Representing Visual Objects in HEVC Coding Loop

Tiejun Huang, *Senior Member, IEEE*, Siwei Dong, and Yonghong Tian, *Senior Member, IEEE*

***Abstract*—Different from the previous video coding standards that employ fixed-size coding blocks (and macroblocks), the latest high efficiency video coding (HEVC) introduces a quadtree structure to represent variable-size coding blocks in the coding loop. The main objective of this study is to investigate a novel way to reuse these variable-size blocks to represent the foreground objects in the picture. Towards this end, this paper proposes three methods, i.e., flagging the blocks lying in the object regions flagging compression blocks (FCB), adding an object tree in each Coding Tree Unit to describe the objects' shape in it additional object tree (AOT) and confining the block splitting procedure to fit the object shape confining by shape (CBS). Among them, FCB and CBS add a flag bit in the syntax description of the block to indicate whether it lies in the objects region, while AOT adds a separate quadtree to represent the objects. For all these methods, the additional bits are then fed into the HEVC entropy coding module to compress. As such, the representation of visual objects in the pictures can be implemented in the HEVC coding loop by reusing the variable-size blocks and entropy coding, without additional coding tools. The experiments on six manually-segmented HEVC testing sequences (three in 1080P and three in 720P) demonstrate the feasibility and effectiveness of our proposal. To represent the objects in the 1080P testing sequences, the BD rate increases of FCB, AOT, and CBS over the HEVC anchor are 1.57%, 3.27%, and 5.93% respectively; while for the 720P conference videos, those are 4.57%, 17.23%, and 26.93% (note that the average bitrate of the anchor is only 1009 kb/s).**

***Index Terms*—Entropy coding, high efficiency video coding (HEVC), variable-size coding blocks, video coding, visual object representation.**

I. Introduction

IN GENERAL, foreground visual objects are regarded as the most important part of a video sequence, especially for vision-related applications. Accordingly, it is crucial to represent foreground objects with arbitrary shape in the coded video stream. For example, for a TV shopping program, audiences are interested in interacting with the specific picture area which contain the promoting commodity; in the video conference applications, the conferees' figures in each camera may be directly extracted and be further blended into an available virtual conference room; while in a surveillance video, the independent representation of moving objects or interested regions will significantly facilitate the subsequent detection and recognition tasks.

In spite of many previous works on visual object representation in the field of computer vision, MPEG-4, the object-based standard, firstly standardized the representation of visual objects in the coded stream [1]. Basically, MPEG-4 represents a scene with dynamic visual (and audio) objects, something like multiple actors performing in a theatre, by encoding the detailed representation of each object and background (e.g., the motion, shape, and color information [2]) as well as the spatial-temporal relationship between different objects. More recently, as the latest video coding standard, high efficiency video coding (HEVC) [3] leads the video coding efficiency to a new milestone, doubling that of MPEG-4 AVC (Part 10)/H.264 [4] and achieving about three times than the MPEG-4 Part 2 advanced simple profile (ASP). Obviously, the quadtree structure in HEVC makes it possible to achieve better representation for visual objects than the fixed-size macroblocks in MPEG-4. However, in its current version, HEVC does not support visual object representation as offered by MPEG-4. Thus, it is necessary to develop a compact object representation in the HEVC coding loop so as to support the vision-related applications based on HEVC.

In this study, we aim to investigate the potentialities of reusing the variable-size coding blocks in HEVC to represent foreground objects with arbitrary shape. Our basic idea is to make the best use of the high efficient coding tools already developed in HEVC. Towards this end, we propose three methods to represent the visual objects.

The first method, which is referred to as flagging compression blocks (FCB) in this paper, flags the HEVC coding blocks on the object regions. Since the blocks are originally generated for the purpose of optimal compression efficiency, this method may not be able to describe the object shape perfectly. On the other hand, FCB does not change the picture encoding procedure, only with the additional bit to indicate whether a block lies on the foreground objects area or not. As a result, the negligible loss in compression performance is induced due to the additional flag bits.

The second method, named as additional object tree (AOT), adds an object tree in each coding tree unit (CTU) that contains at least one foreground pixel so as to describe the objects' shape in it. In this way, the object shape can be well described with variable-size blocks. Like FCB, AOT does not change the HEVC encoding procedure, thus very small loss in compression performance is induced as well due to the usage of additional bits to represent the object tree.

The third, called as confining by shape (CBS) in this paper, is to exert a shape constraint on the block splitting procedure so as to force the blocks on the object boundary to be divided to

Manuscript received September 08, 2013; revised November 24, 2013; accepted December 29, 2013. Date of publication January 27, 2014; date of current version March 07, 2014. This work was supported by the Chinese National Natural Science Foundation under Contract 61035001, Contract 61390515, and Contract 61121002. This paper was recommended by Guest Editor B. Yan.

The authors are with the National Engineering Laboratory for Video Technology, School of Electrical Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: yhtian@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2014.2298274

fit the object shape. Then the objects can be well described with the blocks that are exactly inside of the objects contour. CBS probably decreases the coding efficiency since it constrains the block partitioning. However, one of its potential advantages is that it can satisfy the special requirement in some applications that more bits need to be allocated to the foreground objects (e.g., video conference which often does not care the details in the background).

In the three methods, the additional bits are all fed into the HEVC entropy coding module for compression. That is, the representation of visual objects can be implemented in the HEVC coding loop by reusing the variable-size coding blocks and the entropy coding module without additional new coding tools.

To evaluate the feasibility and effectiveness of our proposal, we performed several experiments on six manually-segmented HEVC testing sequences (three in 1080P and the other three in 720P). Experimental results show that, for representing the objects in the 1080P testing sequences, the BD rate increases of FCB, AOT and CBS over the HEVC anchor are 1.57%, 3.27%, and 5.93%, respectively; while for the 720P conference videos, those are 4.57%, 17.23%, and 26.93% (note that the average bitrate of the anchor is only 1 009 kb/s). Moreover, compared with the so-called ACE method that encodes the binary alpha plane as a supplemental channel to the original video stream [5], the proposed methods can achieve higher precision in object representation. These results validate that the variable-size coding block is a perfect tool to represent the visual objects with arbitrary shape, and the entropy coding is also a powerful tool to compress the representation. Therefore, HEVC can be used as a good platform to support the representation of foreground objects with a very high efficiency.

The rest of this paper is organized as following. The related works are briefly discussed in Section II. Section III presents the three proposed methods. Section IV introduces the experimental video sequences in which the foreground objects are manually segmented. Section V shows the experimental comparison of the three methods. Finally, the paper is concluded in Section VI, with our proposal on extending HEVC to support the object representation.

II. Related Works

MPEG-4 is the first well-known object-based video coding standard. Different from the previous standards such as MPEG-1, MPEG-2, and H.26x, the visual objects (VOs) and their temporal instances, video object planes (VOPs), are the basic components of a MPEG-4 video. In MPEG-4, each video can be considered as a scene consisting of VOPs that may undergo a variety of changes such as translations, rotations, scaling, brightness, and color variations, etc. A VOP can be fully described by texture variations (a set of luminance and chrominance values) and (explicit or implicit) shape representation. For natural scenes, VOPs may be obtained by automatic or semi-automatic segmentation, and the resulting shape information can be represented as a binary shape mask. [6] gave a detailed review on the coding of the VOs with arbitrary shape in MPEG-4. Instead of implicit representation based on chromakey and texture coding, MPEG-4 video employs the explicit shape representation with boundary coding separately from texture coding. To code the corresponding sequence of binary alpha planes, each VO in a VOP sequence is enclosed by a rectangular bounding box with its horizontal and vertical dimensions being multiples of 16 pixels (i.e., the macroblock size). As such, the pixels on the boundaries or inside the object are assigned a value of 255 and are considered opaque, while the pixels outside the object but inside the bounding box are considered transparent and are assigned a value of 0. If a 16×16 block structure is overlaid on the bounding box, three types of binary alpha blocks exist: completely transparent, completely opaque, and partially transparent (or partially opaque). Then coding such a binary block can be performed either lossy or lossless, controlled by a loss threshold which can take values of $0, 16, 32, 64, \ldots, 256$ from lossless to high lossy shape coding. Similar to the coding of texture macroblocks, each binary alpha block can be coded in intra mode (without explicit prediction) or in inter mode (with explicit shape prediction and coding of the resulting binary shape prediction error).

It should be noted that, besides the standardized methods in MPEG-4, some novel or improved arbitrarily-shaped object coding methods were also proposed. For example, the work in [7] presented a new scheme for coding both the shape and texture of an arbitrarily-shaped visual object based on the set partitioning on hierarchical trees (SPIHT). The proposed shape and texture SPIHT (ST-SPIHT) implemented the shape-adaptive discrete wavelet transform (SA-DWT) using in-place lifting, along with the parallel coding of texture coefficients and shape mask pixels to create a single embedded code that allows for fine-grained rate-distortion scalability. Since this paper tries to represent visual objects in the HEVC coding loop without additionally introducing any new tools, no more papers on independent visual object representation is to be reviewed here.

Besides MPEG-4, all previous ITU-T and ISO/IEC JTC 1 video coding standards make use of fixed-size 16×16 mac-blocks as the basic coding units. This configuration may work well for compression, but places a major restriction on directly using the blocks to describe arbitrary shape. Obviously, the fixed-size block is too coarse to accurately fit the boundary of an arbitrary shape. On the other hand, multiple 16×16 blocks are needed to cover a larger area that can be directly covered by a 32×32 or 64×64 block.

The latest video compression standard, HEVC, introduces variable-size coding blocks as the core of the coding structure [4]. In HEVC, each picture is partitioned into slices, tiles, and coding tree units (CTUs). CTU in HEVC is analogous to the macroblock for the previous video coding standards. The subdivision of CTU results in coding units (CUs) while each CU is split into prediction units (PUs) and transform units (TUs). These variable-size block structures in HEVC can be easily used to represent the shape of objects. That is, the HEVC blocks with the maximum size of 64×64 can provide an efficient way to cover a large object, and at the same time the variable-size block with the minimum size of 4×4 can be reused to well fit the fine boundary. Therefore, this paper will investigate the possibility to describe the objects with arbitrary shape by taking advantage of the HEVC variable-size blocks.

In general, the quadtree is a common region-based method to represent the shape of a visual object. Naturally, it is very

straight to employ the coding tree structure of HEVC to represent foreground objects in the pictures. To do so, the high efficient tools already developed in HEVC could be utilized to compress the additional bits consumed by the object representation, without new hardware or software modules. In other word, it is possible to spend *fewer bits* to represent visual objects in the HEVC coding loop.

III. The Proposed Methods

In this study, our basic idea is to utilize the variable-size coding blocks in HEVC to represent foreground objects with arbitrary shape, with the slightest extension to HEVC (if necessary). With this in mind, we will investigate different potential object representation methods in this section.

In HEVC, a picture is firstly divided into $L \times L$ coding tree blocks (CTBs) where L could be 64, 32, or 16 (64 is adopted in this paper). Then, each CTB could be recursively split into squared coding blocks (CBs) until the size for a CB reaches the minimal CB size which is usually 8×8. Each CB could be partitioned again into a set of prediction blocks (PBs) and a set of transform blocks (TBs). The size of PBs in an $M \times M$ CB could be $M \times N$, where$N \leq M$. The TBs within a CB are also organized as a quadtree, and the minimal TB size is 4×4. Therefore, to find a partition of a picture for object representation, there are four types of blocks available in HEVC (see Fig. 1).

1) CTB: Typically, CTB is the largest CB. It is too large to describe the details of the object boundary. But for the application that does not care the representation precision of the object shape so much, CTB is a good choice since flagging CTBs needs very few bits.
2) CB: CB can be as small as 8×8 in size. Despite not so accurate, the representation granularity is fine enough for most applications. As shown in Fig. 1, all CBs are organized as a quadtree in a CTU.
3) TB: TB is a lower level to CB. TBs in one CB are organized as a quadtree. The smallest TB is with 4×4. Thus, the TB-based representation can reach to the 4×4 precision.
4) PB: PB is another lower level partition of CB. A CB could be divided into multiple PBs in non-square shape, for example, a 16×4 PB and 16×12 PB in a 16×16 CB. In some case, the nonsquared PB spends even fewer bits than multiple squared TBs.

With the above four types of blocks, there are five combinations to form a complete partition of a picture: CTBs only, CTBs+CBs, CTBs+CBs+PBs, CTBs+CBs+TBs, and CTBs+ CBs+PBs+TBs.

As both PBs and TBs can construct a complete partition of the CB that contains them, it is possible to employ them together to get a better partition for object representation. However, this combination will lead to a complicated foreground black flagging mechanism with a high bit cost. Thus this paper does not go further on this direction.

The smallest size of the "block" in HEVC is 4×4, which is also the finest granularity in HEVC to represent the objects in a picture with the variable-size blocks. To reach higher accuracy such as 2×2 or only 1 pixel, additional tools are required. In the shape coder of MPEG-4, for example, there is a specific tool to describe the arbitrary boundary in a 16×16 macroblock.

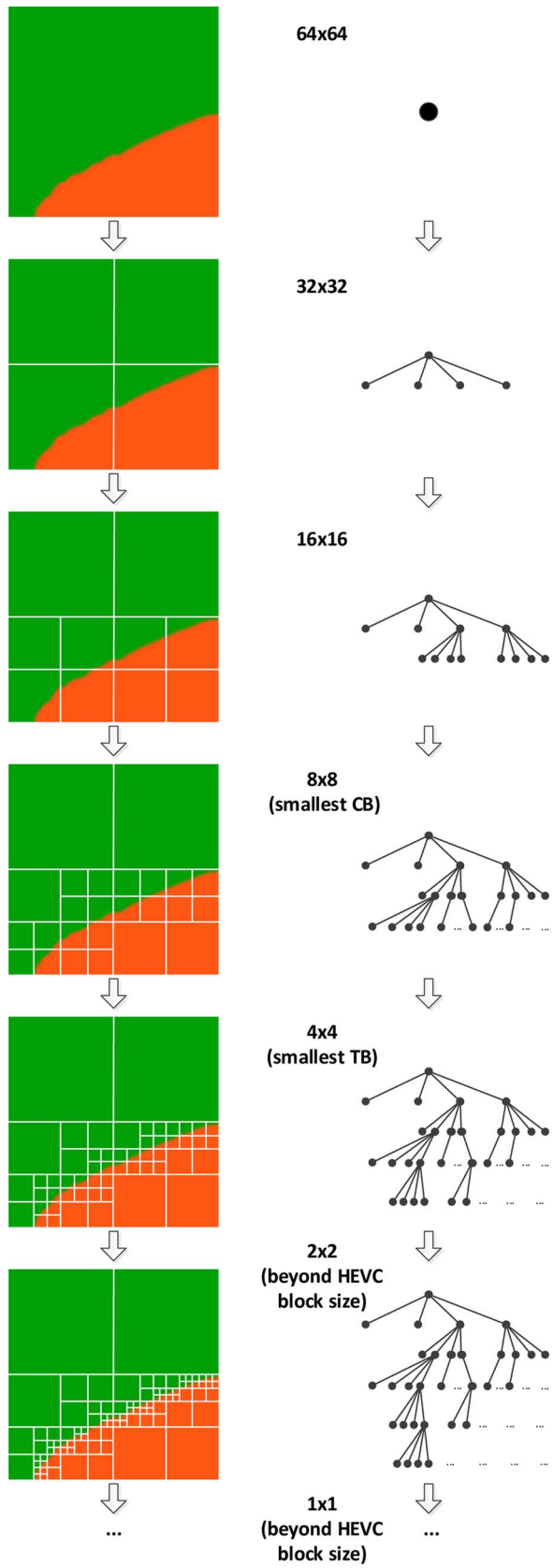

Fig. 1. Partition of a CTB with CBs, TBs, and smaller blocks.

Another approach is to employ the HEVC quadtree again to describe the subtle structure till to pixel, which is easier to be

implemented in the HEVC framework. Fig. 1 demonstrates the possibilities to partition a CTB at different levels as mentioned above (where PB is not demonstrated).

To simplify the discussion, this paper takes the "HEVC block" or "block" as the general term referring to CTB, CB, PB, or TB. In the following description, the CB is employed as the basic blocks. However, the presented methods can also be used in other precision levels.

A. Method I: Flagging Compression Blocks

From the partitioning prospective, all coding blocks (CBs) in a picture make up a complete partition of that picture (like a jigsaw with variable-size pieces). Even the partition is originally generated to achieve higher compression efficiency, it can be reused to describe the foreground objects. More specifically, the objects could be represented by simply flagging the blocks overlaying the object regions. Therefore, this method is named as flagging compression blocks (FCB). One possible drawback of FCB is the foreground objects could not be represented accurately because their boundary may not align with that of the blocks.

Given a partition, the representation of an object means to flag each block whether it lies inside an object (or more than half of the block is overlaid with the object). In HEVC, this can be done by adding a flag of one-bit descriptor in the syntax of the description unit of the corresponding block (e.g., a CU for its CB). Except this extension to the HEVC syntax, no more change is introduced. In other word, no additional *coding tool* is added into the HEVC coding loop.

In general, the CB at the leaf node of the coding quadtree (referred to as the leaf CB), which does not need to be further split, is considered as a *foreground block* only if more than half of its pixels are foreground pixels; otherwise, it is a *background block*. Recursively, a nonleaf CB is defined as a foreground block if it contains at least one leaf CB that is a foreground block; otherwise, it is a background block. Moreover, if a CB or CTB does not contain any background pixel, it is recognized as a *full* background block.

One way to indicate a CB is a foreground or background block is to insert a flag named as "foreground_flag" into the syntax of its CU. A block is flagged as "1" to indicate it is a foreground block; otherwise, it is flagged as "0" to indicate a background block. A more efficient way is to add the flag on the middle nodes of the quadtree. That is, if the corresponding CTB is a full background block, then the foreground_flag should be set to 0, and all of its descendant blocks (if they exist) do not add the foreground_flag; otherwise, the foreground_flag should be set to 1, and all of its descendant blocks (if they exist) should add the foreground_flag after their split_cu_flag. This procedure will be repeated to the leaf nodes or a full background block.

Table I is the extension of the HEVC coding tree syntax in which the flag descriptor is placed just after the "split_cu_flag". If a split_cu_flag is "0" (i.e., no further splitting), the following foreground_flag indicates the corresponding CB as a foreground ("1") or background ("0") block. For the case of split_cu_flag being "1," if the corresponding CB is a background block, a foreground_flag "0" is followed and no foreground_flag is added for all of the descendant blocks; otherwise, a foreground_flag "1" is followed and all its descendant blocks still need the foreground_flag.

TABLE I
ADDING A FOREGROUND FLAG DESCRIPTOR IN THE HEVC CODING TREE SYNTAX (FOR FCB METHOD)

(a)

coding_tree_unit(xCtb, yCtb) {	Descriptor
…	
coding_quadtree(xCtb, yCtb, Log2CtbSizeY, 0, 1)	
}	

(b)

coding_quadtree(x0,y0, log2CbSize,ctDepth,**parent_foreground_flag**){	Descriptor
if(x0+(1<<log2CbSize)<=pic_width_in_luma_samples && y0+(1<<log2CbSize)<=pic_height_in_luma_samples && log2CbSize>MinCbLog2SizeY)	
split_cu_flag[x0][y0]	ae(v)
if(ctDepth== 0 || **parent_foreground_flag**==1){	
foreground_flag[x0][y0]	ae(v)
parent_foreground_flag=foreground_flag[x0][y0]	
}else{	
parent_foreground_flag=0	
}	
…	
if(split_cu_flag[x0][y0]){	
x1 = x0 + (1 << (log2CbSize − 1))	
y1 = y0 + (1 << (log2CbSize − 1))	
coding_quadtree(x0,y0,log2CbSize-1, ctDepth+1,**parent_foreground_flag**)	
if(x1<pic_width_in_luma_samples)	
coding_quadtree(x1,y0,log2CbSize-1, ctDepth+1,**parent_foreground_flag**)	
if(y1<pic_height_in_luma_samples)	
coding_quadtree(x0,y1,log2CbSize-1, ctDepth+1,**parent_foreground_flag**)	
if(x1<pic_width_in_luma_samples && y1<pic_width_in_luma_samples)	
coding_quadtree(x1,y1,log2CbSize-1, ctDepth+1,**parent_foreground_flag**)	
}else{	
…	
}	
…	
}	

At the decoding side, the object region could be reconstructed by collecting the blocks whose foreground_flag bit equals to "1."

Since the FCB method just adds a flag bit in the CTU syntax and no change is made to the picture encoding procedure, the loss in compression performance induced by this method is negligible. The drawback of FCB is that the coding tree originally optimized for high efficient compression may not be perfect to fit with the object shape. That is, the blocks lying on the object boundary may be too coarse, consequently limiting the precision for object representation.

B. Method II: Additional Object Tree

For better representation, an improved way is to describe the shape with a quadtree just as HEVC does in organizing CBs in a CTU. The second method, named as additional object tree

(AOT), adds an independent object tree in each CTU to describe the objects' shape in it. Like FCB, the AOT method does not change the HEVC encoding procedure, thus very small loss in compression performance is induced as well.

In AOT, all CTBs which contain at least one foreground pixel should be split till to each block in the CTB is either a foreground block or background block. Here, for the smallest block, an 8×8 CB, if more than half pixels in it are foreground pixel, it is a foreground block, otherwise a background block.

The procedure to produce the quadtree for a CTB is as following: *Start from the CTB, the splitting process will be repeated until each leaf node either reaches the smallest size or becomes a full foreground or full background block.* More formally, let $O_{i,j}^{(d)}$ denote the object block at position (i, j) with the splitting depth d ($0 \le d \le 3$ in the experiments, corresponding to the smallest size of CBs) and the size of $2^{6-d} \times 2^{6-d}$, then the corresponding object tree $\boldsymbol{\mathcal{T}}(i, j, d)$ can be generated in a recursive manner

$$\boldsymbol{\mathcal{T}}(i,j,d) = \begin{cases} \bigcup_{\substack{s\in\{i,i+2^{5-d}\}\\ t\in\{j,j+2^{5-d}\}}} \boldsymbol{\mathcal{T}}(s,t,d+1), & d \neq 3 \text{ and } 0 < f(O_{i,j}^{d}) < 1 \\ \{O_{i,j}^{d}\}, & d = 3 \text{ or } f(O_{i,j}^{d}) = 0 \text{ or } 1 \end{cases} \tag{1}$$

where $f(O_{i,j}^{d})$ is the percentage of the foreground pixels in $O_{i,j}^{(d)}$, and d is initialized to 0.

To simplify the procedure of judging whether a CB is a full foreground or a full background block in each step, Fig. 2 shows an alternative way: A picture is firstly partitioned to an array of the smallest CBs (here is a 8×8 block), where each CB is assigned to a foreground or background flag based on whether the foreground pixels in it account for more than half of the pixels. Then each CTB (an array of 64 smallest blocks) is recursively split into squared CBs (larger than 8×8) until all these descendent CBs are full of foreground or background flags. For example, as shown at the top of Fig. 2, if a CTB contains foreground pixels (the foreground proportion is between 0% and 100%), it should be split into four 32×32 CBs. Each 32×32 CB is then split into four 16×16 CBs based on its foreground proportion, so does each 16×16 CB. Once the size of a CB is 8×8 or it is full of foreground/background flags, the split procedure terminates.

Table II shows the syntax of the object quadtree whose root is also associated with the corresponding CTU, while the function of the object tree generation is given in Table III.

Like as FCB, AOT has little influence on the coding performance loss. However, AOT may use more bits to approximate the object shape.

C. Method III: Confining by Shape

The advantages of both FCB and AOT are: 1) only negligible loss in compression performance is induced, and 2) the HEVC encoding procedure is not changed. However, the precision of object representation in FCB is confined by the coding tree that is generated for the purpose of optimal compression efficiency, while AOT may cost more bits to represent the independent object tree.

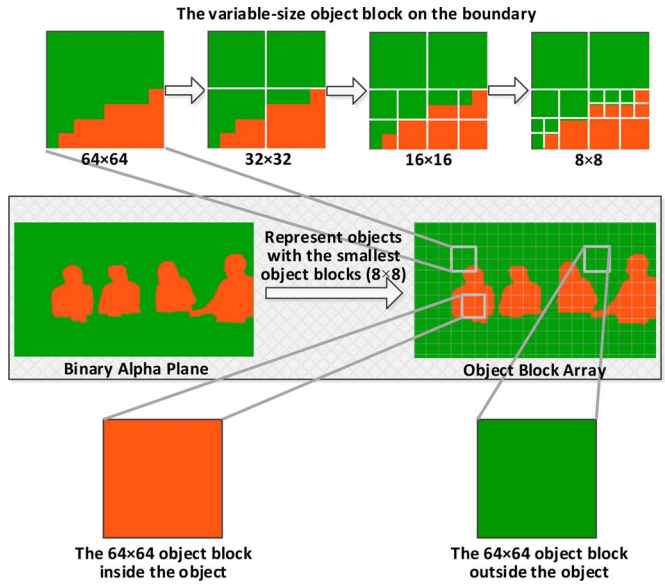

Fig. 2. Quadtree partition of a CTB overlaying visual objects.

TABLE II
SYNTAX FOR OBJECT TREE UNIT

coding_tree_unit(xCtb, yCtb) {	Descriptor
…	
coding_quadtree(xCtb, yCtb, Log2CtbSizeY, 0)	
ou_quadtree(xCtb, yCtb, Log2CtbSizeY, 0)	
}	

TABLE III
FUNCTION FOR OBJECT QUADTREE GENERATION

ou_quadtree(x0, y0, log2ObSize, otDepth) {	Descriptor
if(x0+(1<<log2ObSize)<=pic_width_in_luma_samples && y0+(1<<log2ObSize)<=pic_height_in_luma_samples && log2ObSize>MinCbLog2SizeY)	
split_ou_flag[x0][y0]	ae(v)
if(split_ou_flag[x0][y0]){	
x1 = x0 + (1 << (log2CbSize − 1))	
y1 = y0 + (1 << (log2CbSize − 1))	
ou_quadtree(x0, y0, log2ObSize-1, otDepth+1)	
if(x1 < pic_width_in_luma_samples)	
ou_quadtree(x1, y0, log2ObSize-1, otDepth+1)	
if(y1<pic_height_in_luma_samples)	
ou_quadtree(x0, y1, log2ObSize-1, otDepth+1)	
if(x1<pic_width_in_luma_sample && y1<pic_height_in_luma_samples)	
ou_quadtree(x1,y1,log2ObSize-1,ctDepth+1)	
}else{	
foreground_ou_flag[x0][y0]	ae(v)
}	
}	

In HEVC encoder, when coding a 64×64 CTB region, it firstly determines whether this block should be encoded as a whole 64×64 CB or recursively encoded in the form of four individual parts until the minimal size of CB (i.e., 8×8) is reached. In this block splitting procedure, the encoder usually makes the decision by recursively calculating the RD cost of

each kind of partitions of the input region. Here a shape constraint on the block splitting procedure can be exerted to force dividing the blocks on the objects' boundary so as to fit the object shape. This is the main idea of the third method, which is referred to as confining by shape (CBS) in this paper.

In general, there are two ways to implement the encoding procedure constrained by the object shape. One way is to firstly produce an object tree for each CTB containing foreground pixels, just like AOT does. This initializes a partition of the picture. Starting from this initial partition, the HEVC block splitting procedure is carried out to minimize the RD cost. Since the initial partition is generated to well describe the objects, the further partitioning performed by HEVC can be perfect for both object representation and picture compression.

The other way is to tune the HEVC block splitting procedure directly. A picture is firstly partitioned by the normal HEVC encoder to achieve a minimum RD cost. Then each CB in the partition is double-checked whether it lies on the object boundary. If it is true, the CB should be further split till to all its descendants are pure foreground or background. As a result, a coding tree will be generated by optimizing compression efficiency and achieving the best representation of object shape.

Obviously, the final partition of a picture is the same in the two ways. It is a superset of the partition result of FCB (for compression purpose) and that of AOT (for shape description purpose). Therefore, the object representation in CBS is also the same to AOT, namely, adding an object sub-tree in the HEVC coding tree.

However, CBS needs to pay the cost for the additional object sub-tree and may induce the possible loss for the block partitioning confined by the object shape. Despite of this, since the blocks composing the objects are identified explicitly by the object sub-tree, it is possible to allocate more bits to these blocks so as to satisfy the application requirements that the foreground objects should be emphasized. For example, the person region is more important than the background for video conference, especially when bandwidth is limited.

D. Discussion

To represent foreground objects with arbitrary shape in the HEVC coding loop, one of the most important prerequisites is how to obtain the object shape mask or binary alpha plane for a video frame. In this study, the ground truth is manually segmented such that we can accurately compare the three methods in the experiments. So, one may question us because accurate foreground segmentation is still a difficult problem in the field of computer vision. However, we have been evidenced that despite of not being applied in the general domain, automatic foreground segmentation is highly applicable in many specific applications. For example, the moving persons or vehicles in a surveillance video could be segmented by the background modeling and subtraction algorithms. One of such examples is shown in Fig. 3, in which foreground objects are segmented by a low-complexity background modeling method [8]. Despite still not very perfect, the segmented objects are good enough for detection and recognition tasks. In this case, the FCB method can be used for object representation since the nonperfect representation is probably acceptable to the tasks.

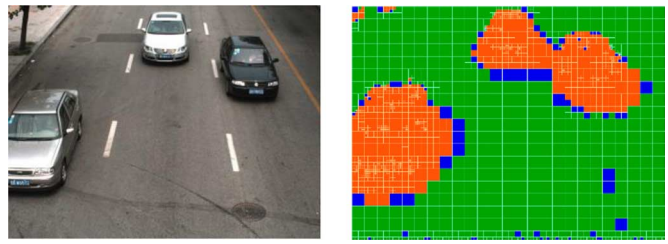

Fig. 3. Example of foreground objects segmented by a low-complexity background modeling method [8], where red grids indicate the objects (small red grids means texture details), green ones are background, while blue ones are ambiguous.

TABLE IV
SIX HEVC TESTING VIDEOS USED IN OUR EXPERIMENTS

Resolution	Sequence name	Frame count	Frame rate (fps)	Foreground feature (person count, size)
1080P (Class A)	Kimono	240	24	1, large
	ParkScene	240	24	2, small to middle
	BasketballDrive	500	50	multiple, variable sizes
720P (Class E)	Johnny	600	60	1, large
	KristenAndSara	600	60	2, large
	FourPeople	600	60	4, middle

However, the requirement of conference videos is different from surveillance videos. To blend a conferee's figure into a virtual environment, his contour should be very accurate to avoid the artifacts on the boundary. In this case, as the pure background is often available, the foreground figures could be accurately segmented by automatic algorithm. Thus, the AOT method is a good choice to represent the contour with a high precision (for virtual conference synthesizing purpose). Although AOT needs a larger additional bitrate cost than FCB, it is still practicable since the coding efficiency of HEVC on conference videos is very high.

IV. TEST SEQUENCES

In the HEVC call for proposal [8], there are totally 15 video sequences in five classes (corresponding to five typical resolutions: 2560×1600, 1080P, WVGA, WQVGA, and 720P). Among them, three 720P conference videos (Class E) were replaced with three new ones in 2012 [10].

To generate the ground truth for this study, we need to segment the foreground objects manually from the videos. So we select the 1080P and 720P sequences, which are normally used in practice, as the representative testing videos in our experiments. The selected sequences include three of the five 1080P sequences in Class A which contain one or more persons, and three 720P conference videos containing 1–4 persons. Table IV shows the detailed information of all these sequences, including the sequence name, frame count, frame rate, and foreground features. To conduct our experiments, all the foreground persons in each sequence are manually segmented[1].

[1]The binary descriptions for each sequence used in our experiments is publically available at http://mlg.idm.pku.edu.cn/resources/dataset.html.

Fig. 4. Key frames and the shape masks of the foreground objects in the six experimental video sequences.

For better visualization purpose, one key frame and the corresponding segmented shape of the foreground objects in each sequence are shown in Fig. 4. We can see that these sequences used in our experiments can cover a wide spectrum of video scenes in real-world scenarios.

- In terms of scene types, two sequences were originally collected from outdoor movies (i.e., Kimono and ParkScene), one was from sports videos (i.e., BasketballDrive), and the remaining three were from conference videos.
- In terms of scene complexity, two sequences contain relatively more complex scenes (i.e., BasketballDrive and FourPeople), each with four or more persons; while the other four correspond to simple scenes.
- In terms of foreground object sizes, three sequences contain persons of large size (i.e., Kimono, Johnny, and KristenAndSara), one contains several persons of middle size (i.e., FourPeople), and the remaining two contain persons of variable size.

Therefore, these selected videos are fairly representative, and consequently the experimental results on these videos can be well generalized to other cases.

V. Experimental Results

In this study, four experiments are designed on the sequences to investigate the potentialities of utilizing the variable-size blocks in HEVC to represent foreground objects.

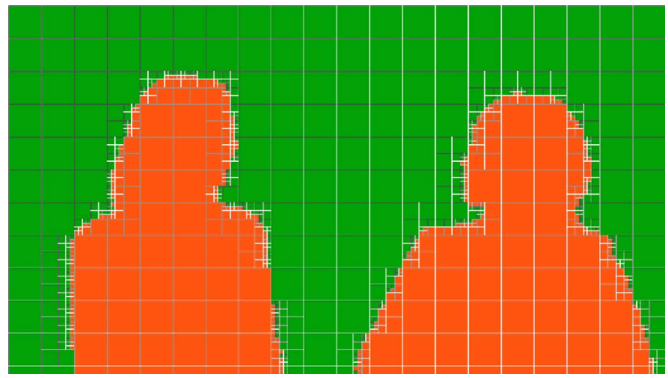

Fig. 5. Partition of a picture with CTBs and CBs.

- The first experiment is to evaluate the overall cost for object representation using the three methods, in terms of the total bitrate increase and Bjontegaard distortion rate (BD rate) increase on each sequence.
- In the second experiment, we aim to experimentally analyze the potential correlation between the object representation cost and the foreground proportion.
- In the third experiment, three methods are compared in terms of the precision of object representation.
- Finally, we compare the proposed methods with a straightforward way to represent foreground objects, namely the ACE (Alpha Channel Encoding) method that encodes the binary alpha plane utilizing the HEVC coding tools as a supplemental channel to the original video stream [5].

In our experiments, HEVC test model HM12.0 with random-access configuration [11] is used as the basic experimental platform and the anchor. To compare the three methods FCB, AOT, and CBS in the same common conditions, this paper adopts the CTBs+CBs combination to form a complete partition of a picture in the experiments. The object shapes shown in Fig. 4 are all partitioned by CTBs and CBs. As an example, Fig. 5 shows a clearer version of the partition of a picture from the "KristenAndSara" sequence.

A. Overall Bitrate Cost and BD-Rate Increase

Experimental results in Table V shows the overall object representation cost of the three proposed methods. It is easy to notice that, although the absolute bitrate increase in Class E is much lower than that in class A, the relative percentage of BD rate in Class E is much larger than in class A. This is mainly due to the HEVC's high coding efficiency for conference videos. As we can see from Table V, the average coding bitrate of the HEVC anchor is as small as 1009.17 kb/s for the three 720p conference videos. As a result, the corresponding BD rate increase reaches a relatively large value.

Also as shown in Table V, FCB, AOT, and CBS averagely utilizes 1.57%, 3.27%, and 5.93% of the total bitrate to represent foreground objects for sequences in class A, whereas employing 4.57%, 17.23%, and 26.93% for sequences in Class E. Among them, FCB needs the lowest additional bitrate, while CBS pays the highest cost for object representation. Compared with CBS, AOT needs a reasonable bitrate increase. These results show that in CBS, the additional shape constraint will tend to select

TABLE V
OVERALL OBJECT REPRESENTATION COSTS IN THE THREE METHODS

Video	Anchor	FCB vs Anchor (kbps)		AOT vs Anchor (kbps)		CBS vs Anchor (kbps)	
	Bitrate	Bitrate increase	BD rate increase	bitrate increase	BD rate increase	bitrate increase	BD rate increase
Kimono	2190.65	19.84	1.40%	38.89	3.10%	61.65	5.30%
ParkScene	3184.74	15.15	0.80%	26.61	1.50%	37.11	2.20%
BasketballDrive	7359.27	106.81	2.50%	190.90	5.20%	335.07	10.30%
Average results for Class A	**4244.887**	**47.27**	**1.57%**	**85.47**	**3.27%**	**144.61**	**5.93%**
Johnny	761.77	27.20	5.70%	79.14	19.20%	120.02	30.00%
KristenAndSara	989.05	31.03	4.50%	103.85	17.80%	162.05	28.30%
FourPeople	1276.68	33.40	3.50%	118.67	14.70%	178.92	22.50%
Average results for Class E	**1009.17**	**30.54**	**4.57%**	**100.55**	**17.23%**	**153.66**	**26.93%**

TABLE VI
DETAILS ON THE CODING BITRATES UNDER DIFFERENT QPS (ABSOLUTE BITRATE INCREASE (KB/S) AS WELL AS BD RATE INCREASE)

Video	QP	Anchor Bitrate	FCB vs Anchor (%)	AOT vs Anchor (%)	CBS vs Anchor (%)
Kimono	22	4895.96	26.66 (0.5%)	40.64 (0.8%)	64.65 (1.3%)
	27	2237.46	21.40 (1.0%)	39.20 (1.8%)	60.73 (2.7%)
	32	1086.65	17.18 (1.6%)	38.19 (3.5%)	61.29 (5.6%)
	37	542.52	14.13 (2.6%)	37.51 (6.9%)	59.95 (11.1%)
	BD rate increase		**1.40%**	**3.10%**	**5.30%**
ParkScene	22	7390.40	20.10 (0.3%)	28.80 (0.4%)	38.05 (0.5%)
	27	3201.25	17.10 (0.5%)	27.25 (0.9%)	36.75 (1.1%)
	32	1466.56	13.41 (0.9%)	25.85 (1.8%)	36.66 (2.5%)
	37	680.75	10.01 (1.5%)	24.53 (3.6%)	36.99 (5.4%)
	BD rate increase		**0.80%**	**1.50%**	**2.20%**
BasketballDrive	22	18236.98	156.24(0.9%)	195.26(1.1%)	344.11 (1.9%)
	27	6558.14	116.91 (1.8%)	191.04(2.9%)	337.35 (5.1%)
	32	3051.60	87.58 (2.9%)	189.15(6.2%)	329.48 (10.8%)
	37	1590.34	66.53 (4.2%)	188.16(11.8%)	329.34(20.7%)
	BD rate increase		**2.50%**	**5.20%**	**10.30%**
Johnny	22	1778.14	38.57 (2.2%)	79.79 (4.5%)	120.25 (6.8%)
	27	699.96	27.63 (3.9%)	79.02(11.3%)	119.49 (17.1%)
	32	363.25	22.76 (6.3%)	78.86(21.7%)	119.89 (33.0%)
	37	205.73	19.82 (9.6%)	78.88(38.3%)	120.46 (58.6%)
	BD rate increase		**5.70%**	**19.20%**	**30.00%**
KristenAndSara	22	2202.90	45.42 (2.1%)	104.81 (4.8%)	174.18 (7.9%)
	27	955.95	31.97 (3.3%)	103.99(10.9%)	163.31(17.1%)
	32	508.05	25.30 (5.0%)	103.37(20.3%)	156.87(30.9%)
	37	289.29	21.42(7.4%)	103.25(35.7%)	153.83(53.2%)
	BD rate increase		**4.50%**	**17.80%**	**28.30%**
FourPeople	22	2702.70	49.01(1.8%)	119.67(4.4%)	185.36 (6.9%)
	27	1283.94	35.32(2.8%)	118.52 (9.2%)	178.22(13.9%)
	32	710.85	27.56(3.9%)	118.27(16.6%)	177.23(24.9%)
	37	409.24	21.72(5.3%)	118.23(28.9%)	174.88(42.7%)
	BD rate increase		**3.50%**	**14.70%**	**22.50%**

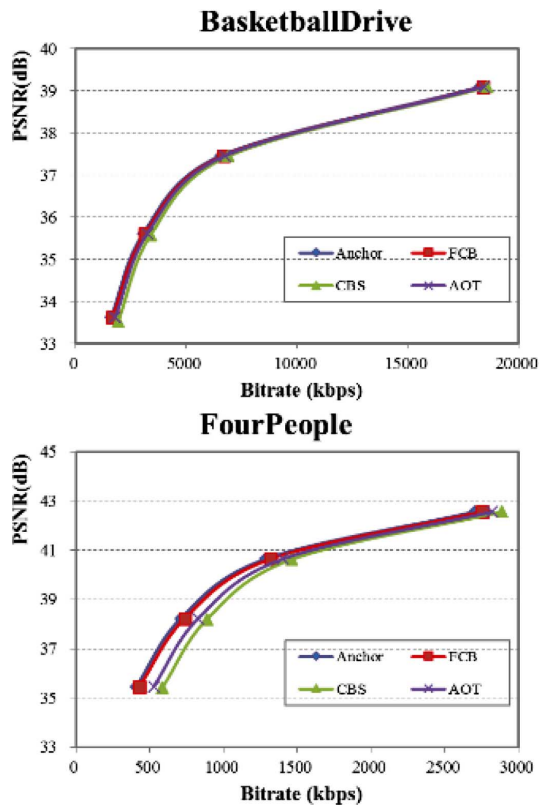

Fig. 6. Rate-distortion curves for BasketballDrive and FourPeople.

a block partition that does not have the minimal RD cost, inevitably leading to a slight increase of BD rate (2.66% for 1080P and 9.7% for conference videos). Despite of this, the additional bitrate paid for object representation by the three methods keeps very low and is at the acceptable level.

Table VI further shows the details on the coding bitrates under different QPs. Similarly, for all the three methods, the higher the QP is used, the larger the relative percentage of bitrate increase is paid for object representation, despite the absolute bitrate increase becomes smaller. This is reasonable because the HEVC encoder can employ much fewer bits to code a picture under a higher QP, but the bits for object representation cannot be reduced significantly.

Combining the results shown in Tables V and VI, we can also find that for all the three methods, the additional bitrate increases on sequences with more complex scenes (e.g., BasketballDrive and FourPeople) are significantly larger than those on the other sequences. As two examples, Fig. 6 shows the rate-distortion curves for BasketballDrive and FourPeople. This finding motivates us to further investigate the potential correlation between the overall object representation cost and the foreground proportion in the next experiment.

B. In-Depth Analysis According to Foreground Proportions

We have seen that the bitrate increase varies between Class A and Class E, and also is very different for diverse sequences within each class. For example, FCB averagely utilizes 106.81 kb/s for object representation in BasketballDrive and 19.84 kb/s in Kimono; similarly, 190.90 kb/s and 38.89 kb/s for AOT, and 335.07 kb/s and 61.65 kb/s for CBS. That is, for all the methods,

TABLE VII
BD RATE INCREASE AND THE AVERAGE FOREGROUND PROPORTION

Video	FCB	AOT	CBS	Foreground proportion
Kimono	1.40%	3.10%	5.30%	**13.26%**
ParkScene	0.80%	1.50%	2.20%	**2.63%**
BasketballDrive	2.50%	5.20%	10.30%	**17.06%**
Johnny	5.70%	19.20%	30.00%	**31.64%**
KristenAndSara	4.50%	17.80%	28.30%	**40.30%**
FourPeople	3.50%	14.70%	22.50%	**23.57%**

there is a large bitrate gap between two sequences in a class. Intuitively, this is because these sequences have different proportions of foreground and background pixels, consequently leading to the difference in the bitrate increase of object representation. To validate this conjecture, we further conduct an experiment to check whether there is a correlation between the foreground proportion and the object representation cost for each method.

Table VII summarizes the BD rate increase and the average foreground proportion for each sequence. Clearly, the results totally support our conjecture that the larger foreground proportion a sequence has, the higher object representation cost each method will pay. One exception is that for all the three methods, the BD rate increase on the sequence Johnny (with 31.64% foreground proportion) is larger than that on the sequence KristenAndSara (with 40.30% foreground proportion). This is mainly due to the very low bitrate that is used for the sequence Johnny by the HEVC encoder.

Fig. 7 plots the foreground proportions *every* second (totally 10 s for better visualization) for all sequences and the corresponding bitrate increases for all methods. From Fig. 7, we can further find that there is a strong correlation between the foreground proportion and the corresponding bitrate increase for each method. This indicates that the bitrate increase, on all these sequences, varies strictly following the changes of the foreground proportion every second. Despite very intuitive, this experimental observation can enlighten us to take the foreground proportion into account as an important variable when jointly optimizing the compression distortion and the representation precision.

C. Comparison of Representation Precision

From Table V, we have seen that FCB needs the lowest additional bitrate, AOT needs more bits than FCB, while CBS pays the most cost for object representation. That is, let $R(x)$ be the rate cost of method x, we have $R(\mathrm{FCB}) < R(AOT) < R(\mathrm{CBS})$. However, here we cannot conclude that FCB is the best choice for object representation, since the bitrate cost is just one side of a coin. The precision of object representation is also an important side for each method. Thus in the last experiment, we further compare three methods in terms of the object representation precision.

In the field of image segmentation and foreground subtraction, there are many metrics to evaluate the quality of the segmentation results with respect to the ground-truth. Following the segmentation task in the well-known PASCAL VOC challenge [12], we adopt a scale-invariant metric, *overlap*, to measure the precision of the object representation with different methods in our experiments. For a give frame, let G be the set of the ground truth, S be the set of pixels which are marked by the object representation method as foreground, then the overlap can be calculated by

$$\mathrm{Overlap}(S,G) = \frac{|G \cap S|}{|G \cup S|} \tag{2}$$

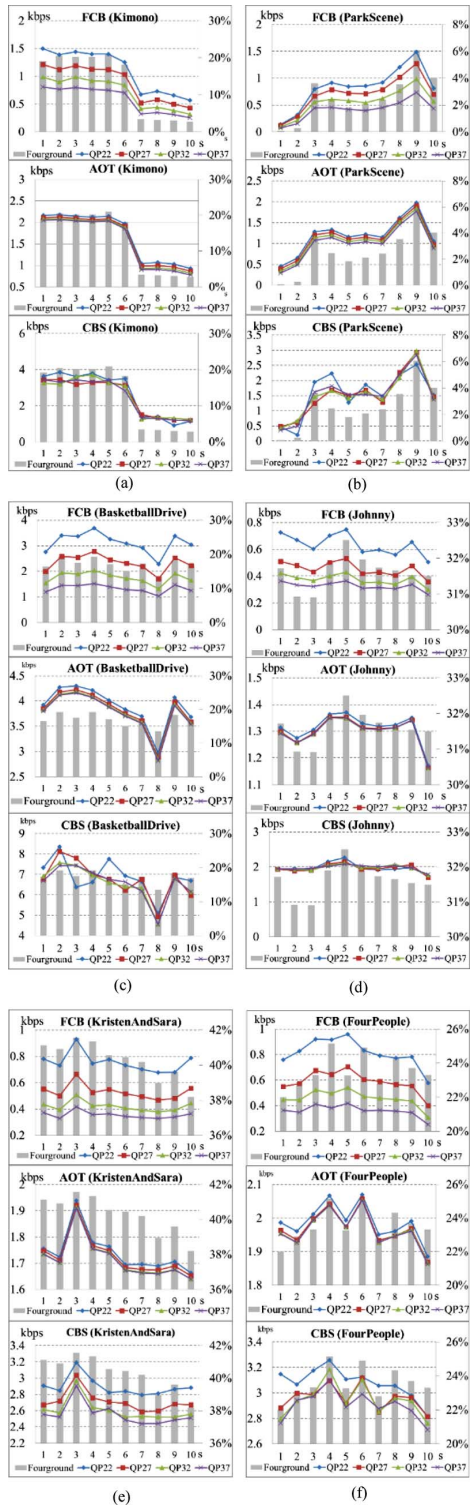

Fig. 7. Foreground proportion and bitrate increase *every* second. (a) Kimono. (b) ParkScene. (c) BasketballDrive. (d) Johnny. (e) KristenAndSara. (f) FourPeople.

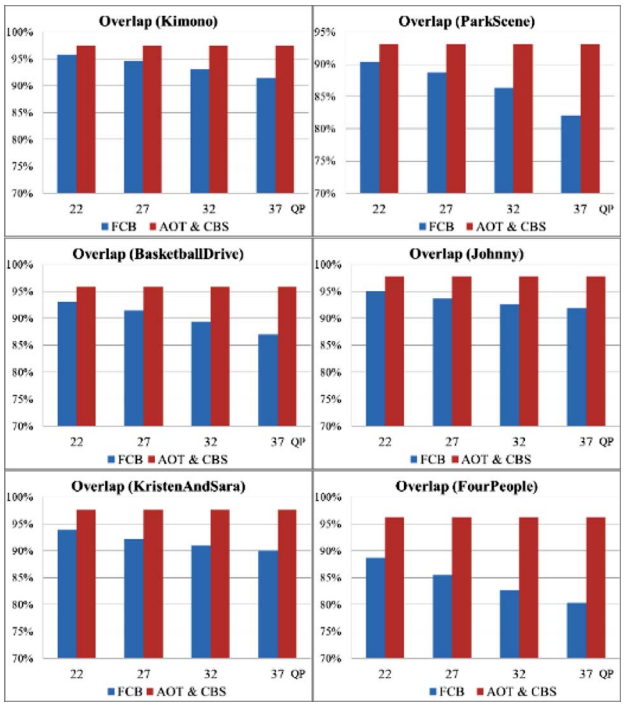

Fig. 8. Histograms of average overlaps for different methods on each sequence under different QPs.

where $|X|$ denotes the number of elements in the set X. Then we can calculate the representation precision for each method on a given sequence by averaging the overlaps on all frames in that sequence.

Following this, we further conduct an experiment to evaluate the precision of object representation for each method. Fig. 8 shows the histograms of average overlaps for different methods on each sequence under different QPs, while Table VIII summarizes their average values of all QPs. We can see that AOT and CBS have the same precision of object representation, both much better than FCB. This is because FCB determines the CB size by the mode decision process with minimizing the RD cost. Therefore, the decided CB might contain both foreground and background pixels. In such a case, no matter what it was marked as foreground or background CB, the overlap would be decreased. Instead, AOT and CBS can describe the objects shape exactly. The reason for the precision is not 100% is that the smallest block employed in the experiments is 8×8 CB. Theoretically speaking, the precision of AOT and CBS is 100% if the smallest block employed is one pixel. Similar observations can be found in Fig. 8, where the representation precision of AOT and CBS keeps unchanged for different QPs, while that of FCB decreases gradually with the increase of QP.

By summarizing the above two cases, we can safely conclude that $O(\text{AOT}) = O(\text{CBS}) > O(\text{FCB})$, where $O(x)$ denotes the overlap of method x and can be used to measure its object representation precision.

In summary, FCB is a straightforward method to extend the HEVC framework to represent visual objects, with the lowest additional bitrate. Instead, both AOT and CBS can be used to more accurately represent visual objects in the HEVC coding loop, despite they consume slightly more additional bits than

TABLE VIII
AVERAGE OVERLAPS FOR DIFFERENT METHODS ON EACH SEQUENCE

Video	FCB	AOT	CBS
Kimono	93.68%	97.42%	97.42%
ParkScene	86.88%	93.14%	93.14%
BasketballDrive	90.24%	95.91%	95.91%
Johnny	93.27%	97.83%	97.83%
KristenAndSara	91.77%	97.69%	97.69%
FourPeople	84.32%	96.25%	96.25%
Average	**90.03%**	**96.37%**	**96.37%**

TABLE IX
OVERALL BIT COSTS AND OVERLAPS OF ACE AND FCB

Video	ACE vs Anchor			FCB vs Anchor		
	Bitrate increase (kbps)	BD rate increase	Overlap	Bitrate increase (kbps)	BD rate increase	Overlap
Kimono	7.61	0.60%	85.51%	19.84	1.40%	93.68%
ParkScene	8.20	0.40%	59.68%	15.15	0.80%	86.88%
Basket-ballDrive	31.62	0.60%	71.03%	106.81	2.50%	90.24%
Johnny	9.94	2.30%	94.84%	27.20	5.70%	93.27%
Kris-tenAndSara	11.10	1.70%	93.51%	31.03	4.50%	91.77%
FourPeople	11.41	1.30%	88.02%	33.40	3.50%	84.32%
Average	**13.313**	**1.15%**	**82.10%**	**38.905**	**3.07%**	**90.03%**

FCB. Moreover, since the block splitting procedure is to jointly optimize compression efficiency and object representation, CBS provides a potential choice for further enhancing the object-based coding in HEVC (e.g., using a smaller QP for objects than background).

D. Comparison With the Alpha Channel Encoding Method

As discussed in Section II, despite several methods were proposed to represent visual objects in the coded stream such as MPEG-4 boundary coding [1] and ST-SPIHT [7], there is few work so far that utilizes the quadtree structures in HEVC for the purpose. Similar to [5], we implement a straightforward method that encodes the binary alpha plane utilizing the HEVC coding tools as a supplementary video stream of the original video. Here we call this method as ACE (Alpha Channel Encoding). Thus, the final experiment is to compare the proposed methods with ACE. The experimental result is shown in Table IX. For simplicity, here we use only FCB as the representative in the comparison since FCB is also a straightforward method to extend the HEVC framework for object representation.

From Table IX, the average bitrate increase in ACE is smaller than FCB because ACE utilizes the techniques of intra-/inter-picture prediction, quantization and scaling, transform, etc. However, ACE also induces a significant loss in object representation, especially on the videos of Class A. On all the sequences, the representation precision of FCB outperforms that of ACE by about 8% on average. For the purpose of representing objects, the precision is more important if the bit cost is not large. Therefore, FCB should be a more appropriate method than ACE for visual objects representation in HEVC. Similar conclusion can also be applied to AOT and

Fig. 9. Subjective quality comparison of different representing methods.

CBS because their representation precision is even better than FCB.

To subjectively compare the representation quality among the proposed three methods and ACE, Fig. 9 visualizes some object representation results of different methods on several frames from two testing sequences with complex scenes (e.g., BasketballDrive and FourPeople). We can see that among the four methods, the representation quality of AOT and CBS is definitely the best on all these frames. Meanwhile, the representation quality of FCS is significantly better than ACE on the frames of BasketballDrive; but on the frames of FourPeople, their difference is visually imperceptible. It should be noted that on the frames of both two videos, the representation results of ACE have a remarkably serrated border. Clearly, these observations are consistent with the quantitative results shown in Tables VIII and IX.

VI. Conclusion

This paper proposes to reuse the key coding tools in HEVC originally designed for compression to represent the foreground objects with arbitrary shape. By employing the variable-size blocks and the entropy coding tool, we have investigated three methods which can represent and encode the object shape in the HEVC coding loop with a small bitrate cost. The main contributions of this paper can be summarized as follows.

1) For the first time, we discovered the new usage of the HEVC coding besides the compression. That means HEVC can be extended to represent the shape of visual objects in a video without additional coding tools, at a reasonably low cost on bitrate increase.
2) Three specific methods are proposed to represent visual objects in the HEVC coding loop. The experimental results on six manually segmented 1080P or 720P videos show that all the methods can be potentially applicable for different practical applications.

From a more general view, FCB, AOT, and CBS proposed in this paper are three typical methods to accurately represent visual objects in the video pictures (at the smallest block level). If the distortion on the representation is allowed to a larger variable range, the bitrate cost can be further reduced, especially when the representation precision is considered as a parameter in the distortion model for the whole picture.

Moreover, in our experiments, the smallest block flagged as inside-block is the 8×8 CU. It is possible to employ 4×4 blocks by flagging leaf TUs or PUs to better fit the object shape. If necessary, the object tree can be further split till to the pixel level to perfectly fit the object contour. This is another valuable area to study in the future.

For the applications sensitive to the bitrate cost, there are many ways to reduce the bitrate. One straightforward method is to make use of the correlation between the shapes of objects in the neighbor pictures with the intra-/inter-picture prediction tools. Another way is to optimize the HEVC entropy coding tool. Currently, to compress the additional bits for object representation, the context model for the coding tree is reused with the same initial parameters. Obviously, a better context model that reflects the statistical characteristics can further reduce the bitrate cost.

Even though this study is based on the manually-segmented ground truth, the experimental results can still inspire us to investigate the relation between video compression and visual object recognition such that they can share the same coding module at the representation level. As the cost for the object representation in HEVC is relatively low, it is worth taking into account the automatic object detection and representation in the high efficient coding framework. One solution is to merge the precision of object representation and the distortion of object region coding into a unified criteria for picture distortion evaluation, thus the coding efficiency, the object representation and the visual quality of the object regions can be jointly optimized in a new rate-distortion model.

According to the evidence revealed in this paper, the visual object representation is worth being one of the prior options for the extension of the HEVC standard. The new capability will enlarge the applications of HEVC from compression to a wide spectrum of machine vision tasks.

REFERENCES

[1] Coding of audio-visual objects—Part 2: Visual Apr. 1999, ISO/IEC JTC 1, ISO/IEC 14496-2 (MPEG-4 visual ver. 1).

[2] H. G. Musmann, M. Hotter, and J. Ostermann, "Object-based analysis-synthesis of moving images," *Image Commun.*, vol. 1, no. 2, pp. 117–138, 1989.

[3] High Efficiency video coding (HEVC) text specification draft 10 (for FDIS & consent) Jan. 2013, JCT-VC, doc. JCTVC-L1003.

[4] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[5] M. Naccari and M. Mrak, "Binary alpha channel compression for coding of supplementary video streams," in *Proc. IEEE Int. Workshop Multimedia Signal Process.*, Sep. 2013, pp. 200–205.

[6] F. Pereira and T. Ebrahimi, *The MPEG-4 Book*. Upper Saddle River, NJ: Prentice-Hall, Jul. 2002.

[7] K. Martin, R. Lukac, and K. N. Plataniotis, "SPIHT-based coding of the shape and texture of arbitrarily shaped visual objects," *IEEE Trans Circuits Syst. Video Technol.*, vol. 16, no. 10, pp. 1196–1208, Oct. 2006.

[8] X. Zhang, Y. Tian, T. Huang, and W. Gao, "Low-complexity and high-efficiency background modeling for surveillance video coding," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process.*, Nov. 2012, pp. 1–6.

[9] Joint call for proposals on video compression technology Jan. 2010, ITU-T and ISO/IEC JTC 1, ITU-T SG16/Q6 VCEG-AM91 and ISO/IEC MPEG doc. N11113.

[10] Common conditions and software reference configurations Feb. 2012, JCT-VC, joint collaborative team on video coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, doc. JCTVC-H1100.

[11] High efficiency video coding (HEVC) test model 12 (HM12) encoder description Aug. 2013, JCT-VC, JCT-VC N1002.

[12] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, The PASCAL visual object classes challenge 2009 (VOC2009) results [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html

Tiejun Huang (M'01–SM'12) received the B.S. and M.S. degrees in computer science from Wuhan University of Technology, Wuhan, China, in 1992 and 1995, respectively, and the Ph.D. degree in pattern recognition and intelligent system from Huazhong (Central China) University of Science and Technology, Wuhan, China, in 1998.

He is a Professor of the School of Electronic Engineering and Computer Science, the Director of the Institute for Digital Media Technology, Peking University. His research area includes video coding, image understanding, digital right management (DRM), and digital library. He published more than 100 peer-reviewed papers and three books as author or co-author.

Prof. Huang is a member of the Board of Directors for Digital Media Project, the Advisory Board of IEEE Computing Now, and the Board of Chinese Institute of Electronics.

Siwei Dong received the B.S. degree from Chongqing University, Chongqing, China, in 2012. He is currently working toward the Ph.D. degree at the School of Electrical Engineering and Computer Science, Peking University, Beijing, China.

His research interests include surveillance video coding and multimedia learning.

Yonghong Tian (M'05–SM'10) received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2005.

He is currently a Professor with the National Engineering Laboratory for Video Technology, School of Electronics Engineering and Computer Science, Peking University, Beijing, China. His research interests include computer vision, multimedia analysis, and coding. He is the author or coauthor of over 100 technical articles in refereed journals and conferences. He is currently a Young Associate Editor of the *Frontiers of Computer Science in China*.

Dr. Tian a member of the IEEE TCMC-TCSEM Joint Executive Committee in Asia (JECA).