WORTHY VISUAL CONTENT ON MOBILE THROUGH INTERACTIVE VIDEO STREAMING

Ivan Alen Fernandez [1], *Christophe De Vleeschouwer* [1], *Fabien Lavigne* [2] *and Xavier Desurmont* [3]

[1] TELE, Université Catholique de Louvain-la-Neuve
Emails: {ivan.alen,christophe.devleeschouwer}@uclouvain.be
[2] Département TCTS, Université de Mons [3] Image department, Multitel, Mons
Emails: Fabien.Lavigne@umons.ac.be, desurmont@multitel.be

ABSTRACT

This paper builds on an interactive streaming architecture that supports both user feedback interpretation, and temporal juxtaposition of multiple video bitstreams in a single streaming session. As an original contribution, we explain how these functionalities can be exploited to offer improved viewing experience, when accessing high-resolution or multi-views video content through individual and potentially bandwidth-constrained connections. This is done by giving the client the opportunity to select interactively a preferred version among the multiple streams that are offered to render the scene. An instance of this architecture has been implemented extending the liveMedia streaming library and using the x264 video encoder. Automatic methods have been designed and implemented to generate the multiple versions of the streamed content. In a surveillance scenario, the versions are constructed by sub-sampling the original high resolution image, or by cropping the image sequence to focus on regions of interest, in a temporally consistent way. In a soccer game context, zoomed-in versions of far view shots are computed and offered as alternatives to the sub-sampled sequence. We demonstrate the feasibility and relevance of the approach through subjective experiments.

Keywords— Mobile Stream, Interactive System, Automatic Enriched Content, Video Segmentation, H.264/AVC, Region of Interest

1. INTRODUCTION

Mobile streaming service through cell phone is becoming the highlight of new value-added mobile services. Based on the present CDMA1x wireless data network, and the adoption of compression technologies such as H.264 [1][2][3], several media players are currently being designed and implemented for mobile handsets. At the same time, video analysis and processing have largely been investigated to identify regions or periods of interest, especially in sport events broadcasting context, as reviewed by [4]. Several proposals to create summaries of sport events have also been developed, as presented in [5], which acquire high relevance for *on demand* video streaming content.

In this context, with the massive diversification of mobile users, and due to the limitations of bandwidth in the mobile network, the concept of client profile is earning more and more importance. Its default purpose is to offer different quality and type of content, according to the services that each client has purchased. For example, if the clients mention their hobbies or interests through their profile, the application server can search its content database for related video sequences targeting particular users for who the content is delivered. In addition, the user profile can be used as well to enable targeted advertising through interactive applications.

In this paper, we focus on mobile terminals, and propose a streaming architecture that offers personalized and interactive access to high resolution video content for any client using an H.264/AVC compliant player. Our work results from the two following observations. At first, due to mobile networks bandwidth limitation, it is often not possible to transmit high-resolution video content. On the other hand, content produced for conventional wired broadcast or captured by surveillance networks is gaining in resolution. As a consequence, this content has to be downsampled to be accessed through mobile networks, thereby losing a lot of its value. A possible solution might be to manually produce a second version of the content that is dedicated to low resolution accesses. However, this solution is expensive and not appropriate in many application scenarios (e.g. surveillance or real-time post-production of broadcasted content). For those cases, the only alternative is to design automatic video processing systems that produce low resolution content from the native high-resolution content. This low resolution content typically focuses on Regions-of-interest (RoI) which are application dependent consisting in a cropped version of the original content. Such kind of automated tools have already been investigated in the literature, and a general conclusion is that none of the existing method is 100% reliable in the way it defines RoIs. Therefore, human supervision of the process is always required to check that the automatic system always behaves properly. Besides, in some cases, several regions are likely to be of interest for the user.

Our framework proposes to circumvent those issues by leaving the end-user decide about the version of the content (s)he would like to visualize. Hence, the automatic system produces one or several versions of the content, and the user gets the opportunity to switch interactively between the versions. Typically, the (automatically) produced low-resolution versions include a sub-sampled version of the native video, plus one or several zoomed-in (and cropped) versions, each one focusing on one (of the) region(s)-of-interest detected in the native high-resolution video. In some application scenarios, replays of hot spots actions are also proposed to the user. In practice, client-transparent switching between versions is enabled by splitting each produced video in short clips that are temporally pipelined by the server, based on user's requests. From the client's perspective, everything happens as if a single pre-encoded video was streamed by the server. In the following, the set of clips resulting from this versioning process and the associated organizational information are referred to as enriched or enhanced content.

In the context of real-life and large scale deployment of the system, one could imagine to monitor how the end-users actually visualize the content, so as to cancel the non-accessed versions from the available list of clips. This mechanism has not been considered in our experiments, but would be easy to implement on top of our architecture.

The remaining of the paper is organized as follows. Section 2 presents an overview of the proposed system architecture. Section 3 explains its underlying concepts and implemented components, namely the client-transparent temporal content pipelining and the user feedback interpretation. In Section 4, we present several techniques that

Thanks to Walloon Region project WalCoMo and Belgian NSF for funding part of this work.

we have developed to generate automatically the multiple versions of the video content that are required to offer enhanced interactive viewing experience. Finally, Section 5 presents some qualitative results, while Section 6 concludes.

2. ARCHITECTURE OF THE STREAMING SERVER

The main objective of the architecture that we propose and develop is to offer interactivity to any client of a mobile video streaming session when using an H.264/AVC compliant player. A preliminary version of this architecture was already proposed by the authors in [6]. All the modules of this architecture are now fully implemented. The communication is established with the client through the Real Time Streaming Protocol (RTSP). As the interactivity mainly relies over H.264 features, it can be extended to any other kind of clients who just support decoding a standard Baseline Profile of H.264/AVC. In addition, our architecture considers the automatic generation of enriched content which implies resizing the video data for a mobile phone screen (just by focusing the view over the RoI). In addition, our scheme supports browsing (e.g. fast forward or reward actions) through pipelining. All together provides a worthy visual content on mobile, achieved through interactive video streaming.

The global architecture described in Figure 1 is based on three main elements: the enhanced content creation unit, the streaming server and the session control module.

2.1. Enhanced Content Creation Unit

This unit fills the Video Content Database, without actively taking part afterwards in the streaming system. Its purpose is threefold:

- It analyzes the TV like video content to identify regions-of-interest and produce several zoomed-in or replay versions of the content, as described in Section 4.
- It divides the video sequences in small segments which are encoded in H.264, as explained in 3.2.
- It generates the metadata introduced in section 3.3 to define the interactive playing options associated to the segments. The metadata information is used by the session control to monitor the streaming session.

Therefore this module provides all the input required for the interactive system, which is delivered afterwards through the streaming server, which dispatches the Real-time Transport Protocol (RTP) packets to the client terminal.

2.2. Streaming Server

The streaming server module is the core of the system, where the interactive streaming mechanisms are implemented, based on client-transparent content pipelining, as described in section 3. Client-transparent temporal content pipelining allows the server to stream multiple successive stream segments (or clips) in a single session, without negotiating with the client the establishment of a new streaming session. Hence, with this feature the server is able to change the streamed content while maintaining a unique output stream and keeping the existing session uninterrupted. As a result, both a temporal and computational gain are achieved as the client does not need to establish more than one streaming session.

2.3. Session Control

The control module determines, every moment, which video segment has to be transmitted next by the server, based on the user's feedback and the metadata associated to the last data segment that has been dispatched. This unit consequently decides which video sequences are juxtaposed using temporal content pipelining, acting as an interface between the streaming server and the metadata.

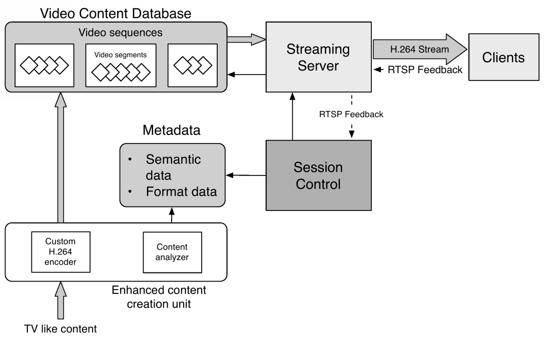


Fig. 1. Diagram of the architecture's workflow

The session control is an essential part of the system, as it controls almost any information flowing through the system. The processing of the metadata information (based on the status of the stream session), is described in section 3.3. Afterwards, it provides the result to the server which is responsible of accessing the video content database.

3. TEMPORAL CONTENT PIPELINING

Temporal content pipelining is the technique that allows a streaming server to juxtapose multiple streams in a single continuous sequence, so that multiple streams can be forwarded to the client through an unique streaming session. The key for implementing this functionality is the session control module using the advanced features of the H.264 codec[7], regarding the encoding parameters transmission.

3.1. H.264 Advanced Features for Streaming Interactivity

When a RTSP transmission is settled, the sink must deliver RTP packets which contain unformatted binary data. However, a sequence encoded with a H.264 codec is composed by series of Network Abstraction Layer Units (*NALUs*). Therefore, a parser has to be inserted before the source. This parser reads the H.264 data coming from the source and breaks it into H.264 *NAL* units before providing them to the sink. The *NAL* units may contain data from the sequence (*VCL NAL* units) or additional data such as parameter sets (non-*VCL NAL* unit) with *VLC* standing for Video Coding Layer. These units contain information which is not likely to change for several frames.

The H.264 standard defines two kinds of parameter sets : Sequence Parameter Sets (*SPS*) and Picture Parameter Sets (*PPS*). The first applies to a wide range of images, while the latter only to particular pictures. Every *VCL NAL* unit contains in its header a parameter linking it to a *PPS*, which in turn links to a specific *SPS*. During the decoding process, these parameters are stored and used every time a *VCL NAL* unit is associated to them.

In our proposed framework, the *SPS* updates are sent between two pipelined segments in order to enable any potential change of encoding parameters. In fact, every segment in a transmission of two or more pipelined sequences is encoded independently to any other one. To avoid transmitting parameter set updates synchronously to the slice packet stream, the encoder and decoder can maintain a list of more than one picture parameter ser. As opposite to [6], a different *SPS* id is applied/transmitted between consecutive segments with different encoding parameters. This means that the new parameters are added rather than updated. According to [8], overwriting parameters might result in decoding errors at the receiver, particularly in a multiparty session. Furthermore, as new encoding parameters in the sequence might be stored in the decoder previously in the streaming session, a profit for the efficiency of the video coding is achieved [8].

As the first *NAL* unit of an H.264 segment always contains the *SPS* and the *PPS*, multiple sequences can be transmitted consecutively without any interruption and the output is still compliant to the H.264 standard. The new sets of parameters are sent out-of-band through the RTSP protocol and the new related NAL units are not sent prior to acknowledgement of delivery from the signaling protocol. On the client's side, a unique sequence is received. This sequence, however, is built step by step by the server depending on the received feedback and on the metadata associated to every segment.

3.2. Video Content Segmentation

The video content database contains all the available video sequences. The enhanced content creation unit produces video sequences divided in small video segments. These video segments represent the basic video unit that the streaming server is able to access and play. Some segments are redundant as they may describe the same temporal period of the original sequence with distinct sampling or encoding parameters (e.g. to render a zoom of the action, or to display different RoIs in a videosurveillance scenario).

Thus, when a whole sequence is requested, the server streams all of its individual segments in the appropriate order, potentially replacing the segments with one of their redundant versions depending on the user's feedback. When the switching between versions carries a switching between encoding parameters, the *SPS* that are associated to the new kind of parameters are activated in the decoder. The forehand transmission and acknowledgment of the parameters through the RTSP protocol is required to start delivering the new sequence.

In order to not to interrupt the temporal line playback, every segment is temporally adapted to another segment belonging to a redundant sequence, both of them sharing either the startup time stamp or the first frame number.

3.3. Session module control implementation

The session control processes the user's feedback and using the metadata associated to the last segment sent in the streaming session, decides which is the next element to be delivered. The metadata associated to every video segment is stored in one Extensible Markup Language (XML) script, which the session control processes after sending every segment.

The metadata is produced by the enhanced content creation unit when the video sequences are inserted into the database, after creating the enriched video content. We have considered two different cases of semantic metadata describing how the segments are related, as displayed in Figure 2. In case *A*, the related segment is introduced in the sequence as additional content and the sequence playback is resumed without any offset. One example of this relation, is replay actions or the insertion in the stream of target advertising when delivering *on demand* video content. In case *B*, the related segments replace the original sequence's segments which are completely discarded. At the end of the related sequence of segments, the sequence playback is resumed with an offset corresponding to the related segment duration. This is very useful in multicamera scenarios and the Fast Forward/Reward actions, both of them supported by our system. In the scenarios developed in section 4, this one is also the relationship that has been considered.

In order to achieve the temporal consistency, all the related segments as in case *B*, need to have the same startup instant of one of the related sequence segments. There is always another segment which finishes right before, linking to the next ones. Therefore, when the switching is triggered in the server, the current segment being sent, has to be completely delivered before switching. Right after, the related sequence can start from the segment that according to the metadata, is

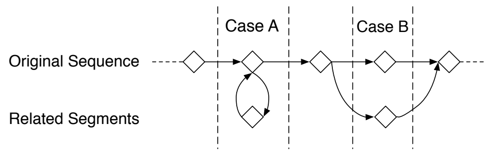


Fig. 2. Metadata considered structures

temporally consistent to the previous one. This is the main reason that all the sequences have to be mandatory divided in multiple small video segments. Otherwise the global system would become inefficient (with slow response).

The session control decision is based on a state machine as a function of the available related segments. If only one sequence is available at some point of the playback, any user request is saved until a related sequence is available through the next frames.

In the scenario of section 4.1, there are two different related sequences: the original version and the zoomed one. When receiving the user's request, the playback is automatically switched between them. If during some segments the zoom sequence is not available after being delivered for several frames, the playback is switched back to the original version. Also when the zoom version is not available at the moment of the user's request, the zoom video segments are triggered as soon as they are available again. In the scenario 4.2, the system response is the same with the only difference that when several RoI are available - therefore more than two sequences available - the playback is switched using *round-trip* between all the possible outputs when the client's requests are received.

In case that the temporal consistency is not required, we have developed a new source that is able to discard any buffered data in the server and begins reading from the new segment file. This way the latency in the server after receiving the user's request is completely avoided. Like in the previous cases, the playback can continue without causing any decoding error and the streaming's behavior is not damaged, performing the switching flawlessly.

3.4. Interactivity and user's feedback

The system's interactivity relies on the RTSP commands which are exchanged between the server and the client, defining a system response to the user's feedback. The client is able to trigger an input stream-switching by sending an appropriate command to the server. A communication channel between the client and the server is already available as both exchange RTSP messages and these messages can be used to obtain feedback from the client.

Any standard client can thus connect to such an interactive system without having to implement special user feedback routines. When the user's request is received, the metadata of the current segment is used by the control module in order to determine which segments are related from other video sequences. When the session control detects the availability of such a segment, it enables the source to switch right at the end of the playback of the current video segment.

Therefore, the client can send a standard RTSP message to communicate their feedback. One of the frequently used RTSP messages is PLAY. Not only does it trigger a stream playback, but it may also seek inside the stream using a parameter called Range [9]. As many clients implement a seek function that send such a message, it is an efficient and simple way to signal user's feedback during the transmission. A simple use of the seek button would then trigger the playback of the related segment to the previously delivered one.

If multiple related sequences are available at the moment of the request, as for example in a multiangle camera scenario, the user can send the request as many times as desired, to switch in between all the available sequences. In a more advanced scheme, the client could desire to switch directly to one of the possible views, without selecting in *round-trip* all the related sequences available. In order to select a custom one, the client must be able to send a custom RTSP message identifying the desired sequence of segments. This would require a custom designed client terminal, designed specifically to send the appropriate RTSP messages to the streaming server.

4. GENERATION OF AUTOMATIC ENHANCED CONTENT THROUGH ROI DETECTION

This section presents RoIs detection methods through two different contexts, sport event broadcasting and video surveillance. Detected RoIs are then used to produce a new resized video focused on interesting parts for the end-user. Such a system offers multiple versions of the same scene. Users can select the original video which offers a general view of the scene or related video sequences by just focusing on specific RoIs.

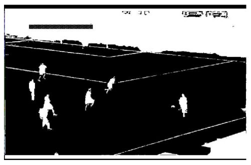
4.1. Automatic zooming for enhancement of sport team video

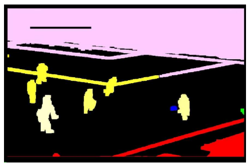
In a soccer scene, RoI is usually unique and well defined. The ball is the central element of the scene. Indeed, players react according to its position. Consequently, detecting the ball in the video provides focusing on the RoI of the scene. This issue will be explained first. Then, we will explain how to produce a new reduce sized video from the RoI which will be always pleasant to see for the user.

Ball detection in soccer scene is an issue discussed in various studies as in [10] wherein the best ball candidate is detected with a modified version of the directional circle Hough transform and validated with a neural network classifier. Hereby, the ball is detected following three main steps.

- **Image segmentation :** First, non-green pixels are selected using color components to separate the soccer field to other entities. Then a morphological operator of closure is applied to delete small artifacts and to merge sparse clusters of pixels belonging to the same object (the players´ legs for example).
- **Object description :** A label map is built from the segmented image to identify each object present in the scene. Then, features of each object are extracted to support the recognition process (size, shape, bounding box ...).
- **Ball detection :** At this step, the ball is selected from a set of objects. Bleachers and players are removed using their size and white lines are removed using their shape. Other artifacts (non existing entities) are deleted using the width/height ratio of their bounding box and their color.

(a) Input image.


(b) Soccer field segmentation.


(c) Label map after closure.

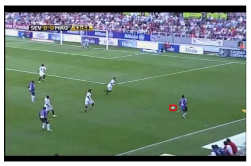

(d) The ball bounding box in red.

Fig. 3. The ball detection process.

The aim of our system is to generate a new video of a soccer game, focused on the RoI. By defining a zooming frame the new zoomed video is generated. This frame has to be centered on the ball and close enough to the ground to allow the watcher understanding what happens in the scene (See Figure 4).

Both aspects are controlled by parameters of position $C_i = (C_{ix}, C_{iy})$ and zoom factor Z_i of the zooming frame. The size of the zooming frame is computed with the following expression $\hat{S}_i = (1 - Z_i).S_i,\ Z_i \in [Z_{min}, Z_{max}],\ Z_{min}, Z_{max} \in [0, 1[$. S_i is the size of the original frame and $\hat{S}_i$ is the size of the zooming frame. The zoom factor is initialized at a defined value Z_0. The second parameter is the position of the zooming frame. Ball position $B_i = (B_{ix}, B_{iy})$ is detected in the current frame i to locate the region of interest of the scene. Then it is used to estimate the new position of the zooming frame. Frame Rate FPS $(frames\ per\ second)$ and the displacement of the ball $V_i = B_{i-1} - B_i$ are also used as parameters of the process.

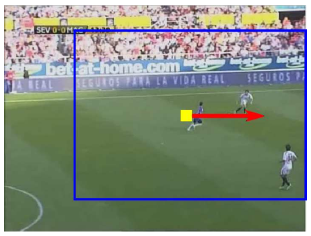


Fig. 4. The zooming frame in the blue rectangle. In the center of the rectangle, the displacement of the zooming frame.

Taking into account these parameters, the zoom factor and the zooming frame position are computed using the following expressions:

$$Z_i = Z_{i-1} + \frac{(1-\alpha)f(i) + \alpha(Z_{i-1} - Z_{i-2})}{FPS} \tag{1}$$

$$C_i = C_{i-1} + \frac{1}{\beta.n} \sum_{k=i-n+1}^{i} (B_k - B_{k-1}) \tag{2}$$

$$f(i) = \begin{cases} \rho_1, \text{ if } \theta_i \text{ and } V_i < s, & (\rho_1 \in]0, 0.1]) \\ \rho_2, \text{ if } \theta_i \text{ and } V_i \geq s, & (\rho_2 \in [-0.1, 0[) \\ \rho_3, \text{ otherwise}, & (\rho_3 \in [-0.1, 0[) \end{cases} \tag{3}$$

θ_i is true if a ball has been detected by the system. s is a threshold for the speed of the ball V_i. In order to provide a pleasant video to watch, visual artifacts have to be avoided such as jumping of the zooming frame position, zooming and dezooming effects. Thus a smoothing term is added to control the zooming frame trajectory and the zoom variation. The terms α and β are smoothing factors, the higher these factors are, the smoother the evolution of the zooming frame is. The ball position B_i is defined as the center of the original video if no ball is detected.

(a)

(b)

Fig. 5. Two different versions of the same video.

4.2. Automatic RoI extraction in videosurveillance scenarios

In this section, we shortly resume another RoI extraction method we used in a surveillance context in order to automatically produce a new zoomed version of the video stream. We want to find the RoI in order to crop the image on the specific area of the scene. These RoIs will be extracted by tracking the moving objects in the scene (The RoIs will follow the trajectories of the objects). Figure 6 presents examples of trajectories (the RoI is shown with a white rectangle).

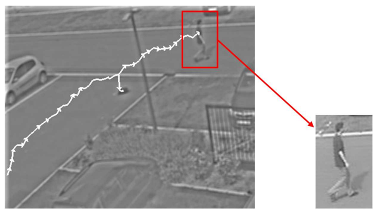


Fig. 6. Examples of trajectories.

The videosurveillance scenarios to be handled use fixed cameras. Therefore, it is possible to use specific image analysis methods such as mobile objects segmentation and tracking techniques [11]. Indeed, the common bottom-up approach for segmenting moving objects with fixed cameras uses both background estimation and foreground extraction of pixel values. We implemented the Mixture of Gaussian (MOG) [12] methods. In each frame of the video, mobile objects are detected using the picture segmentation. In order to find the trajectories, one should match objects between consecutive frames. This process, called object tracking, is implemented as proposed by the authors previously in [11].

From objects trajectories, it is possible to automatically produce a new zoomed version of video stream in order to highlight what is really happening in the watched scene (See Figure 7).

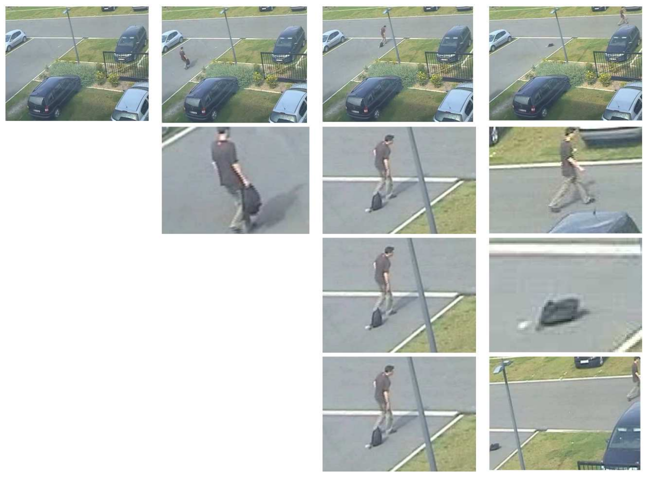


Fig. 7. New "zoomed versions" of video stream. First row is the original video stream. Second row is created when a first mobile object appears in the scene. Third row is created when a second object is detected and tracked (the abandoned luggage). Forth row is the stream that includes the two mobile objects.

5. RESULTS AND VALIDATION

The implementation of the streaming server is based on two different software: liveMedia streaming library and x264 video encoder. The video encoder is used in advance to prepare the sequences before placing them in the content database, in order to make them compatible with the temporal content pipelining. The streaming abilities are implemented using the liveMedia library[13] which has been extended to deliver H.264 encoded files, including the advanced feature for temporal content pipelining through the session control and the enhanced content creation unit. In section 5.1 we explain the constraints in the H.264 codec associated to the juxtaposition of video segments, while in section 5.2 we include some test parameters and subjective analysis over the scenario of section 4.1.

5.1. Cost of the Temporal Content Pipelining

The feature of the content juxtaposition is not completely costless. Several simulations have shown that the fact that the video sequence is segmented in several pieces, as described in section 3.3, doesn't have any effect in the fluency of the streaming playback. Neither interruptions nor decrements in the output from the server to the streaming session have been observed, when running the demo scenarios with enhanced video content. Although, when finishing a segment, a new video file has to be opened and included in the transmission buffer, the processing load is not dramatically increased in the server and the speed delivering the RTP packets is adjusted to maintain a flawless video experience in the client side.

However, some constraints are applied over the encoder H.264 and the consequent NAL units, in order to enable streaming and video content segmentation:

1. Some encoding techniques are enabled to provide a fluent stream. B frames are disabled in order to avoid bi-directional inter-prediction as all the video content is progressively streamed. Context-Adaptive Binary Arithmetic Coding (CABAC) is also disabled due to the fact that it considerably requires more processing to decode than Context-Adaptive Variable-Length Coding (CAVLC). Furthermore, as the target players are the mobile devices the Baseline Profile must be used to encode the files, therefore B frames and CABAC are automatically discarded.

2. The compression speed is clearly damaged as the encoding process of every sequence is divided in the multiple segments. Nevertheless, the scenarios we consider are based on *on demand* video content. Hence, all the segments are preprocessed and included in the video database *in advance*, and because of this, the performance of the global system is not damaged.

3. Every video segment starts with an Instantaneous Decoding Refresh (IDR) frame. Therefore, the segmentation in multiple pieces of every sequence constraints the maximum size of the *GOP* (Group of Pictures) to the size of the encoded segments. The result of using in excess the *IDR* frames is a global increment in the bitrate of the playback. For this reason, a trade-off between the time of the system's reaction response to the user's feedback and the minimum size of the *GOP* has to be achieved, as every segment has to be completely delivered before starting to send the new one (due to the constraint associated to switching the enhanced content in a temporally consistent way). If the segments are too small, the system switches the playback according to the user feedback very fast in all the cases. This is due to the fact that the whole segment can rapidly be sent, apart from the fact that the server could be delivering at that point the first frames of it. However the size of the *GOP* is also small increasing the bitrate. The opposite response is achieved if the segments are huge. In our simulations we used sequences encoded at 25 frames per second and video segmentation approximately every 15 frames. This way, the maximum delay switching of content, in terms of the latency in the response of the server after receiving the user's request, is approximately 600 milliseconds, while a *GOP* of 15 frames is good enough in order to not to trigger the global bitrate.

4. Finally, it is also important to consider the increment of bitrate due to the *SPS* and *PPS* headers that are used in every new video segment. In the case that all the video sequence is encoded once, they

Sequence dimensions	**Quantization Parameter**	**Bitrate increment (%)**
176x144	16	0,86
176x144	32	5,95
352x288	16	0,68
352x288	32	5,73
720x576	16	0,76
720x576	32	3,84

Table 1. Increment of bitrate using video segmentation

have to be sent to the client just one time at the beginning. This is not the case when the sequence is split in several segments as in our framework, in order to enable the pipelining feature in a temporally consistent way along all the playback. In table 1 we include the increment of bitrate for different video resolutions at different levels of quality (by modifying the quantization step: *Qp*).

As we can observe, the cost of the headers is very low and almost negligible for higher quality encoding parameters (*Qp*=16). The size of the header is almost constant in every case, independently of the encoding parameters that are being used. Hence, when the quality of the image is increased at the cost of spending bitrate, the related cost of the headers gets lower and lower.

The reader can find in the web *http://www.tele.ucl.ac.be/Walcomo* some demos of our interactive architecture to subjectively evaluate whether (s)he appreciates the switching capabilities of our prototype. In particular, for soccer game videos, our experiments monitor how often the user asks for a replay or a switch between resolutions. At the end of the test, the users also fills in a questionnaire about his/her perception of the switching/browsing facilities, compared to a regular visualization of sub-sampled content. In the videosurveillance context, a number of explicit questions are defined about the content of the sequence, and our test measures the time needed by the viewer to answer those questions, without and with the browsing capabilities offered by our interactive switching system.

5.2. Tests for worthy visual content scenarios

Our proposed method for enhancement of sport team video has been tested on a 2 hours soccer video with a 25fps frame rate. The resolution of the original video is 800x600 and the resolution of the target screen is 320x240. We used the following configuration: $\beta = 15$, $\alpha = 0.3$, $\rho_1 = 0.03$, $\rho_2 = -0.03$, $\rho_3 = -0.03$ and $n = 10$. We realized a visual subjective evaluation by comparing the video generated by our system and a video created by resizing the original to the resolution of the target screen. The new video is more pleasant and it is easier to understand what happens in the scene.

However some improvements are needed. Indeed, some situations are not optimal for our algorithm. The first one is when the ball is alone and no player follows it (the ball goes in touch for example). In this case, following the players could be more interesting than following the ball. But this situation happens rarely and does not last a long time. The second one is more problematic, it happens when the ball moves slowly during few times and then a player makes suddenly a big shot. It is difficult with a smooth motion to follow the ball when its speed changes suddenly. In this case, several frames are necessary to refocus the zooming frame on the ball.

6. CONCLUSIONS

In this paper, we described a flexible interactive streaming system, over one underlying key mechanism : temporal content pipelining, which allows to switch the video content at whichever point of the playback in a temporally consistent way. This mechanism uses the client's feedback, requiring only one open streaming session per user and no advanced implementation on the client's side. Experimental results show that the video segmentation doesn't affect the fluency of the video streaming and in addition, the bitrate of the encoded sequences is not significantly increased. Therefore, this mechanism doesn't damage the global performance of the streaming system.

We also present two different procedures to automatically generate enriched video content: one for soccer video games and the other for videosurveillance applications. Both clearly increase subjectively the perceived quality of the streaming experience. Furthermore, the personalized content selection between several video versions is offered to the user through the temporal content pipelining in real time.

Finally, the architecture presented is also capable of enabling targeted advertising, by implementing the concept of client profile. In addition to the dynamic interactive selection of enhanced video segments, the architecture is also designed to allow the insertion of promotional (or any other kind of video segments) in the middle of the main streaming playback. The playback can be resumed directly, through the unique streaming session, without any kind of interruption or efficiency cost. The content pipelining is in every case completely transparent for the viewer.

7. REFERENCES

[1] A. Argyriou and V. Madisetti, "Streaming h.264/avc video over the internet," in *Consumer Communications and Networking Conference*, Jan. 2004, pp. 169–174.

[2] M. F. Sayit and T. Tunah, "Video streaming with h.264 over the internet," in *Signal Processing and Communications Applications*, Apr. 2006, pp. 1–4.

[3] Li Zhi-Gang and Zhao-Yang Zhang, "Real-time streaming and robust streaming h.264/avc video," in *Third International Conference on Image and Graphics*, Dec. 2004, pp. 353–356.

[4] Xinguo Yu and Dirk Farin, "Current and emerging topics in sports video processing," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2005.

[5] Y. Takahashi, N. Nitta, and N. Babaguchi, "Video summarization for large sports video archives," *Multimedia and Expo, IEEE International Conference on*, vol. 0, pp. 1170–1173, 2005.

[6] E. Bomcke and C. De Vleeschouwer, "An interactive video streaming architecture for H.264/AVC compliant players," in *IEEE International Conference on Multimedia and Expo(ICME), New-York, USA*, 2009.

[7] ITU-T, "H.264: Advanced video coding for generic audiovisual services," *Series H : Audiovisual and multimedia systems*, 2005.

[8] T. Stockhammer M. Westerlund S. Wenger, M.M. Hannuksela and D. Singer, "RTP Payload Format for H.264 Video," RFC 3984 (Proposed Standard), Feb. 2005.

[9] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2326 (Proposed Standard), Apr. 1998.

[10] T. d'Orazio, C. Guaragnella, M. Leo, and A. Distante, "A new algorithm for ball recognition using circle hough transform and neural classifier," *PR*, vol. 37, no. 3, pp. 393–408, March 2004.

[11] X. Desurmont, A. Bastide, J. Czyz, C. Parisot, J-F. Delaigle, and B. Macq, "A general purpose system for distributed surveillance and communication," in *Intelligent Distributed Video Surveillance Systems*. 2006, S.A Velastin and P Remagnino Eds.

[12] Chris Stauffer and W. Eric L. Grimson, "Adaptive background mixture models for real-time tracking," in *CVPR*. 1999, pp. 2246–2252, IEEE Computer Society.

[13] "Live555 media server streaming application library's webpage," `http://www.live555.com/liveMedia/faq.html`.