USING PLACEHOLDER SLICES AND MPEG-21 BSDL FOR ROI EXTRACTION IN H.264/AVC FMO-ENCODED BITSTREAMS

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

Department of Electronics and Information Systems – Multimedia Lab
Ghent University – IBBT
Gaston Crommenlaan 8/201, B-9050 Ledeberg-Ghent, Belgium
Email: {peter.lambert, wesley.deneve, davy.deschrijver, yves.dhondt, rik.vandewalle}@ugent.be

Keywords: content adaptation, FMO, H.264/AVC, MPEG-21 BSDL, placeholder slices, ROI, video coding.

Abstract: The concept of Regions of Interest (ROIs) within a video sequence is useful for many application scenarios. This paper concentrates on the exploitation of ROI coding within the H.264/AVC specification by making use of Flexible Macroblock Ordering. It shows how ROIs can be coded in an H.264/AVC compliant bitstream and how the MPEG-21 BSDL framework can be used for the extraction of the ROIs.
The first type of ROI extraction that is described, is simply dropping the slices that are not part of one of the ROIs. The second type is the replacement of these slices with so-called placeholder slices, the latter being implemented as P slices containing only macroblocks that are marked as 'skipped'. The exploitation of ROI scalability, as achieved by the presented methodology, illustrates the possibilities that are offered by the single-layered H.264/AVC specification for content adaptation.
The results show that the bit rate needed to transmit the adapted bitstreams can be reduced significantly. Especially in the case of a static camera and a fixed background, this bit rate reduction has very little impact on the visual quality. Another advantage of the adaptation process is the fact that the execution speed of the receiving decoder fairly increases.

1 INTRODUCTION

The concept of Regions of Interest (ROIs) within a video sequence is useful for many application scenarios. A ROI is an area within the video pane that usually contains visual information that is more important or interesting than the other parts of the video image. If one or more ROIs are defined in a video sequence, they can be used to steer the bit allocation algorithm in such way that the ROIs are coded with a higher quality than the 'background'. This functionality is part of the JPEG2000 standard for still images (Taubman and Marcellin, 2002). The Fine Granularity Scalability (FGS) profile of MPEG-4 Visual (Li, 2001) also has provisions to support the coding of a ROI at a higher quality level.

Besides the fact that the idea of ROIs is adopted by various (standardized) coding schemes to provide different levels of quality within one picture, there are many applications that can benefit from the clever use of ROI coding. In the domain of video surveillance for instance, cameras are developed that capture 360 degrees of video footage resulting in high resolution pictures. Within these large pictures, a ROI is defined and only a coded representation of that area is transmitted over the network in order to reduce the required bandwith. The relative location of the ROI and its size can often be adjusted by an operator in real time. This technique was developed to avoid the delays that are introduced by traditional Pan Tilt Zoom (PTZ) cameras.

The domain of video conferencing is another domain where the use of ROIs can have advantages. In these scenarios, the ROI itself is easy to detect and its position is rather fixed in time. Next to this, the background is virtually always fixed and semantically unimportant; this is in contrast with video surveillance where the background can contain a lot of motion. For instance, such a video conferencing system is deployed in the European Parliament where every speaker is recorded in close-up.

With the emergence of standardization efforts by the Joint Video Team (JVT) regarding Scalable Video Coding (SVC) (Reichel et al., 2005), it has become clear that there is a broad interest in ROI coding and ROI-based scalability (Ichimura et al., 2005; Yin et al., 2005; Thang et al., 2005). Applications that are often mentioned in this context include video surveil-

lance (real-time monitoring) and multi-point video conference. More details about ROI coding and scalability can be found in the requirements document of SVC (ISO/IEC JTC1/SC29/WG11, 2005).

This paper concentrates on the exploitation of ROI coding within the H.264/AVC specification (Wiegand et al., 2003). Notwithstanding the fact that the H.264/AVC standard does not explicitly define a system for ROI coding, the authors have shown that the use of *slice groups* (often called Flexible Macroblock Ordering or FMO) enables an encoder to perform ROI coding (Lambert et al., 2006). Furthermore, it was illustrated that this approach can form the basis for content adaptation (Dhondt et al., 2005). The exploitation of ROI scalability, as well as the exploitation of multi-layered temporal scalability (De Neve et al., 2005), illustrates the possibilities that are offered by the single-layered H.264/AVC specification for content adaptation. The way the MPEG-21 BSDL framework can be used for this content adaptation process, is elaborated in later sections.

In this paper, the authors show how H.264/AVC FMO is to be used to encode one or more ROIs; how the MPEG-21 BSDL framework can be applied to extract the coded ROIs from a given bitstream; how placeholder slices can be used for the replacement of the background slices (not belonging to a ROI); and what the benefits are of this approach by means of some experimental results.

The rest of the paper is organized as follows. The two main enabling technologies, FMO and MPEG-21 BSDL, are described in Section 2 and 3. Section 4 describes the actual ROI extraction process. The concept of placeholder slices is introduced in Section 5. The experimental results are presented in Section 6 while Section 7 draws the conclusion of this paper.

2 ROI-CODING WITH FMO

Flexible Macroblock Ordering is one of the new error resilience tools that is defined within the H.264/AVC standard. Conceptually, it creates an additional level in the hierarchy from picture to macroblock. When applying FMO, a picture is made up of maximally 8 *slice groups*, and every slice group contains one or more slices. Finally, a slice is a collection of macroblocks. The most important aspect of FMO is the fact that every macroblock can be assigned individually to one of the slice groups of a picture, in contrast with the default raster scan order. This results in a so-called *MacroBlock Allocation map* (MBAmap) which is coded in a Picture Parameter Set (PPS). When this map is constructed, an encoder will code the macroblocks of a slice group in raster scan order (within that particular slice group), and they can be further grouped into slices.

Besides the coding of the entire MBAmap, the H.264/AVC standard has specified 6 predefined types of FMO. For these types, the MBAmap has a specific pattern that can be coded much more efficiently. In this paper, we focus on FMO type 2 (this only requires two numbers to be coded for each slice group – see later). This means that the slice groups of a picture are rectangular regions within the video pane, as shown in Figure 1. We will consider these regions as Regions of Interest. A more in-depth overview of FMO is given in (Lambert et al., 2006).

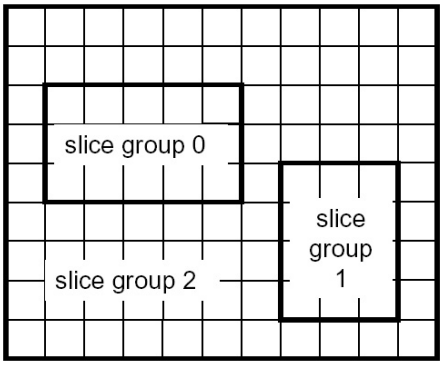

Figure 1: H.264/AVC FMO type 2.

While many applications would implement automatic ways to define ROIs (e.g., by using image processing techniques), the ROIs in the context of this paper were defined manually. To be more precise, an in-house defined syntax for a configuration file was developed that is parsed by the encoder; the latter being a modification of the H.264/AVC reference encoder (JM 9.5). This configuration file contains the macroblock numbers of the top left and the bottom right macroblock of the ROIs. These macroblock numbers are obtained by using a self-developed application that allows one to graphically select one or more rectangular regions within a video sequence on a picture-by-picture basis.

Since the slice group configuration is coded in a Picture Parameter Set, a PPS is inserted into the bitstream before every picture in which the ROI configuration changes (e.g., a ROI is moving across the video pane, a ROI is appearing or disappearing). Every slice has a reference to the PPS that is applicable for the picture the slice belongs to. There are four syntax elements of a PPS that are important in the context of this paper. The syntax element `num_slice_groups_minus1` indicates the number of slice groups. In this paper, the number of ROIs is equal to the number of slice groups minus one (the 'background' also is a slice group). Because we only discuss FMO type 2, the syn-

tax element `slice_group_map_type` is always equal to 2. Finally, for every slice group, the macroblock numbers of the top left and the bottom right macroblock of that slice group are coded by means of the syntax elements `top_left_iGroup` and `bottom_right_iGroup`.

3 MPEG-21 BSDL

In order to be able to deliver (scalable) video in a heterogeneous environment, it is important to be aware of the need of complementary logic that makes it possible to adapt the bitstream or to exploit the scalability properties of the bitstream. This bitstream adaptation process typically involves the removal of certain data blocks and the modification of the value of certain syntax elements.

One way to realize this, is to rely on automatically generated XML-based descriptions that contain information about the high-level structure of (scalable) bitstreams. These descriptions can subsequently be the subject of transformations, reflecting the desired adaptations of the bitstream. Lastly, they can then be used for the automatic generation of an adapted version of the bitstream in question.

The Bitstream Syntax Description Language (BSDL), part of the Digital Item Adaptation standard (DIA) of MPEG-21 (Vetro and Timmerer, 2003), is a language that provides solutions for discovering the structure of a multimedia resource resulting in an XML description (called a Bitstream Syntax Description, or BSD) and for the generation of an adapted multimedia resource using a transformed description. In Figure 2, the entire chain of actions within the BSDL framework is given.

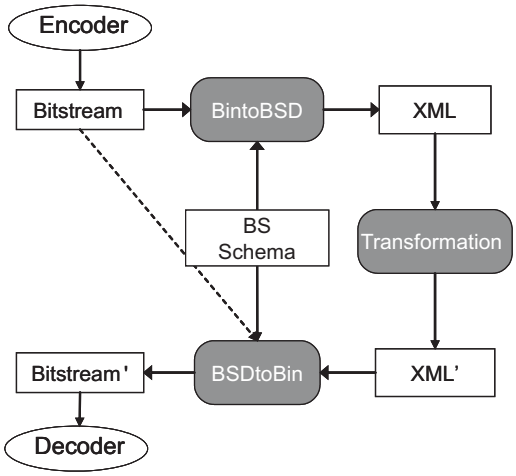

Figure 2: The MPEG-21 BSDL framework.

As illustrated by the figure, one starts from a given bitstream that is encoded with a certain codec. Dependent on the codec, a Bitstream Syntax (BS) Schema is developed that represents the high-level structure of bitstreams generated by the codec in question. Note that the granularity can be chosen freely, often dependent on the application. The language of a BS Schema is standardized in the MPEG-21 DIA specification, as is the functioning of the BintoBSD tool. Once a bitstream description is available in XML, it can be transformed based on, for instance, the characteristics of the network or the consuming terminal.

The way the BSD transformation is to be realized is, however, not defined by the BSDL specification. For instance, one can use Extensible Stylesheet Language Transformations (XSLT, (Kay, 2001)); Streaming Transformations for XML (STX, (Cimprich, 2004)) or an implementation based on an XML API (such as Simple API for XML, SAX, or Document Object Model, DOM).

The final step in the framework for media resource adaptation is the regeneration of the adapted bitstream. This is realized by the BSDtoBin tool, taking as input the adapted description, the BS Schema, and the original bitstream. The latter is needed because the coded data is not part of the BSDs. The functioning of this tool is also standardized within the BSDL specification.

As such, the BSDL framework provides all necessary ingredients for the construction of a generic (i.e., format-agnostic) and modular content adaptation engine: a BSD transformation engine and the format-agnostic bitstream generation by means of the BSDtoBin Parser. In practice, this is supplemented with an Adaptation Decision Taking Engine (ADTE) to steer and control the adaptation process. If such a framework is to support a new coding format, then only a new BS Schema is to be uploaded, accompanied by some transformations. The actual software (the BintoBSD Parser and the BSDtoBin Parser) does not have to be modified.

4 ROI EXTRACTION

The extraction of ROIs that are defined within a coded video sequence goes together with the deletion of the ‘background’ or the replacement thereof by other coded data. When a bitstream is disposed of its non-ROI coded parts, the required bandwidth to transmit the bitstream will be much lower. Next to this, the use of placeholder slices (see Section 5) results in a speed-up of the decoder, a decrease of the decoder complexity, and an easing of the decoder’s memory management (e.g., cache behavior). As will be shown in later sections, this extraction, replace-

ment, and adjustment process can even be used to transform a bitstream conforming to the H.264/AVC Extended Profile to a bitstream that is compliant with the H.264/AVC Baseline Profile.

In order to extract the ROIs from an FMO type 2 coded H.264/AVC bitstream, one has to decide for every slice if that particular slice is part of one of the rectangular slice groups. Every slice header contains the syntax element `first_mb_in_slice`. This number can be used to determine whether or not this slice is part of one of the rectangular slice groups. This can be done in the following manner.

Let S be a slice of a picture containing a number of ROIs R_i and let FMB_S be the macroblock number of the first macroblock of slice S (`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. Finally, let W be the width of a picture in terms of macroblocks (`pic_width_in_mbs_minus1`, coded in a Sequence Parameter Set). Then, S is part of R_i if

$$\begin{aligned} & (TL_i \bmod W \leq FMB_S \bmod W) \\ \wedge \quad & (FMB_S \bmod W \leq BR_i \bmod W) \\ \wedge \quad & (TL_i \operatorname{div} W \leq FMB_S \operatorname{div} W) \\ \wedge \quad & (FMB_S \operatorname{div} W \leq BR_i \operatorname{div} W) \end{aligned}$$

Note that the div operator denotes the integer devision with truncation and the mod operator denotes the traditional modulo operation. In other words, S is part of R_i if its first macroblock is located inside the rectangular region defined by R_i. After a BSD is generated by the BSDL framework, the above calculation can be performed by means of an XPath expression within an XSL Transformation. The result of such a transformation is a BSD in which the descriptions of non-ROI slices are removed. The BSDtoBin Parser can then generate the adapted bitstream.

It is important to note, however, that a bitstream obtained in this way is not compliant with the H.264/AVC standard because the latter specifies that all slice groups need to be present in an H.264/AVC bitstream. This means that an H.264/AVC compliant decoder will not guarantee the correct decoding of such a bitstream. Notwithstanding the fact that only minor modifications of an H.264/AVC decoder are needed for the correct decoding of the adapted bitstream (thanks to the independent nature of slices in H.264/AVC), this may be considered a disadvantage or gracelessness of the procedure described above. It should also be noted that the requirement that all slice groups need to be coded is proposed to be relaxed in the currently developed SVC specification (Reichel et al., 2005).

5 PLACEHOLDER SLICES

In order to avert any deviation of the H.264/AVC specification, as explained in the previous section, the authors propose the use of placeholder slices. 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 (De Neve et al., 2006). This implies that only a very limited amount of information has to be stored or transmitted. In this paper, placeholder slices are used to fill up the gaps that are created in a bitstream due to the removal of certain background slices. Taking into account the appropriate provisions of the H.264/AVC specification, these placeholder slices are in this paper implemented by means of P slices in which all macroblocks are marked as skipped (skipped P slices).

How the XML-driven content adaptation approach can be exerted for the substitution of coded background P or B slices by skipped P slices, is embroidered here. We first focus on background P slices (both reference and non-reference), as this is the most straightforward case. In this situation, there is no need to change any of the syntax elements of the NAL unit header or the slice header: only the actual slice data are to be substituted. The necessary slice data for the definition of a skipped P slice is the number of skipped macroblocks in that slice (by means of the syntax element `mb_skip_run`). Additionally, the slice layer has to be filled with a number of trailing bits (by means of the syntax element `rbsp_slice_trailing_bits`) in order to get byte-aligned in the bitstream. The XML descriptions of both the original and the adapted P slice are given in Figure 3. Note that some simplifications were introduced in the code in order to meet the place constraints and to improve the readability.

Regarding the syntax element `mb_skip_run`, it should be noted that the value of this element depends on the size of the slice being replaced. Since the ROIs are varying in time, the number of macroblocks in the slices of the background is not the same for every picture. Without loss of generality, consider the case that there is only one slice in the background slice group. The number of macroblocks in this slice is equal to the total number of macroblocks in a picture (denoted by `PicSizeInMbs`) minus the number of macroblocks that are contained by the various ROIs in that picture. Using the notation of Section 4, it is rather straighforward to calculate this number using TL_i, BR_i and W (taking into account the fact that ROIs may overlap each other). If there are multiple slices in the background slice group, then they can all be substituted by a single skipped P slice containing the same number of skipped macroblocks as in the

```
------------------- 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_parameter_set_id>
      <frame_num xsi:type="b4">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_parameter_set_id>
      <frame_num xsi:type="b4">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_alignment_zero_bit>
    </rbsp_trailing_bits>
  </skipped_slice_layer_without_partitioning_rbsp>
</coded_slice_of_a_skipped_non_IDR_picture>
```

Figure 3: P slice replaced by a skipped P slice.

case of a single background slice.

If the background slice group also contains B slices, then several changes have to be made to the slice header if one wants to substitute that particular slice with a skipped P slice. First, the slice type (indicated by the syntax element `slice_type`) has to be changed from 1 or 6 (B slice) to 0 (P slice). Changing `slice_type` to 5 (also indicating a P slice) would not be correct since this would imply that all other slices of the current picture are P slices, which is is not always the case since the ROI in the current picture can contain B slices. Next to changing the slice type, B slices contain some slice header syntax elements that cannot appear in the slice header of a P slice. These syntax elements are related to the specific nature of B slices (associated with, for instance, reference picture list L1 or weighted prediction), and need to be removed. To be more specific, the following elements (and the syntax elements that are implied by these syntax elements) are removed by the XSL Transformation in the XML domain:

- `direct_spatial_mv_pred_flag`;
- `num_ref_idx_l1_active_minus1`;
- `ref_pic_list_reordering_flag_l1`;
- `luma_weight_l1_flag` (if applicable);
- `chroma_weight_l1_flag` (if applicable).

The actual slice data is replaced by a serie of skipped macroblocks in the same way as described above. Note that in all cases, the value of the syntax element `slice_qp_delta` can be set to zero to save some aditional bits during the adaptation process. An example of a B slice and a corresponding skipped P slice, as generated by the adaptation process, is given in Figure 4.

```
------------------- 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>
```

Figure 4: B slice replaced by a skipped P slice.

6 RESULTS

In this section, the experimental results of a series of tests are presented. These results give some insight into the implications of the approaches that are described in the previous sections. An overview will be given of the impact on the properties of the bitstreams and on the decoder. Next to this, some performance measurements will be presented regarding the content adaptation process (the BSDL framework). While most of the results demonstrate the benefits of the proposed methods, certain results indicate some

drawbacks and the need for further research.

Four well-known video sequences were used in the experiments: Hall Monitor, News, Stefan, and Crew. The latter sequence has a resolution of 1280×720 pixels whereas the former three have a CIF resolution (352×288). Within these sequences, some ROIs were identified: the moving persons in Hall Monitor and the little bag that is left behind by one of these persons; the heads of the two speakers in News; the tennis player in Stefan; the first two persons of the crew and a separate ROI for the rest of the crew in the sequence Crew. Note that these ROIs have changing positions, as well as sizes in the course of time, and that some ROIs are disappearing at certain points in time.

The sequences were encoded with a modified version of the reference software (based on JM 9.5), once conform to the Baseline Profile and once conform to the Extended Profile (the only difference being the use of B slices). A constant Quantization Parameter (QP) of 28 was used and every slice group contains only one slice. Table 1 summarizes some other properties of the resulting bitstreams, as well as the impact of the adaptation process on the size of the bitstreams. In the table, size_p stands for the size of the bitstream in which all background P and/or B slices were replaced by skipped P slices; size_d stands for the size of the bitstream in which *all* background slices were dropped (leading to bitstreams that are no longer compliant with the H.264/AVC specification – see also Section 4).

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

It is clear from the numbers in Table 1 that the bit rate is reduced significantly when performing the adaptation: up to 64% bit rate reduction by inserting placeholder slices and up to 75% by dropping the background entirely. Of course, this usually has a profound impact on the resulting visual quality. However, in the case of a static background (as for instance in Hall Monitor), the visual quality is almost not affected at all: the resulting average PSNR-Y is 36.7 dB whereas the unadapted version has an average PSNR-Y of 37.7 dB (or 38.0 dB for the Extended Profile). Subjectively, even an expert viewer can scarcely notice that the bitstream has undergone any content adaptation. This also indicates that the encoder could have coded the bitstream much more efficiently if a minor decrease in quality was allowed. This opens up exciting new possibilities for video conferencing applications. Indeed, when the video feeds are to be sent over unreliable networks (as is more and more the case), rather big decreases in available bandwidth can be sustained easily without jeopardizing the visual quality, provided that the network has some Quality of Service (QoS) implemented so that if packets are to be dropped, only the background is affected.

Another important consequence of replacing coded background slices with skipped slices is the impact thereof on the decoding speed of the receiving decoder. Since a considerable amount of macroblocks within a picture are marked as skipped when placeholder slices were inserted into the bitstream, it is expected that this is reflected on the behavior of the decoder because the latter can rely directly on its decoded picture buffer in order to decode the current skipped macroblock without the need for additional computations. To get some more insight, the decoding speed of the reference decoder (JM 10.2) was measured when decoding the various bitstreams (both the original and the adapted bitstreams) several times. The averages of these measurements are summed up in Table 2.

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

sequence		original	placeholders	dropping
IP	crew	1.5	2.0	1.8
	hall monitor	15.6	16.9	17.9
	news	16.7	18.9	18.2
	stefan	10.4	14.9	13.9
IBBP	crew	1.1	1.3	-
	hall monitor	13.8	17.0	17.2
	news	14.3	17.5	17.2
	stefan	9.5	14.4	13.3

As one can see from this table, the decoding speed increases significantly when placeholder slices are inserted into the bitstreams compared to the decoding speed for the original bitstream. The decoding speeds of the decoder, when decoding bitstreams in which the background was dropped, are merely given for completeness. These speeds are comparable to the case of the inserted placeholder slices, but it is important to note that the reference decoder applies a spatial interpolation algorithm to conceal the apparent transmission errors in the background slices by default, rendering a comparison with these numbers unjustified. It should also be noted that, to the best of the authors' knowledge, the reference decoder is the only implementation of the H.264/AVC specification that supports FMO. As a result, measurements for other decoders (having real-time behavior) could not be performed.

Another important observation is that, as mentioned earlier, when replacing coded B slices with skipped P slices, one actually performs 'profile scalability'. Indeed, the bitstreams that are conform to the Extended Profile can be adapted in such way that the resulting bitstreams are conform to the Baseline Profile if *all* coded B slices are replaced by skipped P slices.

The final part of this results section deals with some measurements related to the BSDL framework. It is reported in literature that most performance issues of the BSDL framework are related to the BintoBSD Parser (Devillers et al., 2005; De Schrijver et al., 2006). The results of this paper also lead to the same conclusion. As an illustration, Figure 5 shows the execution speed of the BintoBSD Parser in function of the number of PPSs that are present in a bitstream. All measurements were done on an Intel Pentium 4 2.8 GHz system running a 2.4.19 Linux kernel by making use of the `time` command. The execution speed is given in terms of slices per second as slices are the basic units of parsing for the BintoBSD Parser in the context of this paper.

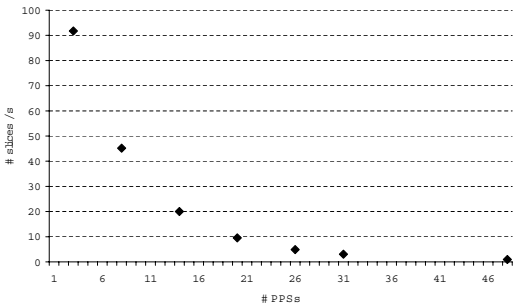

Figure 5: Decreasing execution speed of the BintoBSD Parser.

As one can clearly see from this figure, only when very few PPSs are present in the bitstream, the BintoBSD Parser achieves reasonable execution speeds. The fundamental reason for this, however, lies in the fact that the reference encoder puts all PPSs in the beginning of the bitstream. If the PPSs would appear scattered in the bitstream (at places where the ROI configuration changes), the execution speed would be independent of the number of PPSs, provided that a PPS is never referenced again once another PPS appears in the bitstream. Nevertheless, the fact that the BintoBSD Parser cannot cope with this can be considered a disadvantage.

The transformation of the BSDs (embodying the actual content adaptation) is currently implemented in XSLT. The execution speeds of the transformations barely reach real-time performance, which is caused by the fact that the XSLT engine (Saxon 8) needs the entire DOM tree in memory in order to perform the transformation. The execution times are summarized in Table 3. Therefore, future research will concentrate on the implementation of the transformations by making use of STX. This should dramatically improve the performance.

Table 3: Execution times of XSLT (seconds).

sequence		placeholders	dropping
IP	crew	50.0	48.4
	hall monitor	13.5	13.1
	news	12.9	12.5
	stefan	5.7	5.5
IBBP	crew	49.0	49.2
	hall monitor	13.3	13.3
	news	12.9	12.8
	stefan	5.8	5.4

7 CONCLUSIONS AND FUTURE WORK

In this paper, it is shown how Regions of Interest can be defined within the H.264/AVC specification and how these ROIs can be coded by making use of Flexible Macroblock Ordering. For the adaptation of the bitstreams, the MPEG-21 BSDL framework was applied for the extraction of the ROIs and for the replacement of the coded background slices with placeholder slices (skipped P slices).

By doing so, the bit rate needed to transmit the bitstream can be reduced significantly. Especially in the case of a static camera and a fixed background, this bit rate reduction has very little impact on the visual quality. This opens up interesting new possibilities for certain video applications such as video conferencing where the proposed approach can be seen as the basis for QoS. Another advantage of the adaptation process is the fact that the execution speeds of the receiving decoder fairly increase. Next to this, the concept of placeholder insertion can be used to switch from one Profile to another; for instance going from the Extended Profile to the Baseline Profile.

The XML-driven content adaptation process, as described in this paper, is elegant and generic by design, but the results indicate that there are some performance issues. These shortcomings will be examined in future research aiming to achieve a content adaptation engine that is employable in real-life use cases.

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

Cimprich, P. (2004). Streaming transformations for XML (STX) version 1.0 working draft. http://stx.sourceforge.net/documents/spec-stx-20040701.html.

De Neve, W., De Schrijver, D., Van de Walle, D., Lambert, P., and Van de Walle, R. (2006). Description-based substitution methods for emulating temporal scalability in state-of-the-art video coding formats. In *Proceedings of the 7th International Workshop on Image Analysis for Multimedia Interactive Services*, Korea, accepted for publication.

De Neve, W., Van Deursen, D., De Schrijver, D., De Wolf, K., and Van de Walle, R. (2005). Using bitstream structure descriptions for the exploitation of multi-layered temporal scalability in h.264/avc's base specification. *Lecture Notes in Computer Science, Advances in Multimedia Information Processing, PCM 2005, 6th Pacific-Rim Conference on Multimedia*, pages 641–652.

De Schrijver, D., Poppe, C., Lerouge, S., De Neve, W., and Van de Walle, R. (2006). MPEG-21 bitstream syntax descriptions for scalable video codecs. *Multimedia Systems*, article in press.

Devillers, S., Timmerer, C., Heuer, J., and Hellwagner, H. (2005). Bitstream syntax description-based adaptation in streaming and constrained environments. *IEEE Trans. Multimedia*, 7(3):463–470.

Dhondt, Y., Lambert, P., Notebaert, S., and Van de Walle, R. (2005). Flexible macroblock ordering as a content adaptation tool in H.264/AVC. In *Proceedings of the SPIE/Optics East conference*, Boston.

Ichimura, D., Honda, Y., Sun, H., Lee, M., and Shen, S. (2005). A tool for interactive ROI scalability. JVT-Q020, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q020.doc.

ISO/IEC JTC1/SC29/WG11, . (2005). Applications and requirements for scalable video coding. ISO/IEC JTC1/SC29/WG11 N6880, http://www.chiariglione.org/mpeg/working_documents/mpeg-04/svc/requirements.zip.

Kay, M. (2001). *XSLT Programmer's Reference, 2nd Edition*. Wrox Press Ltd., Birmingham, UK.

Lambert, P., De Neve, W., Dhondt, Y., and Van de Walle, R. (2006). Flexible macroblock ordering in H.264/AVC. *Journal of Visual Communication and Image Representation*, 17(2):358–375.

Li, W. (2001). Overview of fine granularity scalability in MPEG-4 video standard. *IEEE Trans. Circuits Syst. Video Technol.*, 11(3):301–317.

Reichel, J., Schwarz, H., and Wien, M. (2005). Joint scalable video model JSVM-4. JVT-Q202, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q202.zip.

Taubman, D. and Marcellin, M. (2002). *JPEG2000 : Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers.

Thang, T. C., Kim, D., Bae, T. M., Kang, J. W., Ro, Y. M., and Kim, J.-G. (2005). 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.

Vetro, A. and Timmerer, C. (2003). Text of ISO/IEC 21000-7 FCD - part 7: Digital item adaptation. ISO/IEC JTC1/SC29/WG11 N5845, http://www.chiariglione.org/mpeg/working_documents/mpeg-21/dia/dia_fcd.zip.

Wiegand, T., Sullivan, G. J., Bjøntegaard, G., and Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):560–576.

Yin, P., Boyce, J., and Pandit, P. (2005). FMO and ROI scalability. JVT-Q029, http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q029.doc.