such extracted metadata then should be cleaned, stored and indexed to be able to query and analyze it.

Object tracking is a complex task, especially in crowded scenes. It is possible to use various object trackers, for example, OpenCV blobtrack demo that can be extended with feature extraction as in [3]. The outputs of such methods include spatio-temporal locations in the form of trajectories, blobs and other features of moving objects, which can be aggregated, summarized, analyzed and enriched with additional features such as annotations, tags or classes.

All the above features might be used and searched for similarity by VTApi. A trajectory query may relate either to relationships between moving objects or a specific spatio-temporal region. Such an analysis can be performed both on VTApi clients and server, because we have adopted the OpenGIS GEOS library, that has been adopted by PostGIS. In order to perform these operations efficiently, VTApi adds a binary access to geometry types and n-dimensional

cubes that are used as spatio-temporal minimum bounding boxes of (moving) objects.

Many data mining and machine learning techniques can be also performed on moving objects metadata. Such an analysis may involve trajectory clustering, classification, object recognition, outliers detection and so on. The following example shows a clustering of trajectories using VTApi and an OpenCV implementation of Expectation-maximization (EM) algorithm, which estimates parameters of a Gaussian mixture model (GMM) [2]. First, feature vectors representing trajectories are read from the database and training samples for the EM algorithm are prepared (see Fig. 5). Suppose that trajectories are stored in selection "tracks" in this example. Second, GMM is trained by the EM algorithm and appropriate cluster labels are stored in the database (see Fig. 6).

```
Mat samples; // cv::Mat of training feature vectors
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("train"); // training dataset
dataset->next();
Sequence* sequence = dataset->newSequence();
 while (sequence->next()) { // for each video
    Interval* track = new Interval(*sequence, "tracks");
    while (track->next()) { // for each trajectory
        Mat sample; // cv::Mat for feature vector of trajectory
        float feature = track->getFloat("feature");
        // ... read features and fill feature vector
        samples.push_back(sample);
    }
}
```

Fig. 5. Sample code of reading trajectories and preparing training samples.

We performed the trajectory clustering on a set of trajectories extracted from the second dataset of videos forming the i-LIDS dataset[8] used for NIST evaluations and it comes from five cameras at the LGW airport. An example of visualization of some obtained results is shown in Fig. 7. Different colors of trajectories refer to different clusters. On the left, there is a result of clustering trajectories from the first camera using the EM algorithm mentioned above. On the right, there is a result of clustering trajectories from the third camera by the K-means clustering algorithm to show the easy changeability of methods of the same purpose. We have prepared also an outliers analysis within the VideoTerror project.

[8] http://www.homeoffice.gov.uk/science-research/hosdb/i-lids/

```
CvEM model, labels; // GMM-EM model and cluster labels
CvEMParams params; // EM parameters
// ... set EM parameters including number of clusters
model.train(samples, Mat(), params, labels);
// ... choose dataset and sequences according to sample code above
while (track->next()) {
    Mat sample;
    // ... read features and fill feature vector
    int cluster = (int) model.predict(sample); // get cluster label
    track->setInt("cluster", cluster); // store cluster label
}
```

Fig. 6. Sample code of training GMM and storing cluster labels.

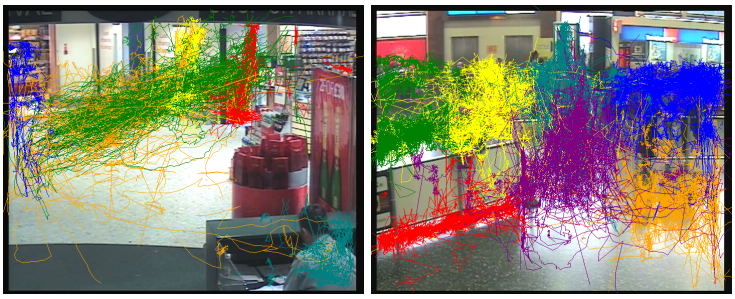

Fig. 7. Examples of trajectory clustering results obtained by EM algorithm on trajectories from the first camera (left) and by k-means algorithm on trajectories from the third camera (right).

4.3 Real-time tracking and trajectory indexing

Because processing trajectories often relates to the real-time, we have performed experiments focused on how much time and resources are needed for the trajectory management. In the experiment (see Table 1), we transform the OpenCV's blobtrack trajectories, we store them in the database as vectors capable of the first order temporal interpolation even stored as discrete points, and we index them using 3D bounding-cube and GiST (bitmap index), so that they can be retrieved very efficiently. The similarity query was performed by the containment operator (`@>`) returning 4 trajectories contained in a spatio-temporal bounding box (selected randomly).

At this simple demonstration we dealt with 7269 trajectories, tracked of about 4 hours of video (49:17 minutes by 5 cameras in parallel) with a very crowdy airport trafic of the i-LIDS dataset. According to the table, we show that this system can eventually run in real-time both on: (a) 13" notebook (dual 1.4 GHz ULV processor, 2 GB RAM, 128 GB SSD) including both the trajectory processing part and the local database and (b) the same machine connected using

Table 1. A simple performance test of insertion and querying trajectory data.

	insert/update all	select all	find similar (4)	total network data
(a) local	220 ± 1 s	43 ± 1 s	0.1 s	0
(b) remote	220 ± 5 s	74 ± 1 s	0.2 s	334 MB

Ethernet network to a remote VTApi database server and the network delivery time must be taken into consideration in favor of the server hardware.

5 Conclusion

In the paper, we present an innovative open source computer vision data and metadata management framework we offer to the public. The main advantages of the proposed API is the reduction of effort and time to produce high-quality distributed intelligent vision applications by unified and reusable both methods and data sets of video, image, metadata and features on all levels. Above that, we offer the novel data and methods interfaces and methodology to be used by researchers and developers of both academic and commercial sectors to collaborate and chain their efforts.

We have selected, integrated and extended a set of progressive and robust open source tools to be efficient for multimedia data and related metadata storage, indexing, retrieval and analysis. The system uses the best from (post)relational databases, it offers unlogged, alternative storages and data structures we need to manage and some others (graph databases) to make the data access more efficient, especially for rapidly changing geography/spatio-temporal data of a very complex nature in binary form, that can be now processed both on VTApi clients and in the database. During the development, we were aware that intelligent vision specialists are not familiar with databases, so we hid the concept and most operations are made even more naturally than working with XML in any form – using *next*(), getters and setters. However, if operations as the data transformation, replication, storage cascades or warehousing are needed, they are still available, because it is open source.

There will always be impossibleconsciously manipulating all pixels of massive real-time video streams, but that's why there is MPEG and OpenCV. In the future, we will supplement more of their activities. Also, we will extend VTApi with the MPEG-7 XML Library[1] to enable a standardized framework for methods' performance evaluation, the KALDI[2] audio and speech processing framework, and some other capabilities especially to enable data transformation and analysis. At the moment, we focus on making the VTApi broadly usable for real-time, real-world and real-need intelligent vision systems.

[1] http://iiss039.joanneum.at/cms/index.php?id=80
[2] http://kaldi.sourceforge.net

Acknowledgments. This work was partially supported by the research plan MSM0021630528, the specific research grant FIT-S-11-2, the VG20102015006 grant of the Ministry of the Interior of the Czech Republic and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

1. Bastan, M., Cam, H., Gudukbay, U., Ulusoy, O.: BilVideo-7: An MPEG-7-compatible video indexing and retrieval system. IEEE Multimedia 17, 62–73 (2010)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
3. Chmelar, P., Lanik, A., Mlich, J.: SUNAR: Surveillance network augmented by retrieval. In: Blanc-Talon, J., Bone, D., Philips, W., Popescu, D., Scheunders, P. (eds.) Advanced Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science, vol. 6475, pp. 155–166. Springer Berlin / Heidelberg (2010)
4. Döller, M., Kosch, H.: The MPEG-7 multimedia database system (MPEG-7 MMDB). J. Syst. Softw. 81(9), 1559–1580 (Sep 2008)
5. Gelasca, E.D., Guzman, J.D., Gauglitz, S., Ghosh, P., Xu, J., Moxley, E., Rahimi, A.M., Bi, Z., Manjunath, B.S.: CORTINA: Searching a 10 million + images database. Tech. rep. (Sep 2007), `http://vision.ece.ucsb.edu/publications/elisa_VLDB_2007.pdf`
6. Guliato, D., de Melo, E., Rangayyan, R., Soares, R.: POSTGRESQL-IE: An image-handling extension for postgreSQL. Journal of Digital Imaging 22, 149–165 (2009)
7. Hellerstein, J.M., Naughton, J.F., Pfeffer, A.: Generalized search trees for database systems. In: Dayal, U., Gray, P.M.D., Nishio, S. (eds.) VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland. pp. 562–573. Morgan Kaufmann (1995)
8. Kosch, H.: Distributed Multimedia Database Technologies: Supported MPEG-7 and by MPEG-21. CRC Press, Boca Raton (2004)
9. Melton, J., Eisenberg, A.: SQL multimedia and application packages (SQL/MM). SIGMOD Rec. 30(4), 97–102 (Dec 2001)
10. Smeaton, A.F., Over, P., Kraaij, W.: Evaluation campaigns and trecvid. In: MIR '06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval. pp. 321–330. ACM Press, New York, NY, USA (2006)