I/Browse: The Bellcore Video Library Toolkit

Paul England, Robert B. Allen, Mark Sullivan, Andrew Heybey,
Mike Bianchi, and Apostolos Dailianas [i]

Bellcore
445 South Street, Morristown NJ 07960

Abstract

I/Browse: The Bellcore Video Library Toolkit is a set of tools for constructing and browsing libraries of digital video. The toolkit is designed to work with video libraries on local or network disks, CD-ROMs or a multimedia server. There are three main components, a preprocessor, a tagger, and a browser. Particular attention is focused on the tools and techniques we have developed to rapidly tag videos. The tagging system allows text fields, type information, and other resources to be associated with frames in the video. The tags are further organized into a hierarchical 'table of contents', which is suitable for browsing and searching.

1. Introduction

The enabling technologies for widespread deployment of video to desktop PCs and workstations are high-bandwidth networks, low cost mass-storage, efficient video servers, and capable video codecs. All of these technological requirements have been met or have become much cheaper in the past few years. Therefore, we expect networked video services to become as ubiquitous as network file and print services.

An obvious use of networked desktop video is to access corporate video archives. For this type of application, tools are also required to index, browse and search the video library. This paper describes a set of tools that addresses the problem of creating and browsing libraries of digital video. This is part of a larger effort at Bellcore to provide low-cost, widespread deployment of digital video and audio to desktops in an organization. A related paper describes a multimedia server system for providing an inexpensive and scaleable repository for the video source material.[1]

The amount of information contained in a videotape is enormous: there is spoken word, text, images, objects, and motion. Unfortunately, searching on any of these keys is very hard. Furthermore, even if a perfect transcript of a videotape can be obtained (e.g., from a closed-caption feed), the transcript does not necessarily convey the 'structure' of the video sequence. For example, there are clear boundaries between stories in a news feed that are not apparent from the text. What is needed to search a video library effectively is an index that can support a flexible textual description of the video. Additionally, to allow browsing the index should be hierarchical so that a viewer can get an overview of the content and then focus on the material of greatest interest.[2]

We have developed a tagging scheme that attempts to solve both of these problems. Individual tags contain arbitrary text fields, type information (*'speaker'*, *'scene description'*, etc.), and links to external documents. Furthermore, these tags can be arranged into a tree-based hierarchy. In limited circumstances, some of these tags can be extracted automatically from a video feed, but in most cases manual authoring is required or is, at least, desirable. We have also developed tools to assist rapid manual tagging of video sequences. The tools give hints where tags may be required, and they present an overview of the video content in various ways to make the tagging process as quick and as painless as possible.

2. System Architecture

I/Browse consists of three tools: the *Video Preprocessor*, the *Video Tagger*, and the *Video Browser*. The tools work in conjunction with libraries on CD-ROMs, networked file-systems, or a video server. The majority of this paper focuses on the indexing or tagging scheme we have designed and the tools we have developed to automate the tagging procedure. To provide context, we will also describe the video storage and distribution architecture.

[i] The authors may be reached at {england, rba, sullivan, ath, bianchi}@bellcore.com and apostolo@cs.columbia.edu

2.1 Overview of Software Components

Figure 1 shows the overall system architecture for the production and distribution of the digital video library. The capture and compression stages can be performed with off-the-shelf tools. Following video capture and compression, the *Video Preprocessor* constructs a low-level frame-by-frame index of the video source material. The index it constructs contains information such as the average luminance, contrast, and the audio frequency spectrum of each frame. This index is used by the *Video Tagger* and the *Video Browser* for two purposes. First, the raw index values are valuable in semi-automatically identifying scene changes, and second, they can be used for scene comparisons in query-by-visual-example searches.

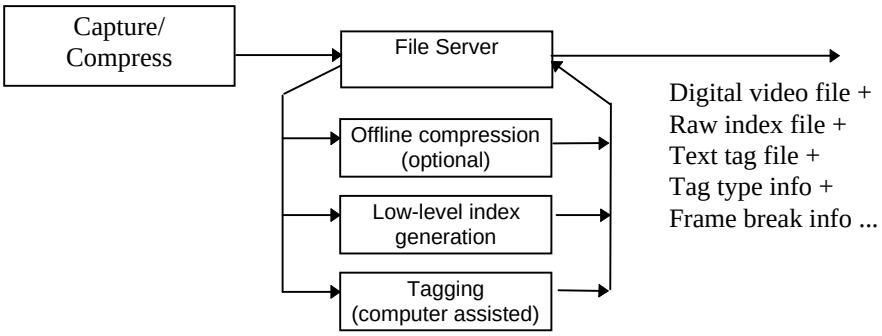

Figure 1: *The tagging process. A live or recorded video feed is digitized and stored. In subsequent steps, the video is compressed again (optional), a low-level index is generated, and the video is tagged. The resulting video and index files can be used directly or loaded onto a video server and database management system.*

The *Video Tagger* is designed to facilitate the production of a tree-based index or table-of-contents of the source video in which the index entries are tagged to individual frames in the video sequence. The index can contain any text together with frame type information (e.g., *'Viewgraph'*). The *Video Preprocessor* and *Video Tagger* are both used in the production of a video library. The *Video Browser* is an example of a tool for viewing and searching video these video libraries. The *Video Browser* takes as input the tree based index generated by the *Video Tagger*, the raw indices generated by the *Video Preprocessor*, and the original digital video file. The *Video Browser* permits the user to quickly navigate and examine video clips using the text-tag index. It also permits text searches (e.g., "ATM and cell loss") as well as simple visual query-by-example searches. The *Video Browser* also supports libraries of video information. A library is a database of videos on similar topics. The browser permits text and image searches of individual video sequences, libraries, and collections of libraries. The next section describes the video storage and distribution architecture. The following sections describe in more detail the indexing scheme and how the various components are used in the construction and presentation of a video archive.

2.2 Video Storage and Distribution Architecture

The toolkit components are designed to be flexible in their requirements for storage media. For example, small video libraries can be cost-effectively stored and distributed on CD-ROMs with readers directly attached to the client machines. To produce a large database or a centralized database accessible to tens or hundreds of simultaneous users, a video server is the cheapest approach. To accommodate these diverse requirements, the tools are designed to work with one of two kinds of multimedia data access - 'client-pull' or 'server-push.'

First, Client-pull provides reliable file-system-like, semantics to the applications. The client may assume that the connection is reliable, and that a seek in the file results in the next access starting at the modified file offset. However, although best-effort response time can be expected, packet loss, file-system overload, or other forms of contention can result in data starvation for the client. To ameliorate these problems, the client video and audio widgets use two strategies. First, the clients maintain read-ahead buffers. If the client applications notice that the buffers are becoming empty, they seek ahead in the files, and attempt only to play the audio data. This selective, dynamic, dropping of frames is reasonably

effective in reducing the problems associated with network congestion, but is not necessarily useful if the server itself is overloaded.[i]

Second, server-push widgets assume an unreliable remote data source (a video server) sourcing packets at precisely the consumption rate required by the video and audio streams. The user can seek in this stream, but the next arriving packet may not necessarily start at the new position in the file (because of packets in transit). The video decoders used in this case must be robust to packet loss, and packet loss should not result in severe video or audio artifacts. The advantage of such a system is that a lost packet may be, in some circumstances, better than a late packet. For example, congestion control mechanisms for TCP/IP can slow the data transmission rate to below that acceptable for audio.

For the browser applications, the client-pull, and server-push widgets are interchangeable depending on the data source. This means that the browsers can work from a network file system or CD-ROM with no additional support, or can work as part of a larger video-server installation. However, even those applications that use server-push semantics also require a reliable client-pull connection for less time-sensitive activities like rendering thumbnail bitmaps.

Finally, in the current implementation, searching for text or like frames is the responsibility of the client application. For wider deployment, the search tools and algorithms that we have developed in the browser will be moved into a remotely accessible database management system.

2.3 The Preprocessor

The *Video Preprocessor* reads and decompresses every frame in a video sequence and applies a number of transformations that yield a scalar or one dimensional vector quantity for each frame. These metrics will be used by the *Tagger* and the *Browsers* for automatic scene identification and visual query-by-example searches. The index format used is extensible. The current version of the preprocessor extracts the following metrics from the video frame sequence:

Scalar:

```
Luminance
Contrast
Red Intensity
Green Intensity
Blue Intensity
Magenta Intensity
Cyan Intensity
Yellow Intensity
Absolute Frame Difference
Frame Difference
Color Histogram Difference
Audio Volume
```

Vector:

```
Audio Power Spectrum
```

In all cases except the audio power spectrum, a single scalar value is extracted per frame. For the audio frequency spectrum, a one-dimensional vector is extracted. The preprocessor (and the raw index file it generates) are designed to be easily extensible so that users may add their own metrics. For example, the user may know that important scene changes are accompanied by motion in the top left corner of the frame, so a specialized filter could be constructed to identify this behavior. Similarly, more filters which extract vector information may be added such as the coefficients of a partial quad-tree decomposition of the frame for texture based queries.[3]

It is useful to extract a large number of parameters at this stage (rather than a select few) because different video sequences are best indexed by different metrics. For example, simple cuts between video sequences are easily identified by frame difference measurements, cuts between speech and music can be (visually) identified with a glance at the frequency spectrum, and plays in a football game are identifiable by brief periods of motion. The preprocessor is not designed to run in real time. For 15 frames-per-second 320 x 240 video (a typical frame rate and resolution for software decoded desktop video today), the *Video Preprocessor* runs at approximately 1/3 real-time on 66 MHz Pentium. Thus, real-time preprocessing of this nature will be realizable with slightly faster machines and some optimizations.

---

[i] The effectiveness of this procedure depends on the bottlenecks in the system. If disk contention and overload is a problem, seeking ahead sometimes does not help. Often with substantial disk read-ahead, the same amount of data needs to be read from the disks at the same rate (that required by the audio stream). Dropping video frames lessens the demands on the network but *not* the disks.

Other preprocessors could be also be run at this stage. For example, a text-based index could be constructed by running an optical character recognizer on the source video, or speech recognition software could be used to construct a transcript. Additional indices merely require text tags and timestamps to be incorporated into the I/Browse framework.

2.4 The Video Tagger

2.4.1 The Frame Tagging Scheme

The main purpose of the *Video Index Tagger* is to facilitate the production of an index that tags frames in the video sequence with textual information. These tags can be subsequently used by a video browser for presentation and for text searches. The text tags are supplemented with an extensible type system. The type tags can be used to narrow down text searches, and are also useful for picking out relevant still frames (e.g., viewgraphs) in a video sequence. The Tagger allows any or all frames be tagged with two text fields and a type field. The two text fields are a *Title Field* and a generic *Text Field*. The pre-defined frame entry-types are:

```
Index Entry
Spoken Word
Written Word
Viewgraph
Scene Description
Audio Description
Speaker
```

More than one type can be applied to a given tag. Other types may be added easily.

The index tags may be entered manually using the indexing tool, or they may be imported from an external file which includes text and timestamps (e.g., a file generated from a closed-caption feed). Any amount of textual detail may be contained in the index tags - the more information that is provided, the more sophisticated will be the searches that are possible. In addition, the frame tags (and hence the underlying video) may be organized into a hierarchical 'table of contents'.[2] The table of contents supports up to twenty levels of index entry in addition to 'leaf' entries, which can exist under any index level.

The table of contents tree provides an overall hierarchy for the chronology of the video sequence, whereas the leaves of the tree be used to contain detailed descriptive information. Thus, the non-leaf entries behave like a table-of-contents, whereas the leaves behave like an index. For example, the text index for a football game might look like this:

```
[0] Super Bowl 2020
     [1] Player Introduction
     ...
     [1] Coin Toss
     [1] First Quarter
          [2] First Play - Kickoff
               [L-Scene Description] NY Jets Kickoff
               [L-Play Type] Kickoff
               [L-Offense] NY Jets
               [L-Player] John Doe II
               ...
          ...
          [2] Third Play - First Down
               [L-Play Type] First Down
```

In this example, nodes marked **[0,1,2]** are tagged *'Index Entry'*. The leaves **[L-*]** show the type of the '*leaf*' tag (*Player name*, *Scene Description*, etc.) where the extensible tagging type system has been used to add new types that are useful for describing this particular type of content (a sporting event). Specific client applications are free to interpret and use the index structure in any method they see fit, but we have found that video browsers that present the non-leaf entries as an overview (a table of contents), and reserve the leaf entries for text searches work quite well (e.g., Fig. 3). Note that the indexing scheme we use is quite flexible, but is not perfect. For example, it is easy to tag much of the important visual or audio content of a sequence, but it is quite hard to have more than one *hierarchy*. In the football example above, one hierarchy might be constructed based on possession of the football, another by quarter, and yet another based on action as

opposed to commercial breaks. Such organizations *can* be accommodated using our framework, but other approaches could probably improve upon it.

Finally, apart from the text entries, other external (file-based) resources can be included in the tags. Again, the use of these external resources is the prerogative of the browser application, but obvious examples are bitmap or graphic metafiles for high-quality still images or viewgraphs, URLs (universal resource locators) for external links, and flat-text versions of transcripts.

2.4.2 Philosophy of the Video Tagger

There has been considerable progress on the fully automated generation of indices for video and audio sequences (see Section 4 for an overview). Unfortunately, although scene change recognition is quite advanced, to construct a useful index and table of contents, it is necessary to 'understand' the video content. Automatic 'understanding' is possible in a restricted fashion in certain limited cases - for example, a technical talk punctuated by viewgraphs, or a conversation in which the camera moves from one speaker to the other can be broken down into useful components. However, in most cases, it is a very difficult problem.

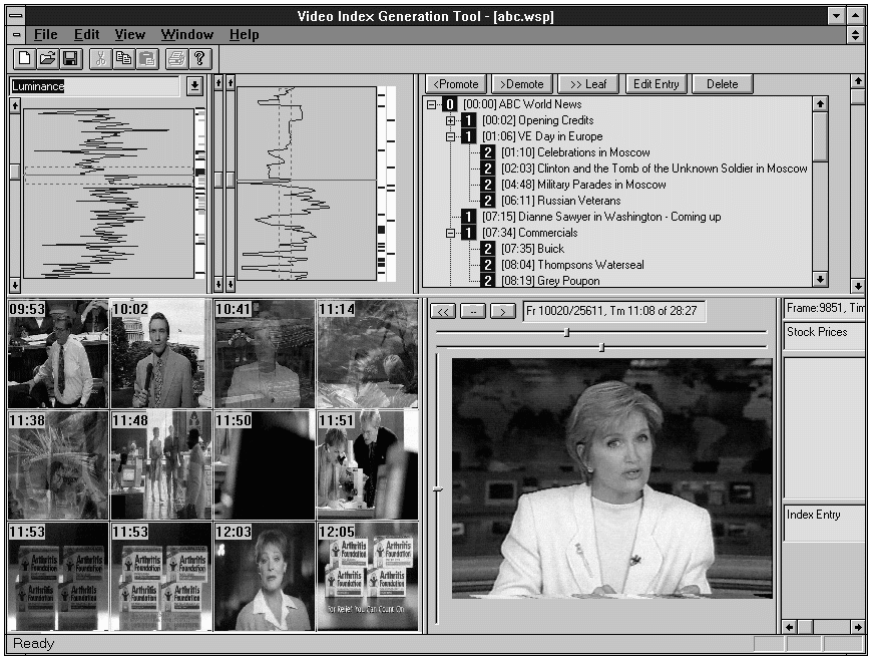

Figure 2*: Screen dump of the Tagger. From top left to bottom right, the panes in the interface show (1) Selected image metric (luminance here) for the whole movie, (2) zoomed-in version of pane 1, (3) the tag tree being constructed, (4) thumbnail view (here, frames that meet the threshold criterion in 2), (5) the video player, and (6) the current text tag.*

The *Video Tagger* adopts a different approach to full machine understanding of a video content. We believe there are large classes of applications for which it makes sense to annotate manually (or at least semi-automatically) the video content using the frame tagging scheme described above. The *Video Tagger* makes this process as simple as possible by giving the operator 'hints' where tags maybe useful, and providing a tool which makes it easy to scan the video rapidly and add and edit tags manually.

In general, it is very difficult to get an overall picture of a video sequence without actually watching it, and of course this makes the whole process time consuming. It is impossible to get around this problem completely, but the *Tagger* offers a number of facilities to ease this predicament. One fundamental reason why is difficult to obtain an overview of a video sequence 'at a glance' is that a video sequence is inherently three dimensional (two spatial dimensions (X, Y) and time (T) - ignoring the soundtrack), whereas the display devices we have are typically only two dimensional (e.g., a CRT). We have

taken a number of approaches to try to solve this problem. In order to present the user with a sequence overview on a two dimensional CRT screen (without using the temporal dimension), some contraction of spatial dimensions must be performed.

2.4.3 Video Tagger Interface

The *Video Tagger* transforms the three-dimensional source material (a video sequence) into a form appropriate for a two-dimensional display device (the CRT). We look for transformations of the original video sequence that contract one or two of the dimensions. For example, each frame maps to an average scalar quantity (e.g., luminance) or a one-dimensional vector quantity (e.g., the audio frequency spectrum). These quantities can be plotted as a function of frame number, and the user can take in some subset of the information in a whole video sequence 'at a glance'. The top left two panes of Fig. 2 demonstrate this: The top left pane is a graph of the image metric (luminance in this case) plotted against time (running down the Y-axis). The adjacent pane is a zoomed version of the full movie, selected by the boxed region in the left-hand frame. Clearly, the amount of information thrown away in such dimensional contractions is enormous (mapping a still image to a single scalar quantity). To improve things slightly, we extract many such quantities (12 in this release) and it is easy to add more. However, it is still not possible to extract 'at a glance' information about the video sequence by viewing (for example) the 'Contrast vs. Time' graph so the *Video Tagger* performs two further operations on this raw index information.

Display Metrics: Most video sequences consist of sub-sequences edited together. Most of the sequence breaks show up as spikes or discontinuities in the raw scalar index quantities (e.g., a sequence break is often associated with a change in the average luminance that exceeds the inter-frame luminance changes within a sequence). To exploit this behavior, the *Video Tagger* semi-automatically finds these frame breaks, and can show representative video frames from each of the sub-sequences (pane 4 of Figure 2). The procedure for identifying these breaks is semi-automatic in the sense that the user selects a threshold for the scene change identifier, and the system identifies scene changes and renders thumbnail images for the threshold set. This is the box in pane 2 of Figure 2. Level crossings of the box are considered scene-changes by the application. Because scene fades are often slow, the system does not use a simple level-crossing, but incorporates hysteresis. Additionally, some metrics identify a scene change by a discontinuity in the metric itself, and others by a discontinuity in its derivative. To allow scene change identifications with the second type of metric, the user can select a derivative plot. An example of where this approach works particularly well is in the tagging of a videotaped technical talk. Here, the video cuts back and forth between viewgraph and speaker. These changes are easily identifiable using almost any image metric; the user can set an appropriate hysteresis, and will automatically see still frames of all the viewgraphs. As a second example, for the football game described above, 'green' is an effective measure for distinguishing the playing field from (say) commentary or commercials.

Further refinements of this idea that we have adopted include restrictions of the sense of the crossing. In the technical talk example, if bright frames tend to be viewgraphs, setting upward-crossings-only displays just the viewgraphs as still frames, allowing upward and downward crossings will show the speaker alternating with the viewgraphs.

Similar Frames: Sometimes the viewers can identify a frame that they know will occur again that marks an important sequence break. Examples of this might be a station identification frame in a broadcast television feed before a break to advertisements or a picture of a viewgraph in a taped lecture. The viewer can exploit these 'distinctive frames' by asking *Video Tagger* to display 'similar frames' to the present frame.

The procedure for identifying similar frames that the *Video Tagger* currently supports is quite simple, but often useful. The abstraction is that each frame can be identified by its position in the 12 dimensional space defined by the parameters extracted by the *Preprocessor*. Further, two frames are similar if the Euclidean distance between the points representing two frames is small. To allow for some customization, users can choose those dimensions that they think are important or distinctive. The similarity metric is presented in an identical fashion to all of the other elementary image metrics: it can be selected for plotting in pane 1 and 2 of Figure 2, and the level crossing can be adjusted until a reasonable signal-to-noise ratio is seen in correctly identified similar frames. When this works (not all frames are sufficiently distinctive for the simple measures of similarity to be useful) it can be a very valuable feature for a first pass at breaking a video sequence into more manageable sections.

2.4.4 Building a Hierarchy

There are many ways for the user to insert a new tag, for exapmle, double-clicking with the mouse on an appropriate image will automatically pop-up a dialog box to insert a tag on the frame shown. Using these techniques (and other features

offered by the tagging tool), a fairly extensive temporal textual representation of a video sequence can by built quite quickly. We estimate a useful index with about 50 tags on a 1-hour video clip takes about two hours. More effort can be expended for more detailed indexing.

Building the hierarchy (turning a flat list into a tree by promoting and demoting index entries) can be done at several stages in the tagging. For the technical talk described above, a first pass could be to index all of the viewgraphs. With this partial index, a knowledgeable operator can quickly identify changes in the *content* of the material being discussed (as opposed to simple visual changes), and impose a content-based hierarchy on the video. The tagging tool allows simple promotion and demotion of tags to form the tree.

There are several simple options and configurations of the tagging tool that speed and simplify tagging a video. For example, all of the frames in the tool are intimately connected. If a video is playing, the current frame will be shown continuously updated in all of the other frames. Similarly if the user clicks on a thumbnail, an index entry, or an interesting spot in one of the image metrics, the video will seek to that point and start playing. Additionally, many options are allowed for many of the panes: The metrics can be plotted linearly, logarithmically, or smoothed. The thumbnail sketches can be tied to frame breaks, existing index entries or just time steps though the video, and the video itself can play fast or slow, forwards or backwards. We have found that all these and other options greatly speed up tagging.

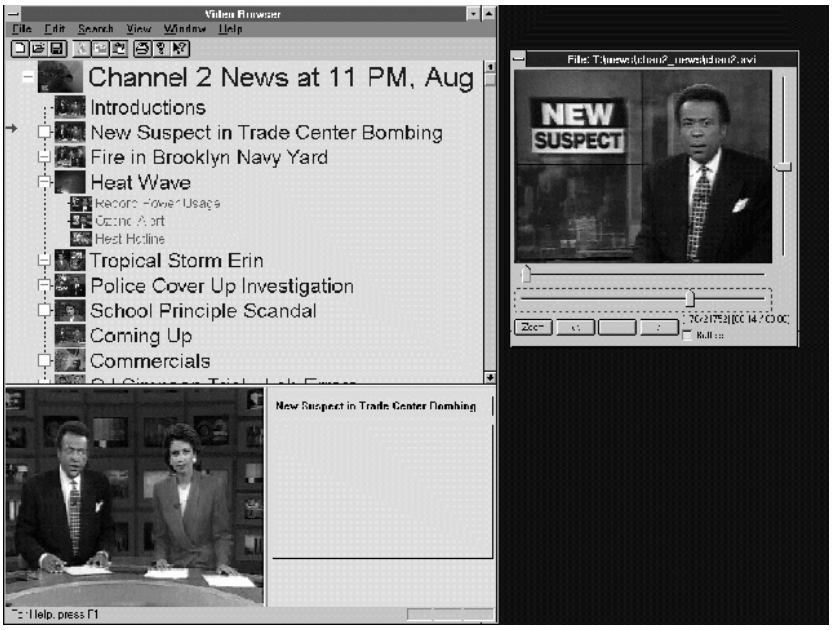

Figure 3: *A screen dump of one of the video browsing tools. The live video window is on the right, the frames on the left show the 'table-of-contents' overview of the clip, a representative still frame, and the current frame text tag.*

2.5 *The Video Browser*

The *Video Browser* is a tool for viewing and searching corpora of digital video indexed using the *Video Tagger*. It supports searching and playback from a local disk or CD-ROM, a networked file system, or a multimedia server using the access schemes described earlier. The *Video Browser* was designed to a be a flexible tool for viewing and searching video. We identified three main types of access to the video library that a user might require: *viewing*, *browsing*, and *searching*. In the following three sections we describe in more detail the facilities offered by the browser to support these operations.

2.5.1 Viewing

The browser must provide high quality playback of digital video and easily support VCR operations. One simple feature that we have found particularly useful is a pair of scroll bars for random access to the movie. One supports scrolling through the whole video, while the other allows relatively precise access to 1 minute of video around the current viewing position.

2.5.2 Browsing

There are two kinds of browsing - browsing the whole video library, and browsing the current video. Browsing the whole video library is supported by arranging the available titles into tree based on content (see Fig. 6). The user can expand or contract the tree and preview the listed videos. Because a single hierarchy cannot necessarily convey all of the organizational hierarchies that a user might desire, multiple tree-views of the same content are supported (e.g., based on author or content).

Browsing a single video is in some respects a more complicated problem. The goal of the Browser is to provide mechanism so that a user can quickly a) identify the content of a video to see if it is relevant or interesting, and b) find and view a relevant section of a video. We have chosen to support these activities by providing a number of static or dynamic views of the underlying video sequence. All of these views are tied to the video playback itself. Thus, clicking on a relevant thumbnail image or piece of text will cause the video playback to seek to the appropriate point in the stream. Alternatively, if the video is playing, the relevant index entry or still-frame is highlighted.

Figure 4*: An overview of a video sequence based on thumbnail images. The thumbnails can be selected to be by time step (shown here), particular tag type (viewgraphs, for instance), or on scene changes that have been identified by the preprocessor.*

The views we have incorporated so far are:

1) *Image-Tree View*. This is the interface presented in Figure 3. This is a representation of the expandable 'table-of-contents' hierarchy generated by the Tagger. Each index entry is enhanced by a still frame thumbnail of a frame close to the tagged frame.

2) *Text-Tree View*. As above, but omitting the thumbnail images (faster, and consumes less screen area).

3) *Thumbnail View*. A simple square array of thumbnail images (Fig. 4). The images displayed can be chosen to be those tagged with index entries, those identified as frame breaks by the preprocessor, or just time steps through the video with an adjustable period (every 10 seconds, every minute, etc.). This view of the underlying video is effective if the index is incomplete or not sufficiently specific. For example, setting a thumbnail view every 10 seconds and scrolling through the images is a fast way of finding a particular sequence if it is easily visually identifiable.

4) *Multi-Video View*. This presents many video panes, each playing real-time video, but starting at a different point in the video. The intent of this view is to provide a fast way of browsing through an extended video. For example, with 6 video views, and a one-hour movie, the whole movie can be viewed in 10 minutes. This may not be very useful for a training film, but it is helpful for getting the 'feel' of a video clip.

2.5.3 Searching

Two types of searching are supported, text and image based. Both search tools can be applied to the current video, or to the whole video library or a subset of the whole library. The text-based search tool uses a query language that supports Boolean operators and searches can be narrowed by tag type (e.g., ((Europe \. %speaker%) & ATM) will search for sections or videos in which the author is speaking about ATM, as opposed to videos in which ATM is deployed in Europe.

The second type of searching is a simple but extensible form of visual querying. Conceptually, the query mechanism is similar to that described in Section 2.4.2. A Euclidean distance measure based on the underlying raw visual index information. At present the search options offered are rather primitive, however, the system is extensible and we hope to improve it with published techniques. Search hits, in this case in order of merit, are presented as a sequence of thumbnails. The user can select and browse through the hits until a relevant sequence is found (Fig. 5).

Figure 5: *A screen dump of the output of a visual search (in this case on color content).*

3. The Library

We have been using the I/Browse tools at for several months to construct an extensive database of digital video. An HP 200T optical jukebox as the primary video repository. This device has a capacity of 144, 1.3 Gbyte optical disks. We currently have about 50 Gbyte of video on-line. The HP jukebox has 4 optical disk readers and dual 10 Mbit ethernet connections. The read rate per drive in principle could support more than one reader (we use video codecs that require around 2 Mbit/sec). Practically, long optical-drive seek times limit the system to one active read or write connection per drive. For more simultaneous users, a video server source is desirable. However, a video server is really only cost effective if many users are watching the same content (a magnetic disk can support about 20 users but only if they all wish to watch the content stored on the disk). Ideally, for a large installation, the system should automatically migrate content to and from the jukebox based on demand, but again this makes sense only if there are many users viewing a limited subset of the total available content.

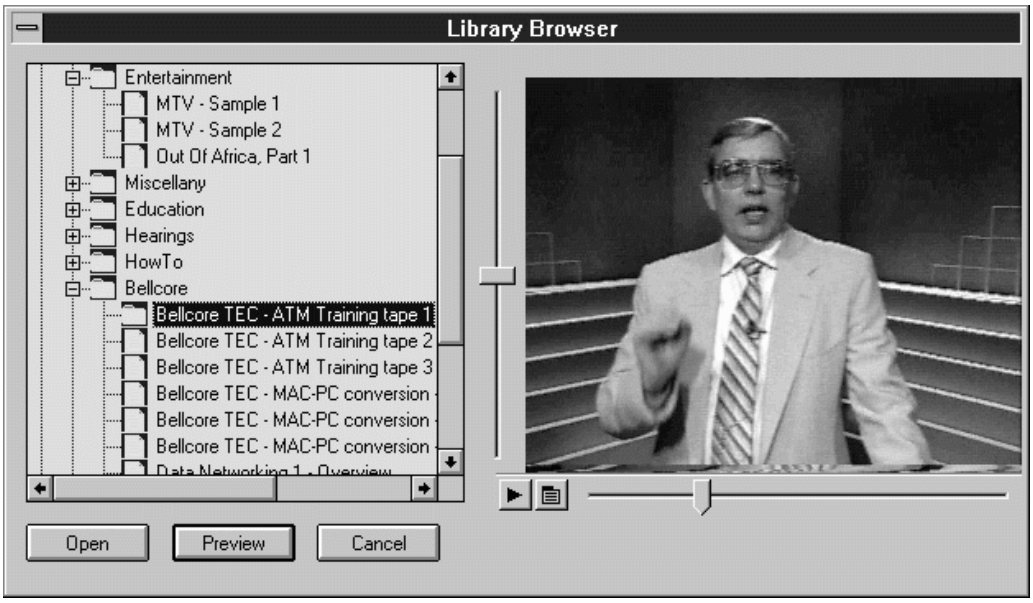

Figure 6: *A library browsing widget. The viewer can get fast visual random access to any video in the collection and many hierarchical views of the same content can be constructed.*

4. Related Work

Many authors have recognized that video information without supplementary indexing information is hard to navigate and use successfully, and so the field of automatic indexing and segmentation has been quite active in recent years.[4] The approaches to indexing that have been explored span the gamut from low-level pixel based approaches, to video segmentation,[5] to sophisticated image content extraction schemes.[6] Other fruitful approaches have included voice recognition techniques applied to a soundtrack and the exploitation of additional related data feeds, like closed captioning or a teletext feed.[7]

At the level of pixel or other low-level frame based parsing, considerable effort has been expended in discovering tractable and robust schemes for automatic scene-change identification.[8] This step is important for almost all subsequent steps, from semantic extraction, to visual query-by-example searches. A set of schemes based on histograms of color content appear popular. They are fast and fairly reliable and we have included such schemes in the preprocessor.

Perhaps the most difficult part of the indexing process is constructing a representation of the content beyond simple tags in the video. Yeung et al. have used inter-shot measures of similarity to construct a directed graph which can effectively represent conversations and some other simple temporal relationships.[9] Zhang, et al. have exploited easily identifiable frames (a news anchor person) to provide some content extraction in the case of a news program.[10] Effective content extraction systems can also use a variety of source for clues; for example a transcript (from a speech-recognition engine) and frame differences can be used to extract subjective semantic boundaries.[11]

5. Conclusion

This paper has described a tagging scheme which is both simple and powerful for indexing digital video. The tagging scheme allows arbitrary text or other tags to be associated with frames in the video. The tags themselves contain type information and can be organized into a hierarchy. In addition, a suite of tools for constructing and browsing libraries of digital video has been described. The approach we have adopted is to provide tools for fast and effective computer *assisted* index generation as opposed to full machine processing. We have used these tools to generate a large database of digital video that is in current use at Bellcore.

There are a number of directions that we hope to take this work. In particular, far more machine intelligence could be adopted in the preprocessing stages. We are still convinced that a human operator is necessary and desirable for producing a useful index of a video, but the more assistance the machine can provide in first-pass segmentation, the faster and easier this process will be.

References

[1] A. Heybey, M. Sullivan, and P. England., Calliope: A Distributed, Scalable, Multimedia Server, *USENIX*, Jan 1996.

[2] R.B. Allen, Interface Issues for Interactive Multimedia Documents. In: *Forum on Research and Technology Advances in Digital Libraries*, Edited by: N. Adam et al., 1995, Springer-Verlag,

[3] J.R. Smith and S.-F. Chang, Quad-Tree Segmentation for Texture-Based Image Query, *Proceedings ACM Multimedia,* 1994, pp 279-286.

[4] F. Arman, R. Depommier, A. Hsu, and M-Y. Chiu, Content-Based Browsing of Video Sequences. *Proceedings, ACM Multimedia*, 1994, pp 97-103.

[5] P. Aigrain and P. Joly, The Automatic Real-Time analysis of Film Editing and Transition Effects and its Applications, *Comput. and Graphics,* Vol 18, No. 1, 1994, pp 93-103.

[6] A. Nagasaka and Y. Tanaka, Automatic Video Indexing and Full-Video Search for Object Appearances, *Visual Database Systems, II*, Elsevier Publishers B.V., 1992, p13.

[7] M.G. Browse, J.T. Foote, G.J.F. Jones, K. Sparck Jones, and S.J. Young, Automatic Content-Based Retrieval of Broadcast News, *Proceedings ACM Multimedia*, 1995, p35.

[8] A. Dailianas, R.B. Allen, and P. England, Comparisons of Automatic Video Segmentation Algorithms. *Proceedings, SPIE Photonics East'95:* Integration Issues in Large Commercial Media Delivery Systems. Oct. 1995, Philadelphia, and references therein.

[9] M.M. Yeung, B.-L. Yeo, W. Wolf, and B. Liu, Video Browsing Using Clustering and Scene Transitionson Compressed Sequences, *IS&T SPIE, Multimedia Computing and Networking*, 1995.

[10] H. Zhang, S.Y. Tan, S. W. Smoliar, and G. Yihong, Automatic Parsing and Indexing of News Video. *Multimedia Systems* (1995) 2:256-266

[11] M. A. Smith and M. G. Cristel, Automating the Creation of a Video Library, *Proceedings ACM Multimedia*, 1995, p357.