VisGenie: a Generic Video Visualization System

Yong Wang
Dept. of Electrical Engineering
Columbia Univ.
ywang@ee.columbia.edu

Lexing Xie
Dept. of Electrical Engineering
Columbia Univ.
xlx@ee.columbia.edu

ABSTRACT

In this project, we designed and realized *VisGenie*, a generic system for visualizing multimedia streams and the associated metadata stream. Information visualization is not only useful for end users, but also beneficial for researchers who work on analyzing the media data and improving our understanding and presentation of multimedia information. The basic building blocks of the system include: the media renderer, the curve drawer, the frame display, the scalable event display, the system initializer, and the external data interface. And the integration of these blocks provides the flexibility and extensibility to different kinds of applications. Prospective applications range from signal-level processing to semantic analysis of multimedia data, and in this report, we present three examples: video coding, shot boundary detection, and soccer video analysis.

I. INTRODUCTION

In this project, we introduce *VisGenie*, a generic multimedia visualization system. The role of our system is to visualize the audio-visual stream, along with its associated metadata stream. The metadata not only include high-level concepts as most existing visualization systems do, it also include low-level features that are of particular interest to researchers who work on analyzing the content and developing automatic content filtering and concept extraction techniques. A better visualization interface will not only facilitate user interaction, e.g. random access, search and browsing; it will also help researchers reach a better understanding of their problem, and in turn develop even better tools for multimedia analysis and representation.

Visualization and user interface has been an active research topic for decades, tracing back to the design of early UI design of computer operating systems such as MS-DOS, and the first ACM SIGCHI conference in year 1982. The principle of visualization, epitomized by Shneiderman [7] as “overview first, zoom and filter, then details on demand”, can be interpreted in three different aspects of a good visualization: multi-scale structure, intuitive presentation, and ease of interaction. In visualizing attributes [1] and concepts [5], the expressiveness of textual data is greatly enhanced by mapping them to a 2-D plane, and giving meanings to the geometric properties such as shape, size and distance. In visualizing multimedia data, the InfoMedia project [3] had an integrated interface for image storyboard, closed caption and random access. And in visualizing other modalities, an increase in dimensionality often brings much richer information than the original signal, such as: visualizing music timber in 3-D space [4], or highlighting protein coding regions in a DNA sequence [2].

The primary scope of our *VisGenie* system is to visualize data streams with temporal evolvement (esp. video and its metadata) on a planar display. And the system has the following characteristics: (1) Generality: it is independent from particular metadata extraction modules, it can compute, parse and display various types of user-defined metadata; (2) Flexibility: the users are placed within the design loop, and they can decide how the final visualization should look like. At the bottom layer of the *VisGenie* system is a dozen independent functional components realizing various basic tasks such as playing the media, drawing continuous curves, displaying image storyboard, etc. The middle layer is the central control of the system; it is responsible for synchronization between components, routing data streams to different components, etc. Preference of the user is at the top of the whole architecture, which and how many components to use, the look and feel of individual components, and the association between each component and different metadata stream are all maintained in this layer.

Section 2 is an overview of the system; in section 3 we briefly introduce each of the system components; in section 4 we present how these components are organized and controlled; in section 5 three sample applications and their generalization are given; and section 6 concludes this report.

II. SYSTEM OVERVIEW

The foundation of the VisGenie system is a blank outer window; it is initialized as soon as the application is launched. And after that, individual functional components are launched according to the specific application and the user preference. These components take the form of floating sub-windows, and the users are allowed to drag-drop, and customize their appearances, independently. We decide to use this freeform component design rather than the popular fixed-panel layout such as [3] because we are addressing a wide variety of potential applications rather than a specific application. And we are aware that it is neither possible to have a universal optimal layout for all applications, nor is it feasible to construct a separate layout for every single application, so this task is left to the user. Take Figure 3, 4, and 5, for example, the appearances of *VisGenie* are vastly different for different applications, hence it is more preferable to optimize the layout for each new application.

Each of the functional sub-windows is realized as a COM component. They are packaged independently, and the entrance of each includes several necessary configuration inputs and the data stream. The synchronization among these components is maintained by a central control that will be discussed in section 4, and the initialization and data routing are the responsibility of a data interface. We will present each component, the system configuration and data interface, and then system integration issues in the sections that follow.

III. SYSTEM COMPONENTS

3.1 The media renderer

Central to the whole system is the media renderer, based on *Microsoft DirectShow* library. A typical media player interface by its look, it also takes the additional responsibility of maintaining the system clock and providing online processing interface.

The primary responsibility of this component is the playback of media streams. The generality of the library enable us to handle a wide variety of media format of different modalities: from MPEG-2, MPEG-4, AVI, MP3, to still images (i.e. nearly every media type that *Microsoft Media Player* supports). *DirectShow* has a *filter-module* structure where each independent *filter* performs a specific task in the whole rendering assembly. Figure 1 is an illustration on how this works for an mpeg stream. For example, the *MPEG Video Decoder* takes the de-multiplexed video stream as input, decodes it into a certain pixel format and forwards the result to the *Video Renderer.*

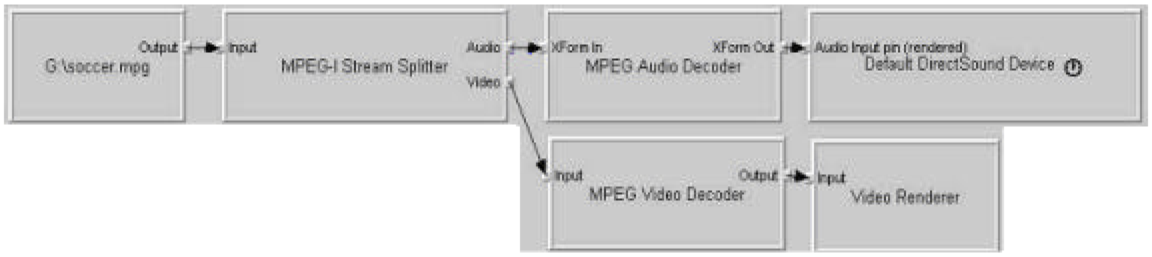


Figure 1. Media rendering flowchart for MPEG audio-visual stream

In addition to using the existing media filters, we also designed a new *DirectShow* filter, the *Video Hook*, to take care of two important tasks: providing online processing interface, and the system synchronization signal.

Figure 2 shows a pixel-level processing hook. The hook intercepts the decoded image data before they are passed to the display, and then these data are sent to a user-defined online processing module. After the image data are processed, the application can send the extracted features to other visualization components, and at the same time send the image data back if the processed image is meant to appear in the display. Shot boundary detection as discussed in section 5.2 (Figure 4) is an example where this scenario is exactly applicable. Other applications of online processing include real-time transcoding, content filtering, object segmentation and tracking, etc.

Similarly, this idea of a processing hook can be easily adapted to the processing of other modalities such as the audio or the closed caption; and compressed domain processing can also be realized by simply moving the hook to an earlier stage where compressed bit-stream is available.

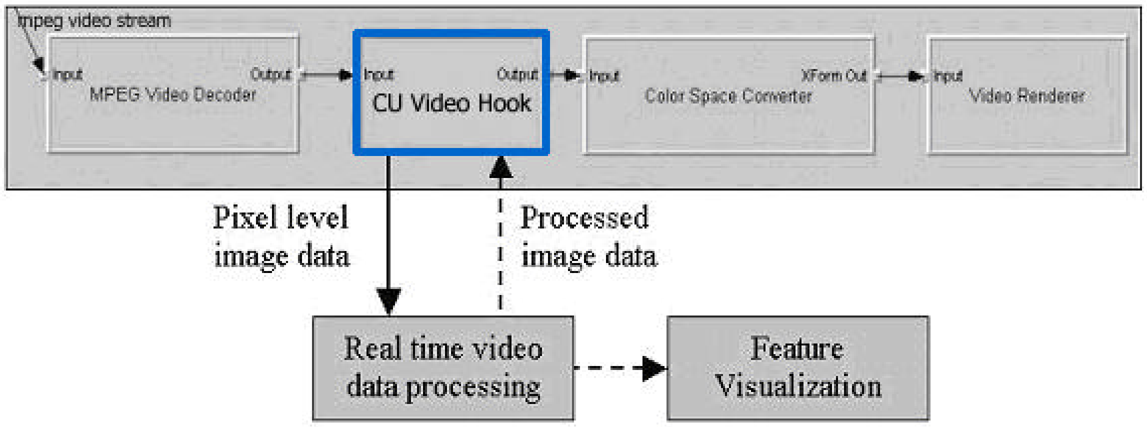


Figure 2. Video Hook

Another task of the *Video Hook* is to send out a global sync signal upon rendering each video frame, and this is used to control the feature display, as further illustrated in section 4.2.

3.2 Generic curve display

Curve display is very useful, as many video features can be abstracted as different types of curves, such as the histogram, PSNR, bit rate, feature distances, etc. Our current version of generic curve window supports various curve formats such as column, line and shape; we provide interface for the users to customize the color, line width and the style of their curves; some convenient controls such as zooming and browsing via the curve are also provided; in addition, this curve component is also the basis of more specialized and sophisticated components such as the event window (section 3.4).

3.3 Image storyboard

The image storyboard display is widely used in many state-of-art visual information systems. The key-frame collection layout has been successful because it transforms the sequential nature of the video stream into a 2D parallel display, thus enables fast overview and navigation, and the loss of audio and motion dynamism is compensated by letting the users quickly access the original video segment from key-frames.

Our frame storyboard can, but is not limited to displaying key frames from the video: the users are allowed to put any meaningful images into this window, such as segmentation maps for each shot, edge maps for each image cluster, etc; the users can also associate a set of textual or numerical metadata with each frame, including inherent frame properties such as the time stamp, size and color, and custom properties such as decision confidence, user comments, etc.

3.4 *Event window*

There are scenarios where low-level features or frame maps are not enough to tell the whole story, and here higher-level abstraction comes into play. Typically, these applications involve machine decision, or semantic abstraction of the content, such as the *indoor/outdoor* concept, output of a *face* detector, or more domain-specific concepts such as a *dialogue* in film, or a *scoring* event in a sports game.

The layout of our event window (Figure 5) is inspired by MovieDNA[6], and the Shneiderman three steps [7] are better embodied in this window than in other components: the iconic compact representation in the lower panel provide an "*overview*" of the whole video clip; the adjustable upper panel let the user "*zoom and filter*" at their will; the user can easily request any "*details*" of the original content from their point of interest in any level of the zoom representations.

3.5 System configuration

VisGenie allows the user to control the look and feel of their visualization via a global configuration file. The kinds of control that the system currently supports can be divided into 4 categories: (1) Global info: which kind of component to use, how many of them; (2) Window properties: the size and position, back ground color, grid size and color, window title… (3) Data association: which media file or data file is associated with this window, data format options, data-specific parameters such as the time-normalizing factor… (4) Window content properties: the color and style of curves, granularity of the event overview window, annotation of key frames, …

Specifically, we adopt the windows *.ini* file format, which is both machine and human readable. We put properties of each window in a separate section, and each key value represents a specific property. Hence it is easy to define new properties or make changes to existing properties.

3.6 Offline data interface

This module is used to parse different data files and supply different kinds of visualization components with offline feature values, event flags, or key-frame info. It can handle sequential data access as well as non-sequential access by searching closest timestamp upon request from the central control.

IV. SYSTEM INTEGRATION

4.1 Initialization

Upon loading a configuration script, the system initializes all components as requested, and pre-fetches data if specified by data option.

4.2 Synchronization

The *Video Hook* filter (section 3.1) starts sending out timestamps after the video starts playing. The control module takes this timestamp, sends it to all other active components; they in turn fetch the data from file or memory, and update the display.

4.3 Extensibility and reusability

We are trying to ensure that *VisGenie* is an easily extensible and reusable system throughout the design. Extensions such as new visualization components, new curve types, new data formats or new data properties are straight-forward; further extensions may include new UI, new modality, as discussed in section VI.

The system is reusable in three different levels: the highest, application reuse, which is most useful for end users and researchers, is our primary target in designing this system, examples can be found in section V; COM reuse stands for the mid-level, where researchers and developers can easily include their own online processing modules or new components with minimum programming; source code reuse is at the lowest level, and it provides the maximum level of flexibility and customization for system developers.

V. SAMPLE APPLICATIONS

In this section, we will present three specific examples where the *VisGenie* system is applied, and these examples are taken in increasing order of content abstraction.

5.1 Video coding

The interface is shown in figure 4. Here is pure signal-level processing, without trying to understand the content.

Specifically, we compare 2 different rate-control schemes in MPEG-4 verification model. Both streams are encoded at a same target bit-rate of 112KBps; one algorithm allocates more bits to I-frames, and the other algorithm does not have such bias among different frame type.

In *VisGenie* interface, two synchronized encoded video streams, along with their PSNR curves are shown. And the quality difference indicated by the PSNR or the perceptual image quality can be easily verified with regard to each other.

This kind of application can generalize to other signal processing scenarios such as: VBR traffic monitoring, visualizing temporal, spatial, or other factors in trans-coding utility functions, showing network condition, etc.

5.2 Shot boundary detection

The interface is shown in figure 5. Here low-level content processing is employed, and this is done without semantic understanding.

In this setup, the shot boundary of a video is detected using color histogram distance, and all the computation is done in real-time.

As the video is playing, the 64-bin HSV histogram for every frame is computed and displayed; the L2 and cosine distance of the current histogram with regard to the histogram of the previous frame is computed and displayed; a shot boundary is declared present if this distance exceed a preset threshold, and the first frame of the new shot (along with its textual metadata) is dumped to the image storyboard panel. The user can also navigate through the video clip and verify that the shot boundaries are indeed correct by clicking the image thumbnails, or using the random access function of the media player.

Examples for generalization include: edge detection, motion estimation, object tracking, content-based video retrieval, and so on.

5.3 Soccer video analysis

The interface is shown in figure 6. Here semantic analysis of the content is involved.

Specifically, the basic semantic units of a soccer game, i.e. play and break, are automatically classified using statistical learning techniques [8].

In VisGenie interface, we can see the video, the two continuous feature curves used in computation, and the semantic states as marked by ground-truth, the intermediate stage of the algorithm, and the algorithm output. Thus it is more intuitive which kind of feature values lead to what decision, and we will have a better idea about why the algorithm success and when it will fail.

This application can also be generalized to real-time content filtering, scene-level video structure analysis, video summarization, …and various others.

VI. CONLUSION AND DISCUSSION

In summary, we have designed and realized a generic video visualization system, which amounts to more than 15,000 lines in C++, and about a dozen COM components.

A prototype as it is, there is much room for improvement:

(1) There has always been a trade-off between the generality of system architecture and the performance it can achieve on specific tasks. To improve the efficiency of the system, we need to consider the amount of assumptions we have on prospective data more carefully. Such as whether the data are stored sequentially, whether the request for data will come sequentially or at random, whether the data are uniformly sampled, whether the data request comes in regular intervals or in a sporadic manner, …

(2) The interface could be better if some cosmetic improvements are employed. For example, having graphical representation of objects will be more intuitive that just drawing color blocks.

(3) As users are more comfortable with graphical dialogue boxes rather than textual initialization scripts, it will be useful if we setup an interactive subsystem for building new demos and customizing existing demos.

(4) In addition, there is much to do for additional visualization schemes in other categories and media data in other modalities. Such as the ViBE [5] interface for visualizing concepts; or visualizing the audio channel in an mpeg stream, though we are aware that audio visualization is an active research problem in itself.

REFERENCES

[1] Ahlberg, C. and Shneiderman, B. "Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays", *Proc. ACM CHI 1994 Conference on Human Factors in Computing Systems*, Boston, pp313-322.

[2] Anastassiou, D., "Genomic Signal Processing", *IEEE Signal Processing Magazine*, v18 n4, July 2001 Page(s): 8 -20

[3] Christel, M., Martin, D., "Information Visualization Within a Digital Video Library", *Journal of Intelligent Information Systems* 11(3); pp. 235-257, 1998

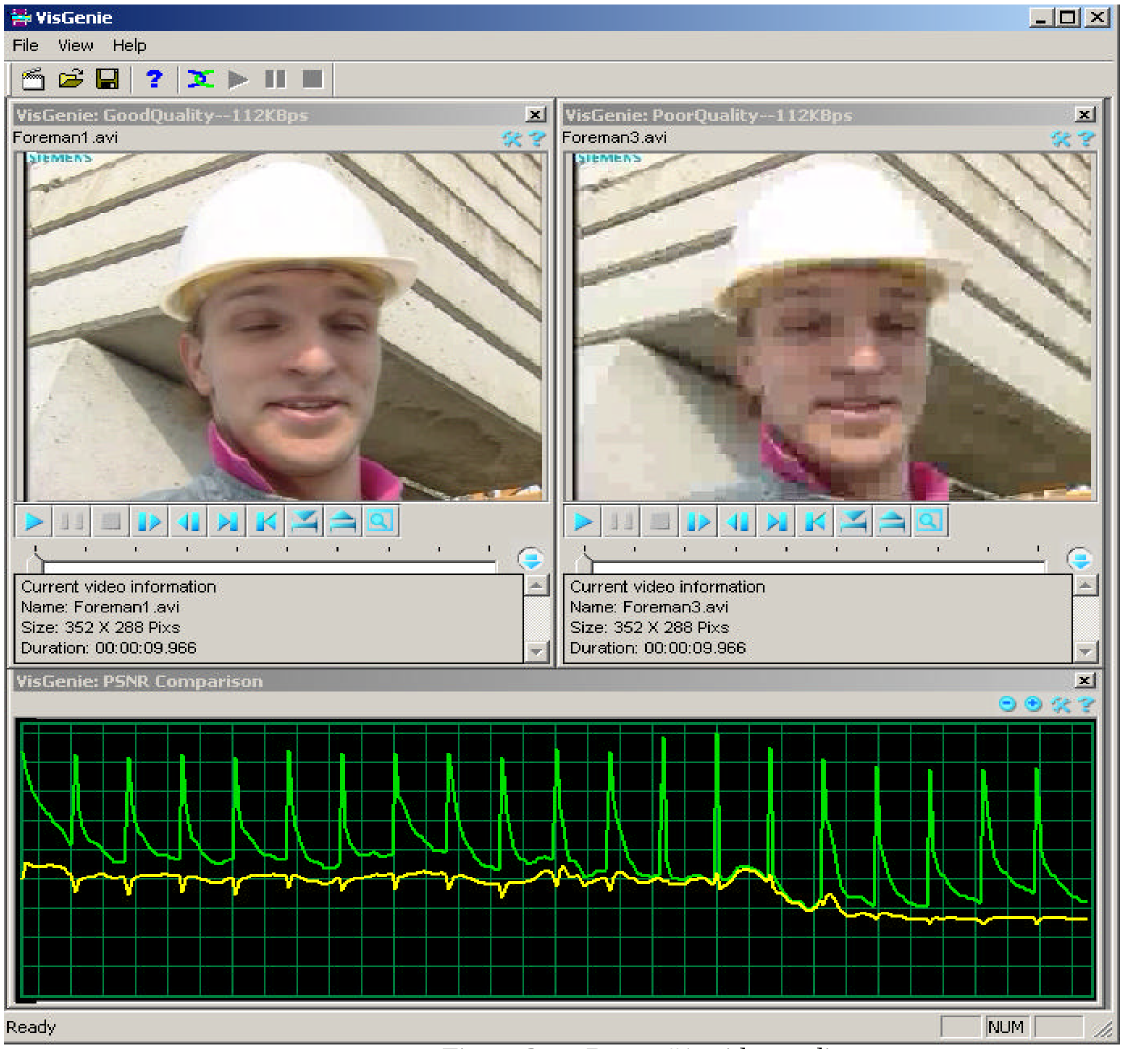
[4] Freed, A. "Improving Graphical User Interfaces For Computer Music Applications," *Computer Music Journal*, vol. 19, 4-5, 1995.

[5] Olsen, K. A. et al. (1993). "Visualization of a document collection: The VIBE system." *Information Processing and Management*, 29(1): 69--81

[6] Ponceleon, D., Dieberger, A., "Hierarchical brushing in a collection of video data", *Proc. 34th Hawaii International Conference on System Sciences* 2001, pp 1654 -1661

[7] Shneiderman, B., "The eyes have it: A task by data type taxonomy for information visualizations", *Proceedings IEEE Visual Languages*, pp 336-343, Boulder, CO, Sept 1996

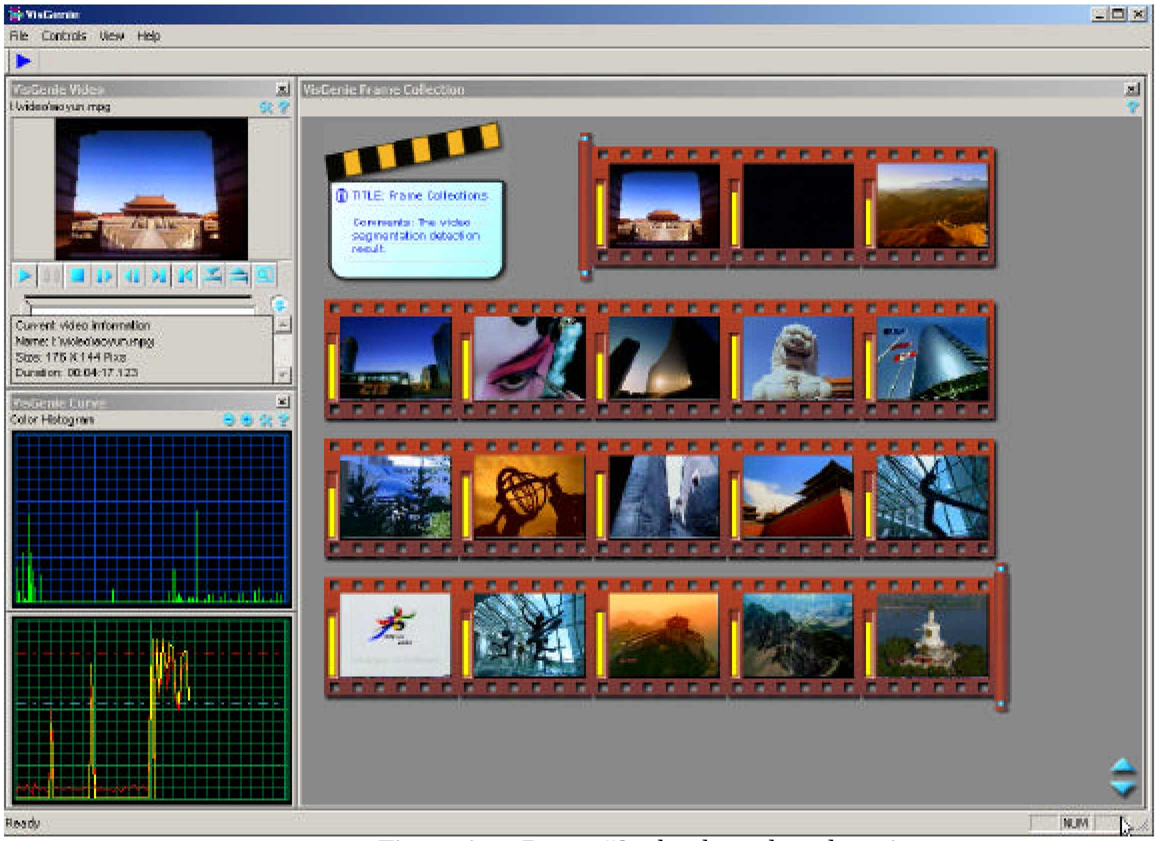
[8] Xie, L., Chang, S.-F., Divakaran, A., Sun, H., "Structure analysis of soccer video with hidden Markov models", *Proc. ICASSP 2002*, to appear


VisGenie
File View Help
VisGenie: GoodQuality--112KBps
Foreman1.avi
VisGenie: PoorQuality--112KBps
Foreman3.avi
VisGenie: PSNR Comparison
Ready
NUM

Current video information
Name: Foreman1.avi
Size: 352 X 288 Pixs
Duration: 00:00:09.966

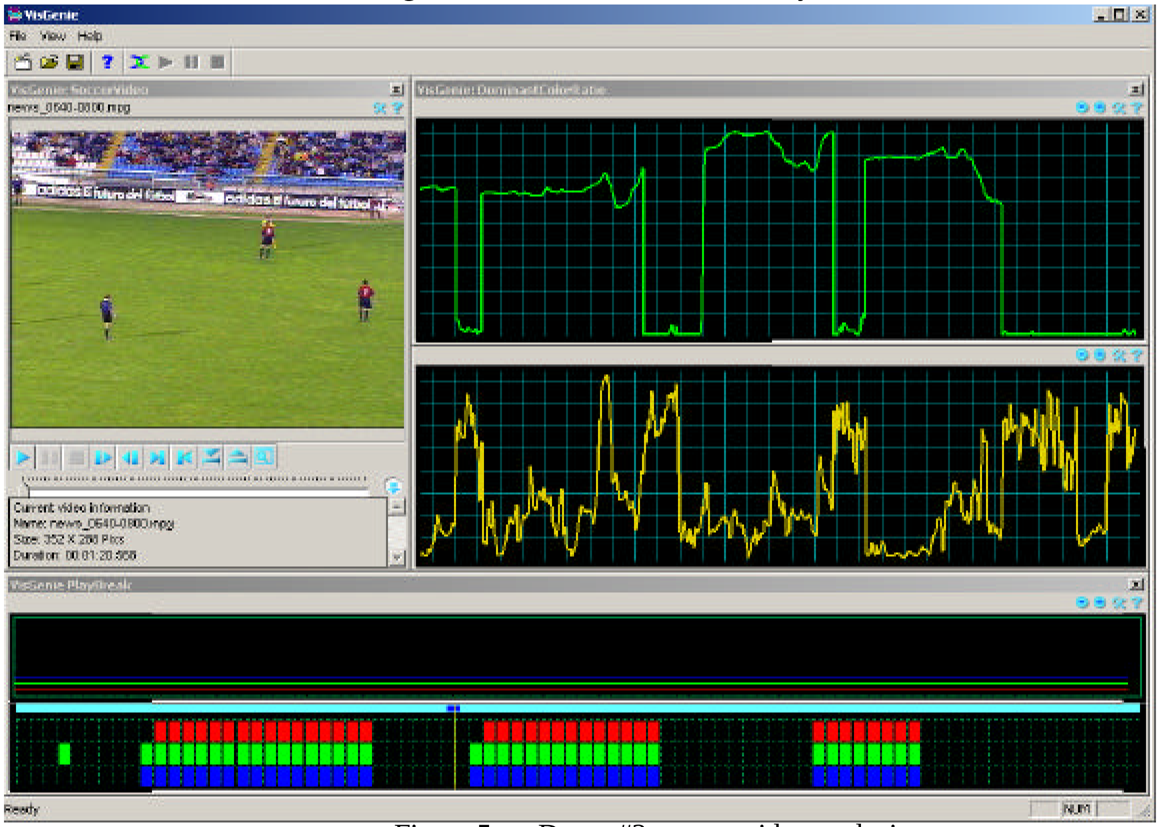
Current video information
Name: Foreman3.avi
Size: 352 X 288 Pixs
Duration: 00:00:09.966

Figure 3. Demo #1: video coding



Current video information
Name: I:\video\aoyun.mpg
Size: 176 X 144 Pixs
Duration: 00:04:17.123

Figure 4. Demo #2: shot boundary detection



Current video information
Name: news_0640-0800.mpg
Size: 352 X 288 Pixs
Duration: 00:01:20.958

Figure 5. Demo #3: soccer video analysis