A Real-time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC

Peter Lambert, Davy De Schrijver, Davy Van Deursen, Wesley De Neve, Yves Dhondt, and Rik Van de Walle

Department of Electronics and Information Systems – Multimedia Lab
Ghent University – IBBT
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
peter.lambert@ugent.be

Abstract. In many application scenarios, the use of Regions of Interest (ROIs) within video sequences is a useful concept. It is shown in this paper how Flexible Macroblock Ordering (FMO), defined in H.264/AVC as an error resilience tool, can be used for the coding arbitrary-shaped ROIs. In order to exploit the coding of ROIs in an H.264/AVC bitstream, a description-driven content adaptation framework is introduced that is able to extract the ROIs of a given bitstream.
The results of a series of tests indicate that the ROI extraction process significantly reduces the bit rate of the bitstreams and increases the decoding speed. In case of a fixed camera and a static background, the impact of this reduction on the visual quality of the video sequence is negligible. Regarding the adaptation framework itself, it is shown that in all cases, the framework operates in real time and that it is suited for streaming scenarios by design.

1 Introduction

In many application scenarios, the use of Regions of Interest (ROIs) within video sequences is a useful concept. A ROI typically is a region within the video pane containing visual information that is more interesting than the other parts of the video pane. In the case of multiple ROIs, they can be equally important or they might have different levels of importance. The remaining area is often called the background. Several image or video coding standards (e.g., JPEG2000 [1] or the Fine Granularity Scalability (FGS) Profile of MPEG-4 Visual [2]) have adopted the idea of ROIs and they often provide functionality to code the ROIs at a higher quality level.

The use of ROIs is, for instance, found in surveillance applications. For instance, more and more cameras are developed that capture 360 degrees of video footage with very high resolution pictures. Because it is often impossible to transmit a coded representation of the entire video sequence, one or more ROIs are defined and only a coded version of these smaller areas is transmitted. The position of the ROIs within the picture can mostly be adjusted in real time by

an operator. The latter avoids the delays that are introduced by traditional Pan Tilt Zoom (PTZ) cameras.

The currently ongoing standardization efforts of the Joint Video Team regarding Scalable Video Coding (SVC) [3] indicate that there is a clear interest in ROI coding and ROI-based scalability [4, 5]. The requirements document of SVC [6] gives some more details about various applications in which ROI coding and ROI-based scalability can be applied, including video surveillance and multi-point video conferencing.

This paper concentrates on the exploitation of ROI coding within the H.264/AVC specification [7]. The H.264/AVC standard does not explicitly define tools for ROI coding, but the authors have shown that the use of *slice groups* (also called Flexible Macroblock Ordering or FMO) enables one to code ROIs into an H.264/AVC bitstream. Nothwithstanding the fact that FMO is primarily an error resilience tool, it was illustrated in [8] that it can be the basis for content adaptation. The combination of ROI coding and a description-driven framework for the extraction of ROIs (ROI scalability as content adaptation) is the main topic of this paper. On top of this, it will be shown that the entire content adaptation framework operates in real time and that it is suited for live streaming scenarios. This technique illustrates the possibilities that are offered by the single-layered H.264/AVC specification for content adaptation. A similar technique for the exploitation of multi-layered temporal scalability within H.264/AVC is described in [9].

The rest of this paper is organized as follows. Section 2 describes the two main enabling technologies: H.264/AVC FMO and the XML-driven content adaptation framework. In Sect. 3, two methods for ROI extraction are introduced (background slice deletion and placeholder slice insertion). The results of a series of tests regarding the proposed content adaptation framework are given in Sect. 4 and, finally, Sect. 5 concludes this paper.

2 Enabling Technologies

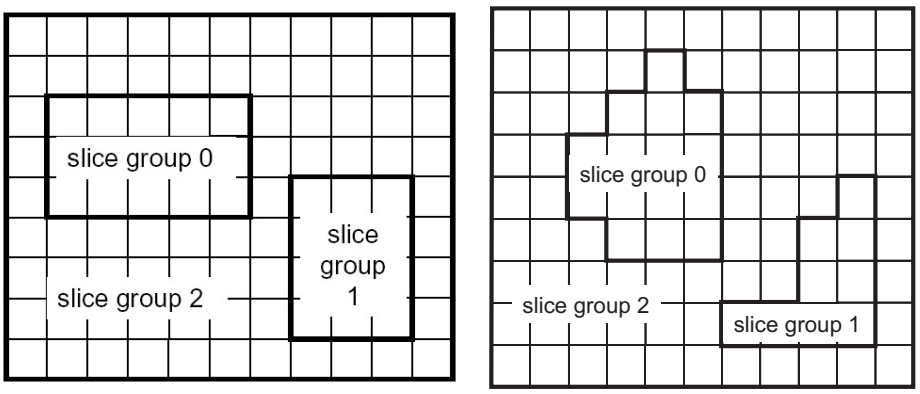
2.1 ROI Coding with H.264/AVC FMO

FMO is a novel tool for error resilience that is introduced in the H.264/AVC specification. By using FMO, it is possible to code the macroblocks of a picture in another order than the default raster scan order (i.e., row per row). One can define up to eight so-called *slice groups* and every macroblock can freely be assigned to one of these slice groups. This assignment results in a MacroBlock Allocation map (MBAmap), which is coded in a Picture Parameter Set (PPS). In fact, the set of slice groups constitute a *set partition*[1] of the set of macroblocks of a picture. An H.264/AVC encoder will encode one slice group after another and the macroblocks that are part of the slice group in question are coded in raster scan order (*within* that particular slice group). Apart from this, the concept of

[1] In a strictly mathematical sense.

(traditional) slices remains the same: macroblocks are grouped into slices, the latter being spatially limited to one of the slice groups.

Because the coding of the entire MBAmap might introduce a considerable amount of overhead, the H.264/AVC standard has specified 6 predefined types of FMO. The MBAmap for these types has a specific pattern that can be coded much more efficiently. FMO type 2, which is used in this paper, indicates that the slice groups are rectangular regions within the video pane, as shown in Fig. 1(a). This type of FMO only requires two numbers to be coded per rectangular slice group. These regions will be considered Regions of Interest. Note that the macroblocks that are left over also constitute a (non-rectangular) slice group. For a thorough overview of H.264/AVC FMO, the reader is refered to [10].



(a) H.264/AVC FMO type 2 (b) H.264/AVC FMO type 6

Fig. 1: ROI coding with H.264/AVC FMO

In H.264/AVC, the slice group configuration is coded in a PPS which contains a number of syntax elements that are the same for a certain number of successive pictures (e.g., the entropy coding scheme). Further, every slice contains a reference to the PPS that is into effect. Since the ROI configuration can change in the course of time (e.g., the relative position of a ROI changes, or a ROI appears or disappears), it is required to code a PPS into the bitstream in order to reflect every change of the ROI configuration. In such a PPS, there are four syntax elements that are important in the context of this paper. The number of slice groups is coded by means of `num_slice_groups_minus1` which means that this number denotes the number of ROIs that are present in the bitstream (the 'background' is also a slice group). The syntax element `slice_group_map_type` will always be 2 since we only focus on FMO type 2. Every rectangular slice group is defined by the macroblock numbers of its top left and its bottom right macroblock. These two numbers are coded in a PPS by means of the syntax elements `top_left_iGroup` and `bottom_right_iGroup`.

Finally, it should be noted that it is possible to define non-rectangular ROIs in H.264/AVC. Indeed, one can always use FMO type 6 (explicit coding of the MBAmap) to define arbitrary-shaped sets of macroblocks, as depicted in Fig. 1(b). The content adaptation framework, as presented in this paper, is able to process FMO type 6; the only modification that is needed, is the algorithm that decides if a slice is part of a ROI or not (see Sect. 3.1).

2.2 XML-driven Content Adaptation Framework

The process of content adaptation based on (pseudo) scalable properties of a bitstream typically requires the removal of certain data chunks, the replacement of certain data blocks, the modification of certain syntax elements, or a combination of these three. One way to accomplish this, is to make use of automatically generated XML descriptions (called Bitstream Syntax Descriptions, or BSDs) that contain high-level information about the bitstreams. For the generation of a BSD, only a limited knowledge is required about the syntax of given bitstream. In stead of performing the adaptations directly on the bitstreams, the generated BSDs can be transformed in such a way that it reflects the desired adaptation. The last step is to automatically generate an adapted bitstream based on the transformed description.

The MPEG-21 Bitstream Syntax Description Language (BSDL) framework is an example of a framework that provides the necessary functional blocks that are described above. However, it is described in literature that some parts of this framework have performance issues [11, 12]. As a result, it is (yet) less suited to be deployed in real-life scenarios which require real-time behavior. Another example is the Formal Language for Audio-Visual Object Representation, extended with XML features (XFlavor [13]). The major drawback of the latter is the fact that the generated descriptions are too large because it is required to fully parse the bitstream up to the lowest level (in fact, *all* information of the bitstream is present in its description). In order to combine the strenghts of both BSDL and XFlavor, the authors have developed BFlavor, which is a modification of XFlavor in order to be able to output BSDL-compatible descriptions [14].

BFlavor allows to describe the structure of a media resource in a C++-alike manner. It is subsequently possible to automatically create a BS Schema, as well as a code base for a parser that is able to generate a BSD that is compliant with the corresponding BS Schema. This implies that the generated BSDs can be further processed by the upstream tools in a BSDL-based adaptation chain. In Fig. 2, an overview is given of the BSD-oriented content adaptation framework, as employed in this paper. It is important to note that the adaptation (see 'filter(s)' in Fig. 2) is the only step in the chain of actions that is not automated. The technology that is used to transform the BSDs is Streaming Transformations for XML (STX, pronounced 'stacks') [15]. The internals of the transformation (embodying the actual ROI scalability) are the subject matter of Sect. 3.

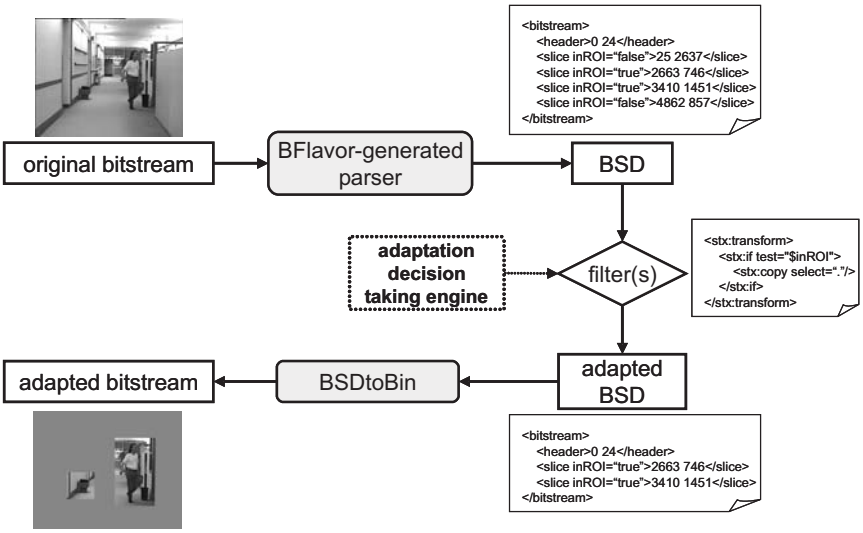


Fig. 2: XML-driven framework for video content adaptation

3 ROI Extraction

In the context of this paper, every ROI is a slice group (containing one or more slices). Consequently, the extraction of the ROIs comes down to the identification of those slices that are part of one of the ROIs. Afterwards, the 'background' can either be dropped or replaced with other coded data. These two approaches are described in the following two subsections. The bandwith required to transmit a bitstream that is disposed of its non-ROI parts will be much lower. On top of this, the use of placeholder slices (see Sect. 3.2) will result in a speed-up of the receiving decoder and a decrease in the decoder's complexity.

3.1 Non-ROI Slice Deletion

For every slice in the coded video sequence, one has to decide whether or not it is part of one of the rectangular slice groups. Based on the syntax element `first_mb_in_slice` (coded in every slice header), this can be done in the following manner. Let R_i be the ROIs and let S be a slice having the macroblock with number FMB_S as its first macroblock (i.e., $FMB_S = \texttt{first\_mb\_in\_slice}$). Further, let TL_i and BR_i be the macroblock numbers of the top left and bottom right macrobock of ROI R_i. Last, let W be the width of a picture in terms of macroblocks (coded by means of `pic_width_in_mbs_minus1` in a Sequence Parameter Set). Then, S is part of R_i if

$$(TL_i \bmod W \leq FMB_S \bmod W) \wedge (FMB_S \bmod W \leq BR_i \bmod W)$$
$$\wedge\, (TL_i \operatorname{div} W \leq FMB_S \operatorname{div} W) \wedge (FMB_S \operatorname{div} W \leq BR_i \operatorname{div} W)$$

In this expression, the div operator denotes the integer devision with truncation and the mod operator denotes the traditional modulo operation. Based on a BSD that is generated by BFlavor, this calculation can be done inside a STX filter. The latter will then discard all parts of the BSD that are related to slices for which the above calculation evaluates to false for all i. Based on this transformed BSD, the BSDtoBin Parser can generate the actual adapted bitstream. It is important to note that a bitstream that is generated by this approach will no longer comply with the H.264/AVC standard as the latter requires that *all* slice groups are present in an H.264/AVC bitstream. Despite the fact that only minor modifications of an H.264/AVC decoder are needed for the correct decoding of such a bitstream, this may be considered a disadvantage of the procedure described above.

3.2 Placeholder Slice Insertion

In order to avoid the disadvantages described in the previous subsection, the authors propose the use of placeholder slices. In this approach, coded P and B slices are no longer dropped, but they are replaced by other coded data. A placeholder slice can be defined as a slice that is identical to the corresponding area of a certain reference picture, or that is reconstructed by relying on a well-defined interpolation process between different reference pictures [16]. Based on the provisions of the H.264/AVC specification, the placeholder slices, as proposed here, are implemented by means of P slices in which all macroblocks are marked as skipped (hereafter called 'skipped P slices'). This subsection will explain how this substitution can be accomplished in the XML-driven content adaptation framework.

The most straightforward case is replacing coded P slices (both reference and non-reference) with skipped P slices. Since the slice header can be kept unchanged, only the slice data are to be substituted. Only two syntax elements are needed to code the slice data for a skipped P slice: `mb_skip_run` to indicate the number of macroblocks that are to be skipped and `rbsp_slice_trailing_bits` in order to get byte-aligned in the bitstream. An excerpt of both the original and adapted BSD of a P slice is given in Fig. 3(a) where some simplifications are introduced to improve readability and in order to meet the place constraints.

In order to replace a coded B slice with a skipped P slice, the substitution process is more complex because of the different nature, and hence header syntax, of P and B slices. The syntax element `slice_type` has to be changed from 1 or 6 (B slice) to 0 (P slice). Next to this, the slice header of a B slice contains a number of syntax elements that cannot appear in a P slice, and they need to be removed. To summarize, the STX filter which adapts the BSDs will remove the following syntax elements (and the syntax elements that are implied by them): `direct_spatial_mv_pred_flag`, `num_ref_idx_l1_active_minus1`, `ref_pic_list_reordering_flag_l1`, `luma_weight_l1_flag` (if applicable), and `chroma_weight_l1_flag` (if applicable).

Regarding the slice data, the same process can be applied as in the case of coded P slices. An example illustrating this scenario is given in Fig. 3(b).

```
---------------- original description ----------------
<coded_slice_of_a_non_IDR_picture>
  <slice_layer_without_partitioning_rbsp>
    <slice_header>
      <first_mb_in_slice>0</first_mb_in_slice>
      <slice_type>5</slice_type>
      <pic_parameter_set_id>0</pic_p...>
      <frame_num>1</frame_num>
      <!-- ... -->
    </slice_header>
    <slice_data>
      <bit_stuffing>7</bit_stuffing>
      <slice_payload>7875 1177</slice_payload>
    </slice_data>
  </slice_layer_without_partitioning_rbsp>
</coded_slice_of_a_non_IDR_picture>

---------------- adapted description ----------------
<coded_slice_of_a_skipped_non_IDR_picture>
  <skipped_slice_layer_without_partitioning_rbsp>
    <slice_header>
      <first_mb_in_slice>0</first_mb_in_slice>
      <slice_type>5</slice_type>
      <pic_parameter_set_id>0</pic_p...>
      <frame_num>1</frame_num>
      <!-- ... -->
    </slice_header>
    <skipped_slice_data>
      <mb_skip_run>108</mb_skip_run>
    </skipped_slice_data>
    <rbsp_trailing_bits>
      <rbsp_stop_one_bit>1</rbsp_stop_one_bit>
      <rbsp_alignment_zero_bit>0</rbsp_a...>
    </rbsp_trailing_bits>
  </skipped_slice_layer_without_partitioning_rbsp>
</coded_slice_of_a_skipped_non_IDR_picture>
```

(a) P slice replaced by a skipped P slice

```
---------------- original description ----------------
<coded_slice_of_a_non_IDR_picture>
  <slice_layer_without_partitioning_rbsp>
    <slice_header>
      <first_mb_in_slice>0</first_mb_in_slice>
      <slice_type>6</slice_type>
      <pic_parameter_set_id>1</pic_parameter_set_id>
      <frame_num>2</frame_num>
      <pic_order_cnt_lsb>2</pic_order_cnt_lsb>
      <direct_spatial_mv_pred_flag>1</direct...>
      <num_ref_idx_active_override_flag>1</num...>
      <num_ref_idx_l0_active_minus1>1</num...>
      <num_ref_idx_l1_active_minus1>0</num...>
      <ref_pic_list_reordering_flag_l0>0</ref...>
      <ref_pic_list_reordering_flag_l1>0</ref...>
      <slice_qp_delta>2</slice_qp_delta>
    </slice_header>
    <slice_data>
      <bit_stuffing>6</bit_stuffing>
      <slice_payload>9543 851</slice_payload>
    </slice_data>
  </slice_layer_without_partitioning_rbsp>
</coded_slice_of_a_non_IDR_picture>

---------------- adapted description ----------------
<coded_slice_of_a_skipped_non_IDR_picture>
  <skipped_slice_layer_without_partitioning_rbsp>
    <slice_header>
      <first_mb_in_slice>0</first_mb_in_slice>
      <slice_type>0</slice_type>
      <pic_parameter_set_id>1</pic_parameter_set_id>
      <frame_num>2</frame_num>
      <pic_order_cnt_lsb>2</pic_order_cnt_lsb>
      <num_ref_idx_active_override_flag>1</num...>
      <num_ref_idx_l0_active_minus1>1</num...>
      <ref_pic_list_reordering_flag_l0>0</ref...>
      <slice_qp_delta>0</slice_qp_delta>
    </slice_header>
    <skipped_slice_data>
      <mb_skip_run>264</mb_skip_run>
    </skipped_slice_data>
    <rbsp_trailing_bits>
      <rbsp_stop_one_bit>1</rbsp_stop_one_bit>
      <rbsp_alignment_zero_bit>0</rbsp...>
    </rbsp_trailing_bits>
  </skipped_slice_layer_without_partitioning_rbsp>
</coded_slice_of_a_skipped_non_IDR_picture>
```

(b) B slice replaced by a skipped P slice

Fig. 3: XML-driven placeholder slice insertion

In order to save some additional bits, it is possible to change the value of the syntax element `slice_qp_delta` to zero in all cases, as this value has no impact on skipped macroblocks.

4 Results

In order to have some insight in the performance and the consequences of the proposed architecture, a series of tests was set up. The measurements include the impact of the adaptation process on the bitstream and on the receiving decoder. Also an assessment of the performance of the overall adaptation framework is given.

In the experiments, four video sequences were used: Crew (600 pictures with a resolution of 1280 × 720), Hall Monitor, News, and Stefan (the latter three having 300 pictures at CIF resolution). In each sequence, one or more ROIs were manually defined: the moving persons in Hall Monitor and the bag that is left behind by the left person; the heads of the two speakers in News; the tennis

player in Stefan; the first two persons of the crew and the rest of the crew as a separate ROI in the Crew sequence. In all sequences, the ROIs are non-static (moving, shrinking, or enlarging) and they may appear or dissappear.

These four sequences were encoded with a modified version of the H.264/AVC reference software (JM 9.5) which allows to encode bitstreams with FMO configurations that vary in the course of time. This encoding was done once conform the Baseline Profile and once conform the Extended Profile (the only difference here being the use of B slices). Other relevant encoding parameters are a GOP length of 16, 2 consecutive B slice coded pictures (if applicable), and a constant Quantization Parameter (QP) of 28. Some properties of the resulting bitstreams are summarized in Table 1. In this table, also the impact of the adaptation process on the size of the bitstreams is given: size_p denotes the size of the adapted bitstreams in which placeholder slices were inserted while size_d denotes the size of the adapted bitstreams of which all background P and B slices are dropped.

Table 1: Bitstream characteristics (sizes in KB)

sequence		# ROIs	# PPSs	# slices	size	size_p	size_d
IP	crew	1–3	48	2020	9641	3448	3441
	hall monitor	1–3	26	924	629	377	374
	news	2	3	904	478	241	237
	stefan	1	31	632	2071	948	945
IBBP	crew	1–3	48	2020	9313	3507	3500
	hall monitor	1–3	26	924	610	381	377
	news	2	3	904	503	241	238
	stefan	1	31	632	2286	1024	1021

The bitstream sizes clearly indicate that the adaptation process (i.e., ROI extraction) considerably reduces the bit rate required to transmit a bitstream. Both extraction methods (placeholder insertion and background deletion) yield bit rate savings from 38% up to 64%. This reduction has in general a serious impact on the quality of the decoded video sequence. Because the coded background P and B slices are discarded or replaced, a correct picture is only decoded at the beginning of every GOP, resulting in bumpiness of the sequence in which the ROIs are moving smoothly. However, because coded macroblocks inside a ROI can have motion vectors pointing outside the ROIs, 'incorrect' decoded data of the background can seep into the ROI which results in erroneous borders of the ROI. This can be avoided by applying so-called *constraint motion estimation* at the encoder so that motion vectors only point to the same slice group the macroblock being predicted belongs to. ROIs that are coded in this way are sometimes called *isolated regions* (this was not used in the tests).

With respect to the (negative) impact of the adaptation process on the received visual quality, there are situations in which this impact is negligible. An example of such a situation is the sequence Hall Monitor in which both the camera and the background are static. The average PSNR-Y of the adapted version is 36.7 dB whereas the unadapted version had an average PSNR-Y of 37.7 dB (or

38.0 dB in case B slices were used). When watching the adapted version, even an expert viewer can hardly notice that the bitstream was subject of an adaptation process. In case of video conferencing or video surveillance applications, this opens up new opportunities. For instance, bitstreams that are coded with ROIs using H.264/AVC FMO can sustain a rather big decrease in available bandwidth without any noticable quality loss. This, of course, on condition that the transporting network first 'drops' the background packets. Alternatively, there might be an active network node (implementing the adaptation framework as presented in this paper) which adapts the bitstreams by removing or replacing the coded information of the background.

Because the processing of P-skipped macroblocks requires less operations for a decoder, it is expected that a decoder, receiving an adapted bitstream with placeholder slices, operates faster compared to the case of decoding the original bitstream. Indeed, for the decoding of a P-skipped macroblock, a decoder can rely directly on its decoded picture buffer without performing any other calculations such as motion compensation. Both the original and the adapted bitstreams were decoded five times using the reference decoder (JM 10.2) in order to measure the decoding speed. The average decoding speed for each bitstream is given in Table 2.

Table 2: Impact on decoding speed (frames per second)

sequence		original	placeholders
IP	crew	1.5	2.0
	hall monitor	15.6	16.9
	news	16.7	18.9
	stefan	10.4	14.9
IBBP	crew	1.1	1.3
	hall monitor	13.8	17.0
	news	14.3	17.5
	stefan	9.5	14.4

As can be seen from this table, the decoding speed is positively affected in all cases when placeholder slices are inserted by the adaptation process. The decoding speed in the cases the background was dropped, depends to a great extent on how a receiving decoder copes with non-arriving slices. If a decoder does nothing in case of missing slices, the decoding speed should be higher than the speeds of Table 2. If a decoder performs an error concealment algorithm, the decoding speed will decrease if the applied algorithm is more complex than decoding P-skipped macroblocks (e.g., spatial interpolation techniques).

The last part of this results section is about the performance of the overall adaptation framework. Both the memory consumption and the execution speed are substantial factors for the successful deployment of such an adaptation framework. Therefore, it is important to have an assessment of those factors with respect to the three main components of the adaptation framework as presented in this paper: the generation of BSDs by a BFlavor-generated parser, the transformation of BSDs using STX, and the generation of adapted bitstreams

by means of the BSDtoBin Parser. Regarding the memory consumption, it is reported in literature that all components give evidence of a low memory footprint and a constant memory usage [11, 12]. As such, the proposed framework satisfies the memory consumption requirements.

With respect to the execution times of the adaptation framework, every component was executed 11 times both for the placeholder slice insertion method and the background deletion. For all cases, the averages of the last 10 runs are summarized in Table 3. This averaging eliminates possible start-up latencies due to the fact that all components rely on a Java Virtual Machine as their execution environment. In Table 3, the execution speed is given in terms of Network Abstraction Layer Units (NALUs) per second, as a NALU is the atomic parsing unit within the framework. Note that the number of NALUs per picture depends on the slice group configuration. Combining the execution speed of the individual components for both content adaptation methods results in the overall execution speed in terms of frames per second (fps), as denoted in the last two columns of the table.

Table 3: Performance of the overall framework

		NALUs per second					total fps	
sequence		BFlavor	STX_p	STX_d	$BSDtoBin_p$	$BSDtoBin_d$	placeholders	dropping
IP	crew	1036.3	273.0	308.7	449.2	572.9	43.3	49.9
	hall monitor	2151.1	235.4	272.2	340.3	422.9	46.7	54.9
	news	2371.8	260.6	302.4	344.3	421.5	46.3	54.4
	stefan	1084.5	199.8	264.6	272.2	347.4	49.4	62.4
IBBP	crew	1038.7	221.1	245.5	397.2	497.2	37.1	42.1
	hall monitor	2090.1	200.3	226.0	308.8	385.3	41.0	47.6
	news	2306.4	249.2	284.5	311.7	381.1	43.4	50.5
	stefan	1016.3	155.3	179.1	253.5	325.8	41.8	49.3

It is clear from this table that the proposed framework is capable to perform the content adaptation in real time in all cases (see 'total fps'). As would be expected, the framework operates slower when performing the placeholder slice insertion because this method requires a more complex transformation in the XML domain. On top of that, the use of B slices also leads to a slow-down in both methods. These two trends can be observed in each component. Notwithstanding the fact that STX is a transformation language that overcomes most performance issues that are encountered when using, for instance, Extensible Stylesheet Language Transformation (XSLT), the transformation in the XML domain still is the slowest component in the framework.

All components of the proposed framework are capable of operating in video streaming scenarios. Indeed, both STX and BSDtoBin are entirely based on SAX events. Although the BFlavor-generated parser currently reads from and writes to a file, it can very easily be modified so that the generated classes use adequate buffers. This streaming capability, and also the performance measurements described above, prove that the proposed framework for the exploitation of ROI scalability within the H.264/AVC specification is suited for real-time

video streaming scenarios. This, of course, provided that the identification of the ROIs (motion detection and object tracking) is also done in real time by the encoder.

5 Conclusions

In this paper, it was shown how ROI coding can be accomplished within the H.264/AVC video coding specification by making use of Flexible Macroblock Ordering. For the extraction of the ROIs (i.e., exploitation of ROI scalability), a description-driven content adaptation framework was introduced that combines the BFlavor framework for the generation of BSDs, STX for the transformation of these BSDs, and the BSDtoBin Parser of the MPEG-21 BSDL framework for the generation of adapted bitstreams. Two methods for ROI extraction were implemented in this framework by means of a STX filter: removal of the non-ROI parts of a bitstream and the replacement of the coded background with placeholder slices.

Bitstreams that are adapted by this ROI extraction process have a significantly lower bit rate than the original version. While this has in general a profound impact on the quality of the decoded video sequence, this impact is marginal in case of a fixed camera and a static background. This observation may lead to new opportunities in the domain of video surveillance or video conferencing where the described approach can from the basis for certain levels of QoS. Next to the decrease in bandwidth, the adaptation process has a positive effect on the receiving decoder: because of the easy processing of placeholder slices, the decoding speed increases.

It was shown that the content adaptation framework, as presented in this paper, operates in real-time. Because each component of the framework is able to function in case of actual streaming video, the framework is suited also suited for live streaming video applications. As such, the framework can be deployed in an active network node, for instance at the edge of two different networks.

Acknoledgements

The research activities as described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), the Belgian Federal Science Policy Office (BFSPO), and the European Union.

References

1. Taubman, D., Marcellin, M.: JPEG2000 : Image Compression Fundamentals, Standards and Practice. Kluwer Academic Publishers (2002)
2. Li, W.: Overview of fine granularity scalability in MPEG-4 video standard. IEEE Trans. Circuits Syst. Video Technol. **11** (2001) 301–317

3. Reichel, J., Schwarz, H., Wien, M.: Joint scalable video model JSVM-4. JVT-Q202, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q202.zip (2005)
4. Yin, P., Boyce, J., Pandit, P.: FMO and ROI scalability. JVT-Q029, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q029.doc (2005)
5. Thang, T.C., Kim, D., Bae, T.M., Kang, J.W., Ro, Y.M., Kim, J.G.: Show case of ROI extraction using scalability information SEI message. JVT-Q077, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q077.doc (2005)
6. ISO/IEC JTC1/SC29/WG11, .: Applications and requirements for scalable video coding. N6880, http://www.chiariglione.org/mpeg/working_ documents/mpeg-04/svc/requirements.zip (2005)
7. Wiegand, T., Sullivan, G.J., Bjøntegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard. IEEE Trans. Circuits Syst. Video Technol. **13** (2003) 560–576
8. Dhondt, Y., Lambert, P., Notebaert, S., Van de Walle, R.: Flexible macroblock ordering as a content adaptation tool in H.264/AVC. In: Proceedings of the SPIE/Optics East conference, Boston (2005)
9. De Neve, W., Van Deursen, D., De Schrijver, D., De Wolf, K., Van de Walle, R.: Using bitstream structure descriptions for the exploitation of multi-layered temporal scalability in H.264/AVC's base specification. Lecture Notes in Computer Science, PCM 2005 (2005) 641–652
10. Lambert, P., De Neve, W., Dhondt, Y., Van de Walle, R.: Flexible macroblock ordering in H.264/AVC. Journal of Visual Communication and Image Representation **17** (2006) 358–375
11. Devillers, S., Timmerer, C., Heuer, J., Hellwagner, H.: Bitstream syntax description-based adaptation in streaming and constrained environments. IEEE Trans. Multimedia **7** (2005) 463–470
12. De Schrijver, D., Poppe, C., Lerouge, S., De Neve, W., Van de Walle, R.: MPEG-21 bitstream syntax descriptions for scalable video codecs. Multimedia Systems **article in press** (2006)
13. Hong, D., Eleftheriadis, A.: Xflavor: bridging bits and objects in media representation. In: Proceedings of the International Conference on Multimedia and Expo (ICME), Lausanne, Switzerland (2002)
14. Van Deursen, D., De Neve, W., De Schrijver, D., Van de Walle, R.: BFlavor: an optimized XML-based framework for multimedia content customization. In: Proceedings of the Picture Coding Symposium 2006 (PCS 2006), accepted for publication (2006)
15. Cimprich, P.: Streaming transformations for XML (STX) version 1.0 working draft. http://stx.sourceforge.net/documents/spec-stx-20040701.html (2004)
16. De Neve, W., De Schrijver, D., Van de Walle, D., Lambert, P., Van de Walle, R.: Description-based substitution methods for emulating temporal scalability in state-of-the-art video coding formats. In: Proceedings of WIAMIS, Korea, accepted for publication (2006)