creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사학위논문

Redundancy Reduction in Interpolation Calculation for HEVC Fractional Motion Estimation

HEVC의 소수 단위 움직임 추정을 위한 보간 필터 중복 연산 감소 방법

2016년 8월

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major
Nguyen Ngoc Luong

Redundancy Reduction in Interpolation Calculation for HEVC Fractional Motion Estimation

지도교수 이혁재

이논문을 공학석사 학위논문으로 제출함

2016 년 08 월

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major

Nguyen Ngoc Luong

Nguyen Ngoc Luong 의공학석사 학위논문을

인준함

2016 년 08 월

Chair Soo-Ik Chae (Seal)
Vice Chair Huyk-Jae Lee (Seal)
Examiner Kiyoung Choi (Seal)

Abstract

High-Efficiency Video Coding (HEVC) [1] is the latest video coding standard established by Joint Collaborative Team on Video Coding (JCT-VC) aiming to achieve twice encoding efficiency with comparatively high video quality compared to its predecessor, the H.264 standard. Motion Estimation (ME) which consists of integer motion estimation (IME) and fractional motion estimation (FME) is the bottleneck of HEVC computation. In the execution of the HM reference software, ME alone accounts for about 50 % of the execution time in which IME contributes to about 20 % and FME does around 30% [2].The FME's enormous computational complexity can be explained by two following reasons:

- A large number of FME refinements processed: In HEVC, a frame is divided into CTU, whose size is usually 64x64 pixels. One 64x64 CTU consists of 85 CUs including one 64x64 CU at depth 0, four 32x32 CUs at depth 1, 16 16x16 CUs at depth 2, and 64 8x8 CUs at depth 3. Each CU can be partitioned into PUs according to a set of 8 allowable partition types. An HEVC encoder processes FME refinement for all possible PUs with usually 4 reference frames before deciding the best configuration for a CTU. As a result, typically in HEVC's reference software, HM, for one CTU, it has to process 2,372 FME refinements, which consumes a lot of computational resources.
- A complicated and redundant interpolation process: Conventionally, FME refinement, which consists of interpolation and sum of absolute transformed difference (SATD), is processed for every PU in 4 reference frames. As a result, for a 64x64 CTU, in order to process fractional pixel refinement, FME needs to interpolate 6,232,900 fractional pixels. In addition, In HEVC, fractional pixels which consist half fractional pixels and quarter fractional pixels, are interpolated by 8-tap filters and 7-tap filters instead of 6-tap filters and bilinear filters as previous standards. As a result, interpolation process in FME imposes an extreme computational burden on HEVC encoders.

This work proposes two algorithms which tackle each one of the two above reasons. The first algorithm, *Advanced Decision of PU Partitions and CU*

Depths for FME, estimates the cost of IMEs and selects the PU partition types at the CU level and the CU depths at the coding tree unit (CTU) level for FME. Experimental results show that the algorithm effectively reduces the complexity by 67.47% with a BD-BR degrade of 1.08%. The second algorithm, *A Reduction of the Interpolation Redundancy for FME*, reduces up to 86.46% interpolation computation without any encoding performance decrease. The combination of the two algorithms forms a coherent solution to reduce the complexity of FME. Considering interpolation is a half of the complexity of an FME refinement, then the complexity of FME could be reduced more than 85% with a BD-BR increase of 1.66%

Keyword: High-Efficiency Video Coding; Motion estimation; Fractional motion estimation; Interpolation; Complexity Reduction

Student Number: 2014-25271

List of Figures

List of Tables

Table of Contents

Chapter 1. Introduction

1. Introduction to Video Coding

1.1. Definition of Video Coding

- Historically, video was stored as an analog signal on magnetic tape. Around the time when the compact disc entered the market as a digital format replacement for analog audio, it became feasible to also store and convey video in digital form. Because of a large amount of storage and bandwidth needed to record and convey raw video, a method was needed to reduce the amount of data used to represent the raw video. Since then, engineers and mathematicians have developed a number of solutions for achieving this goal that involves compressing the digital video data. Video compression is reducing the amount of data used to represent the raw video. The process of reducing the size of a video file is referred to as video coding or video compression. Video compression or video coding is the process of compressing (encoding) and decompressing (decoding) video.

1.2. The Need of Video Coding

- Virtually any digital video we encounter is distributed in a compressed format. It is because raw video data would require bandwidth and storage space far in excess of that available. For example, a raw full HD color video data (without video compression) containing 30 frames per second would require a bandwidth of:

  (1920*1080*8)*3*30 = 1.5 Gb/s

- A bandwidth of 1.5 Gb/s is way too high for current communication channels and a 100 Gigabytes hard disk can store only13 minutes of a raw Full HD video.

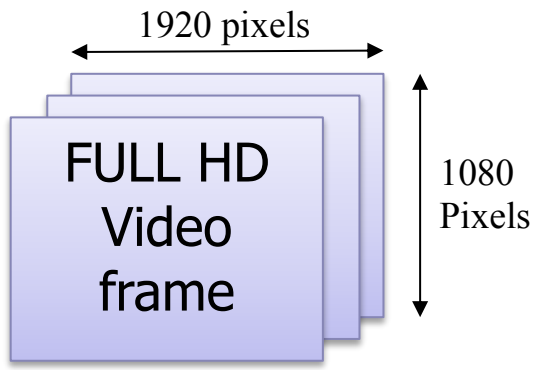


Fig. 1: Full HD Video Frames

- As mentioned above, uncompressed video signals generate a huge quantity of data and video use has become more and more ubiquitous. There is also a constant hunger for higher quality video—e.g., in the form of higher resolutions, higher frame rates, and higher fidelity—as well as a hunger for greater access to video content. Moreover, the creation of video content has moved from the being the exclusive domain of professional studios toward individual authorship, real-time video chat, remote home surveillance, and even "always on" wearable cameras. As a result, video traffic is the biggest load of communication networks and data storage worldwide—a situation that is unlikely to fundamentally change; although anything that can help ease the burden is an important development. As a result, video compression is extremely necessary to save bandwidth and storage memory for videos.

1.3. Basics of Video Coding.

- Most video coding algorithms and codecs combine spatial image compression and temporal motion compensation. Video compression is a practical implementation of source coding in information theory. In practice, most video codecs also use audio compression techniques in parallel to compress the separate, but combined data streams as one package.
- The majority of video compression algorithms use lossy compression. As in all lossy compression, there is a trade-off between video qualities, the cost of processing the compression and decompression, and system requirements. Highly compressed video may present visible or distracting artifacts.
- Some video compression schemes typically operate on square-shaped groups of neighboring pixels, often called macroblocks. These pixel groups or blocks of pixels are compared from one frame to the next, and the video compression codec sends only the differences within those blocks. In areas of video with more motion, the compression must encode more data to keep up with the larger number of pixels that are changing. Commonly during explosions, flames, flocks of animals, and in some panning shots, the high-frequency detail leads to quality decreases or to increases in the variable bit-rate.

1.4. Video Coding Standard

- Standards define a common language that different parties can use so that they can communicate with one another. Standards are thus, a prerequisite to effective communication. Video coding standards define the bitstream syntax, the language that the encoder and the decoder use to communicate. Besides defining the bitstream syntax, video coding standards are also required to be efficient, in that they should support good compression algorithms as well as allow the efficient implementation of the encoder and decoder.
- Multimedia communication is greatly dependent on good standards. The presence of standards allows for a larger volume of information exchange, thereby benefiting the equipment manufacturers and service providers. It also benefits customers, as now they have a greater freedom to choose between manufacturers. All in all, standards are a prerequisite to multimedia communication.
- Since the early 1990s, the development of video coding standards has been driven by two parallel application spaces: real-time video communication and distribution or broadcast of video content. The corresponding specifications have been published by two main standardization bodies, the International Telecommunications Union (ITU) and the International Standardization Organization/ International Electrotechnical Commission (ISO/IEC). Here, a brief overview of the evolution of video coding standards is provided with a focus on the main corresponding application scenarios and the corresponding main technical achievements. For the sake of simplicity both, ITU recommendations and ISO/IEC standards are referred to as standards in this section. The differences between the two are detailed below. An overview of the timeline of the major standards in the two standardization bodies is shown in Fig. 2. For all standards listed in this timeline, several corrigenda and extensions have been published over the time. Here, the publication dates of the key versions of the standards have been included. It can be seen that while having started on separate tracks, the two standardization organizations have engaged in increasingly close collaboration, specifically for achieving the latest milestones AVC and HEVC:
    - ITU-T standards proposed by ITU-T Video Coding Experts Group (VCEG) include the likes of H.261, H.262, H.263, and H.26L.

- ISO/IEC standards proposed by ISO/IEC Moving Picture Experts Group (MPEG) include the likes of MPEG1, MPEG2, and MPEG4.
- The two groups VCEG and MPEG then joined together to form The Joint Video Team (JVT) which proposed H.264 and HEVC (H.265), the next generations of video coding standard.

- The following presents the chronology of Video Coding Standards:

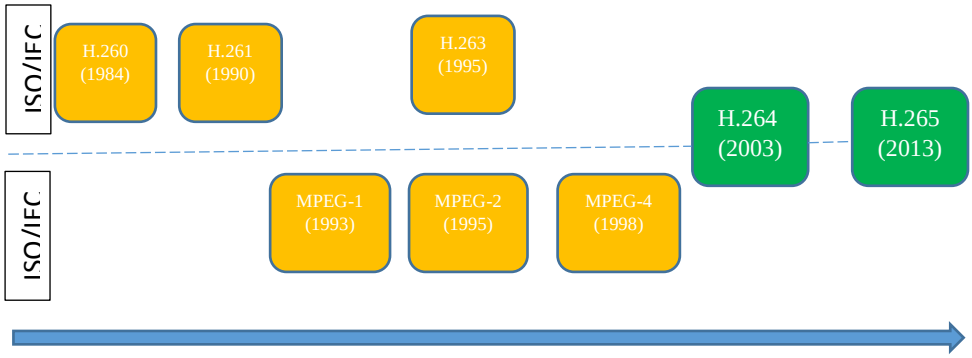


Fig. 2: Chronology of Video Coding Standards

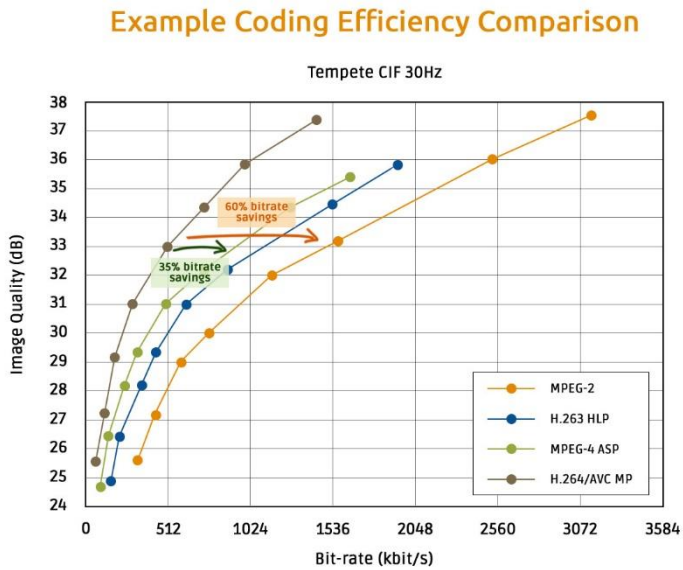


Fig. 3: Coding Efficiency Comparison for Video Coding Standards

- An example of coding efficiency comparison for Video Coding Standards is illustrated in Fig. 3. H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) is a video compression format that is currently one of the most commonly used formats for the recording, compression, and distribution of video content.

2. Introduction to HEVC

2.1. HEVC Background and Development

- The HEVC project was formally launched in January 2010 when a joint Call for Proposals (CfP) was issued by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). Before launching the formal CfP, both organizations had conducted investigative work to determine that it was feasible to create a new standard that would substantially advance the state of the art in compression capability—relative to the prior major standard known as H.264/MPEG-4 Advanced Video Coding (AVC- the first version of which was completed in May 2003). One notable aspect of the investigative work toward HEVC was the "key technology area" (KTA) studies in VCEG that began around the end of 2004 and included the development of publicly-available KTA software codebase for testing various promising algorithm proposals. In MPEG, several workshops were held, and a Call for Evidence (CFE) was issued in 2009. When the two groups both reached the conclusion that substantial progress was possible and that working together on the topic was feasible, a formal partnership was established and the joint CfP was issued. The VCEG KTA software and the algorithmic techniques found therein were used as the basis of many of the proposals submitted in response to both the MPEG CfE and the joint CfP.
- The major video coding standard directly preceding the HEVC project was H.264/MPEG-4 AVC, which was initially developed in the period between 1999 and 2003, and then was extended in several important ways from 2003–2009. H.264/MPEG-4 AVC has been an enabling technology for digital video in almost every area that was not previously covered by H.262/MPEG-2 Video and has substantially displaced the older standard within its existing application domains. It is widely used for many applications, including broadcast of high definition (HD) TV signals over satellite, cable, and terrestrial transmission systems, video content acquisition and editing systems, camcorders, security applications, Internet and mobile network video, Blu-ray Discs, and real-time conversational applications such as video chat, video conferencing, and telepresence systems.

- However, an increasing diversity of services, the growing popularity of HD video, and the emergence of beyond HD formats (e.g., 4k×2k or 8k×4k resolution) are creating even stronger needs for coding efficiency superior to H.264/MPEG-4 AVC's capabilities. The need is even stronger when higher resolution is accompanied by stereo or Multiview capture and display. Moreover, the traffic caused by video applications targeting mobile devices and tablet PCs, as well as the transmission needs for video-on-demand services, are imposing severe challenges on today's networks. An increased desire for higher quality and resolutions is also arising in mobile applications. Interest in developing a new standard has been driven not only by the simple desire to improve compression as much as possible—e.g., to ease the burden of video on storage systems and global communication networks, but also to help enable the deployment of new services, including capabilities that have not previously been practical—such as ultra-high-definition television (UHDTV) and video with higher dynamic range, wider color gamut, and greater representation precision than what is typically found today.
- To formalize the partnership arrangement, a new joint organization was created, called the Joint Collaborative Team on Video Coding (JCT-VC). The JCT-VC met four times per year after its creation, and each meeting had hundreds of attending participants and involved the consideration of hundreds of contribution documents (all of which were made publicly available on the web as they were submitted for consideration).
- The project had an unprecedented scale, with a peak participation reaching about 300 people and more than 1,000 documents at a single meeting. Meeting notes were publicly released on a daily basis during meetings, and the work continued between meetings, with active discussions by email on a reflector with a distribution list with thousands of members, and with formal coordination between meetings in the form of work by "ad hoc groups" to address particular topics and "core experiments" to test various proposals. Essentially the entire community of relevant companies, universities, and other research institutions was attending and actively participating as the standard was developed.
- HEVC has been designed to address essentially all existing applications of H.264/MPEG-4 AVC and to particularly focus on two

key issues: increased video resolution and increased use of parallel processing architectures. The syntax of HEVC is generic and should also be generally suited for other applications that are not specifically mentioned above.

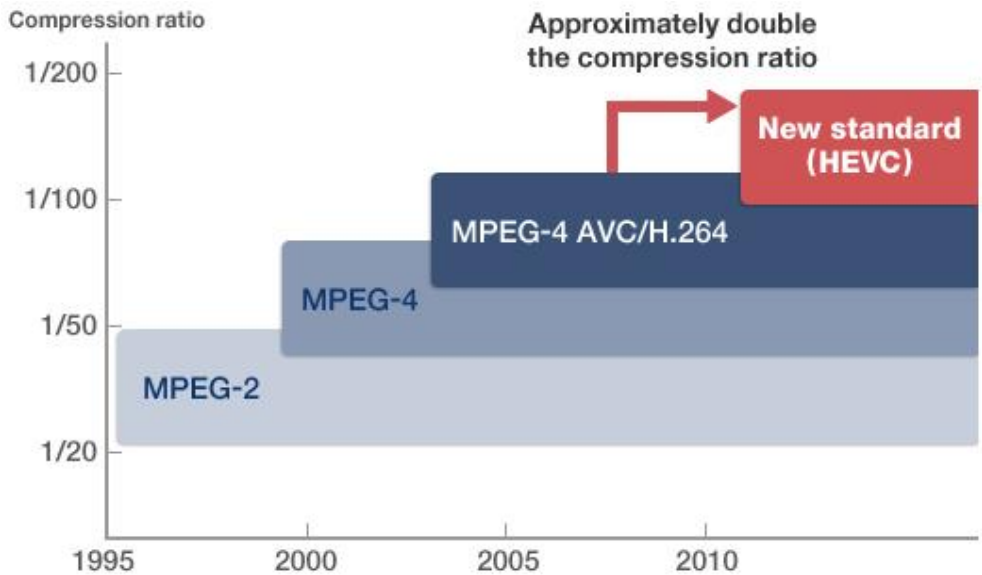


Fig. 4: HEVC compression ratio comparison

- As has been the case for all past ITU-T and ISO/IEC video coding standards, in HEVC only the bitstream structure and syntax is standardized, as well as constraints on the bitstream and its mapping for the generation of decoded pictures. The mapping is given by defining the semantic meaning of syntax elements and a decoding process such that every decoder conforming to the standard will produce the same output when given a bitstream that conforms to the constraints of the standard. This limitation of the scope of the standard permits maximal freedom to optimize implementations in a manner appropriate to specific applications (balancing compression quality, implementation cost, time to market, and other considerations). However, it provides no guarantees of end-to-end reproduction quality, as it allows even crude encoding techniques to be considered conforming.
- To assist the industry community in learning how to use the standard, the standardization effort not only includes the development of a text

specification document but also reference software source code as an example of how HEVC video can be encoded and decoded. The draft reference software has been used as a research tool for the internal work of the committee during the design of the standard, and can also be used as a general research tool and as the basis of products. A standard test data suite is also being developed for testing conformance to the standard.

2.2. Block Partitioning Structure in HEVC.

- The HEVC standard has adopted a highly flexible and efficient block partitioning structure by introducing four different block concepts: Coding Tree Unit (CTU), Coding Unit (CU), Prediction Unit (PU), and Transform Unit (TU), which are defined to have clearly separated roles. The terms Coding Tree Block (CTB), Coding Block (CB), Prediction Block (PB), and Transform Block (TB) are also defined to specify the 2-D sample array of one color component associated with the CTU, CU, PU, and TU, respectively. Thus, a CTU consists of one luma CTB, two chroma CTBs, and associated syntax elements. A similar relationship is valid for CU, PU, and TU. Although the use of a quadtree structure in video compression is not a new concept, the coding tree approach in HEVC can bring additional coding efficiency benefits by incorporating PU and TU quadtree concepts for video compression. Leaf nodes of a tree can be merged or combined in a general quadtree structured video coding scheme. After the final quadtree is formed, motion information is transmitted to the leaf nodes of the tree. L-shaped or rectangular-shaped motion partition is possible through merging and combination of nodes. However, in order to make such shapes, the merge process should be followed using smaller blocks after further splitting occurs. In the HEVC block partitioning structure, such cases are taken care of by the PU. Instead of splitting one depth more for merging and combination, predefined partition modes such as PART−2N×2N, PART−2N×N, and PART−N×2N are tested and the optimal partition mode is selected at the leaf nodes of the tree. It is worthwhile mentioning that PUs still can share motion information through merging mode in HEVC. Although a general quadtree structure without PU concept was investigated by removing the symmetric rectangular partition modes

(PART−2N×N and PART−N×2N) from the syntax and replaced by corresponding merge flags, both coding efficiency and complexity was proved inferior to the current design.

- Another difference is the transform tree. Even though variable block size transforms were used for quadtree structure motion compensation, their usage was rather restricted. For example, transform size was strictly combined with motion compensation block size. Even though multiple transform size could be utilized, it was usual to use same size transform in a motion compensated block. In HEVC, the motion compensated residual can be transformed with a quadtree structure, and the actual transform is performed at leaf nodes. Since the transform tree is rooted from the leaf nodes of coding tree, this creates a nested quadtree. This kind of nested quadtree exists since the transform tree is started from the CU regardless of partition modes, i.e., PU shapes. This is a way to construct a nested quadtree even though we have PU concepts that differ from a general quadtree structure.
- Another noticeable aspect is the full utilization of depth information for entropy coding. For example, entropy coding of HEVC is highly reliant on the depth information of quadtree. For syntax elements such as inter−pred−idc, split−transform−flag, cbf−luma, cbf−cb and cbf−cr, depth dependent context derivation is heavily used for coding efficiency. It has been demonstrated that this can break the dependency with neighboring blocks with less line buffer requirement in hardware implementations because information of above CTU does not need to be stored. In the following sections, the block partitioning structures in the HEVC standard are presented in conjunction with a detailed explanation of those unit definitions.

a. Coding Tree Unit

- A slice contains an integer multiple of CTU, which is an analogous term to the macroblock in H.264/AVC. Inside a slice, a raster scan method is used for processing the CTU.
- In main profile, the minimum and the maximum sizes of CTU are specified by the syntax elements in the sequence parameter set (SPS) among the sizes of 8×8, 16×16, 32×32, and 64×64. Due to this flexibility of the CTU, HEVC provides a way to adapt according to various application needs such as encoder/decoder pipeline delay

constraints or on-chip memory requirements in a hardware design. In addition, the support of large sizes up to 64×64 allows the coding structure to match the characteristics of the high definition video content better than previous standards; this was one of the main sources of the coding efficiency improvements seen with HEVC.

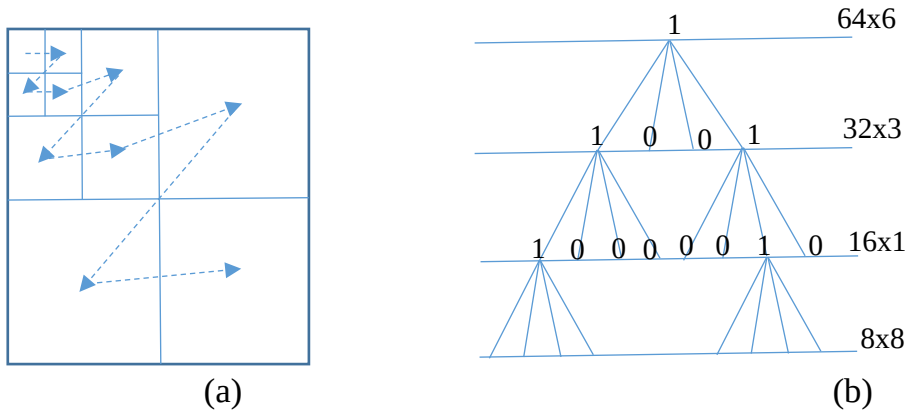


Fig. 5: Example of CTU partitioning and processing order when size of CTU is equal to 64 × 64 and minimum CU size is equal to 8 × 8. (a) CTU partitioning. (b) Corresponding coding tree structure.

b. Coding Unit

- The CTU is further partitioned into multiple CU to adapt to various local characteristics. A quadtree denoted as the coding tree is used to partition the CTU into multiple CUs.
    - *Recursive Partitioning from CTU*: Let CTU size be 2N×2N where N is one of the values of 32, 16, or 8. The CTU can be a single CU or can be split into four smaller units of equal sizes of N×N, which are nodes of the coding tree. If the units are leaf nodes of coding tree, the units become CUs. Otherwise, it can be split again into four smaller units when the split size is equal or larger than the minimum CU size specified in the SPS. This representation results in a recursive structure specified by a coding tree. Fig. 5 illustrates an example of CTU partitioning and the processing order of CUs when the size of CTU is equal to 64 × 64 and the minimum CU size is equal to 8 × 8. Each square block in Fig. 5(a) represents CU. In this example, a CTU is split into 16 CUs which have different sizes and positions.

Fig. 5(b) shows corresponding coding tree structure representing the structure of the CTU partitioning in Fig. 5(a). Numbers on the tree represent whether the CU is further split. In Fig. 5(a), CUs are processed by following the dotted line. This processing order of CUs can be interpreted as a depth first traversing in the coding tree structure. If CTU size of 16 × 16 and the minimum CU size of 8 × 8 are used, the resultant structure is roughly similar to that of H.264/AVC. HEVC utilizes CU as a unit to specify which prediction scheme is used for intra and inter predictions. Since the minimum CU size can be 8 × 8, the minimum granularity for switching different prediction schemes is 8 × 8, which is smaller than the macroblock size of H.264/AVC.

- *Benefits of Flexible CU Partitioning Structure:* This kind of flexible and recursive representation provides several major benefits. The first benefit comes from the support of CU sizes greater than the conventional 16×16 size. When the region is homogeneous, a large CU can represent the region by using a smaller number of symbols than is the case using several small blocks.

  Fig. 6 shows rate-distortion curves of several combinations of the size of CTU and maximum coding tree depth for Traffic 2560×1600@30 Hz sequence. The results are obtained using HM-6.0 Main profile using low delay constraint of the common test condition of HEVC. The size of CTU is represented by character "s" and maximum coding tree depth is represented by character "h" in the figure. Each curve shows the result when s64h4, s16h2, and s64h2 are used, respectively. There is a big gap of coding efficiency about 13.7% in Bjøntegaard delta bitrate between s64h4 and s16h2. This result illustrates that adding large size CU is an effective means to increase coding efficiency for higher resolution content. Coding efficiency difference between s64h4 and s64h2 is about 19.5% and it is also noticeable that coding efficiency difference between s64h2 and s16h2 is similar at low bit rate, but s16h2 shows better coding efficiency at high bit rate because smaller size blocks cannot be utilized for s64h2, where minimum CU size is 32 × 32. These results can be

interpreted as showing that large size CU is important to increase coding efficiency in general but still small size CU should be used together to cover regions which large CU cannot be applied to successfully.

Furthermore, supporting arbitrary sizes of CTU enables the codec to be readily optimized for various content, applications, and devices. Compared to the use of fixed size macroblock, support of various sizes of CTU is one of the strong points of HEVC in terms of coding efficiency and adaptability for contents and applications. This ability is especially useful for low-resolution video services, which are still commonly used in the market. By choosing an appropriate size of CTU and maximum hierarchical depth, the hierarchical block partitioning structure can be optimized to the target application.

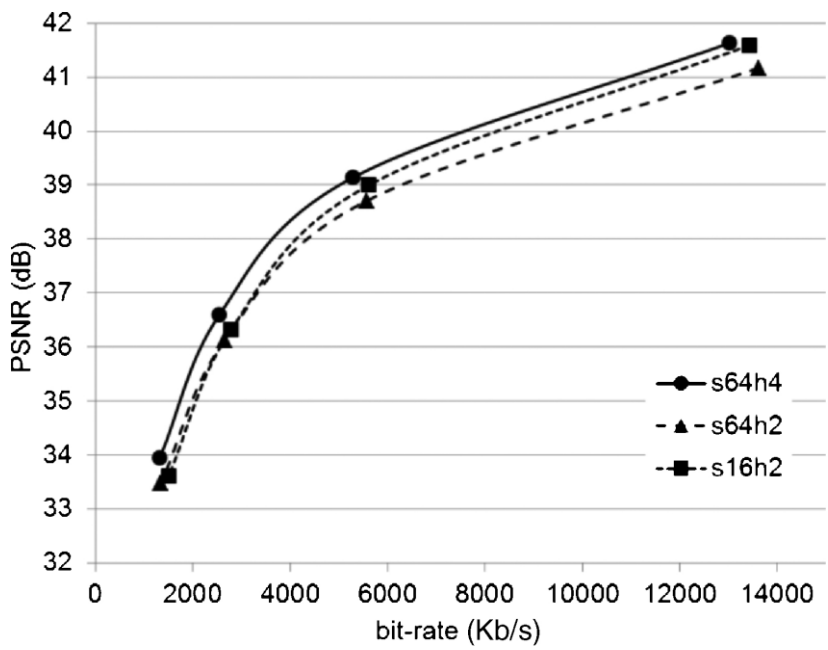


Fig. 6: Rate-distortion curves of several combinations of the size of CTU and maximum coding tree depth for Traffic sequences (2560 × 1600). The size of CTU is represented by character “s” and maximum coding tree depth is represented by character “h.” Each curve shows the result when s64h4, s16h2, and s64h2 are used, respectively

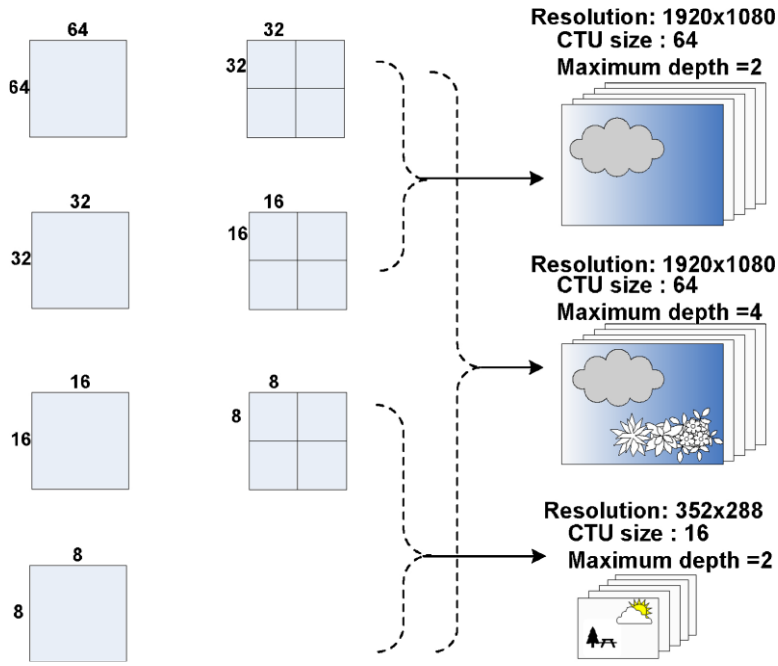


Fig. 7: Example of CTU size and various CU sizes for various resolutions.

Table 1: Simplified Form of Coding Tree Syntax Table

coding−tree(x0, y0, log2CbSize, cbDepth) {
split−coding−unit−flag[x0][y0]
if(split−coding−unit−flag[x0][y0]) {
coding−tree(x0, y0, log2CbSize−1, cbDepth+1)
coding−tree(x1, y0, log2CbSize−1, cbDepth+1)
coding−tree(x0, y1, log2CbSize−1, cbDepth+1)
coding−tree(x1, y1, log2CbSize−1, cbDepth+1)
} else {
coding−unit(x0, y0, log2CbSize)
}
}

Fig. 7 shows examples of various CTU sizes and CU sizes suitable for different resolutions and types of content. For example, for an application using 1080p content that is known to include only simple global motion activities, a CTU size of

64 and depth of 2 may be an appropriate choice. For more general 1080p content, which may also include complex motion activities of small regions, a CTU size of 64 and a maximum depth of 4 would be preferable.

Finally, by eliminating the distinction between macroblock and sub macroblock and using only CU, the multilevel hierarchical quadtree structure can be specified in a very simple and elegant way. Together with the size-independent syntax representation, syntax items of one general size may be specified for the remaining coding tools.

Table 1 shows the recursive part of the coding tree syntax in simplified form. As shown in the table, the splitting process of coding tree can be specified recursively and all other syntax elements can be represented in the same way regardless of the size of CU. This kind of recursive representation is very useful in terms of reducing parsing complexity and improving clarity when the quadtree depth is large.

c. Prediction Unit

- One or more PUs are specified for each CU, which is a leaf node of coding tree. Coupled with the CU, the PU works as a basic representative block for sharing the prediction information. Inside one PU, the same prediction process is applied and the relevant information is transmitted to the decoder on a PU basis. A CU can be split into one, two or four PUs according to the PU splitting type. HEVC defines two splitting shapes for the intra-coded CU and eight splitting shapes for inter-coded CU. Unlike the CU, the PU may only be split once.
    - *PU Splitting Type*: Similar to prior standards, each CU in HEVC can be classified into three categories: skipped CU, inter-coded CU, and intra-coded CU. An inter-coded CU uses motion compensation scheme for the prediction of the current block while an intra-coded CU uses neighboring reconstructed samples for the prediction. A skipped CU is a special form of inter-coded CU where both the motion vector difference and the residual energy are equal to zero. For each category, PU splitting type is specified differently as shown in Fig. 8 when the CU size is equal to 2N×2N. As shown in the figure, only

PART−2N×2N PU splitting type is allowed for the skipped CU.

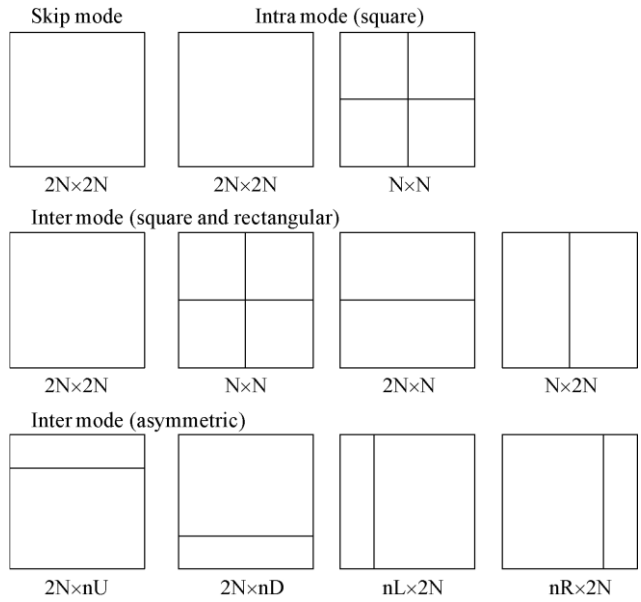


Fig. 8: Illustration of PU splitting types in HEVC.

For the intra-coded CU, two possible PU splitting types of PART−2N×2N and PART−N×N are supported. Finally, total eight PU splitting types are defined as two square shapes (PART−2N×2N, PART−N×N), two rectangular shapes (PART−2N×N and PART−N×2N), and four asymmetric shapes (PART−2N×nU, PART−2N×nD, PART−nL×2N, and PART−nR×2N) for inter-coded CU. Although more sophisticated partitioning was considered, but current PU splitting types were chosen as a good tradeoff between encoding complexity and coding efficiency.

Note that all information related to the prediction scheme is specified on a PU basis. For instance, the most probable mode index and intra prediction mode for intra coded CU or merge flag, merge index, inter prediction flag, motion vector prediction index, reference index, and motion vector difference for inter-coded CU are unique per PU. For most cases, PU partitioning of chroma block shares the same splitting of luma component; however, when the CU size is

equal to 8×8 and PART−N×N is used for the PU splitting type, PART−2N×2N is used for the chroma block to prevent the block size from being less than 4 × 4.

- *Constraints According to CU Size*: In PART−N×N, CU is split into four equal sizes PUs, which is conceptually similar to the case of four equal-size CUs when the CU size is not equal to the minimum CU size. Thus, HEVC disallows the use of PART−N×N except when the CU size is equal to the minimum CU size. It was observed that this design choice can reduce the encoding complexity significantly while the coding efficiency loss is marginal.

  To reduce the worst-case complexity, HEVC further restricts the use of PART−N×N and asymmetric shapes. In the case of inter-coded CU, the use of PART−N×N is disabled when the CU size is equal to 8 × 8. Moreover, asymmetric shapes for inter-coded CU are only allowed when the CU size is not equal to the minimum CU size.

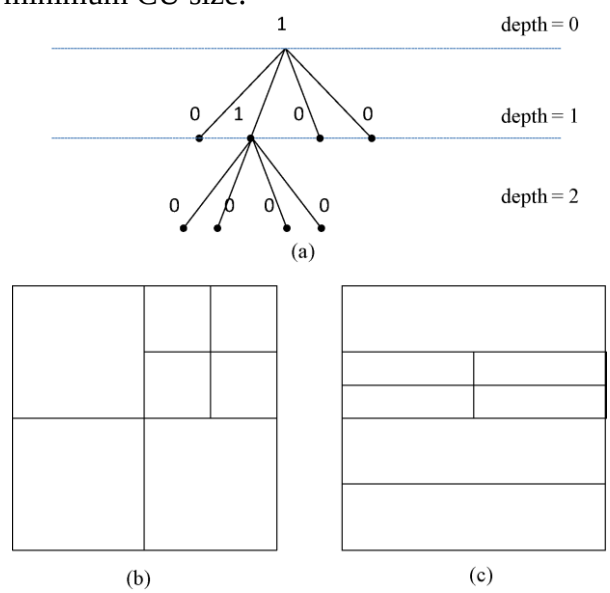


Fig. 9: Examples of transform tree and block partitioning. (a) Transform tree. (b) TU splitting for square-shaped PU. (c) TU splitting for rectangular or asymmetric shaped PU.

d. Transform Unit

- Similar with the PU, one or more TUs are specified for the CU. HEVC allows a residual block to be split into multiple units recursively to form another quadtree which is analogous to the coding tree for the CU. The TU is a basic representative block having residual or transform coefficients for applying the integer transform and quantization. For each TU, one integer transform having the same size to the TU is applied to obtain residual coefficients. These coefficients are transmitted to the decoder after quantization on a TU basis.
    - *Residual Quadtree*: After obtaining the residual block by prediction process based on PU splitting type, it is split into multiple TUs according to a quadtree structure. For each TU, an integer transform is applied. The tree is called transform tree or residual quadtree (RQT) since the residual block is partitioned by a quadtree structure and a transform is applied to each leaf node of the quadtree.

Table 2: Simplified Form of Transform Tree Syntax Table

transform−tree(trafoDepth, blkIdx) {
no−residual−data−flag
if(!no−residuual−data−flag) {
split−transform−flag[x0][y0][trafoDepth]
if(split−transform−flag[x0][y0][trafoDepth]) {
transform −tree(trafoDepth+1, 0)
transform −tree(trafoDepth+1, 1)
transform −tree(trafoDepth+1, 2)
transform −tree(trafoDepth+1, 3)
} else {
transform −unit(trafoDepth)
}
}
}

Similar to the coding tree, which is represented by a series of split−coding−unit−flag, RQT is also structured by successive signaling of the syntax element split−transform−flag in a recursive manner. RQT can be classified into two cases having square shapes and nonsquare shapes, and they are denoted as square residual quadtree (SRQT) and nonsquare residual

quadtree (NSRQT), respectively. The NSRQT was adopted temporarily, but excluded in the final draft text specification. Table *2* shows a syntax table for the recursive structure of RQT

- *Nonsquare Partitioning*: SRQT is constructed when PU splitting type is square shape while NSRQT is utilized for rectangular and asymmetric shapes. For NSRQT, transform shape is horizontal when the choice of the partition mode is a horizontal type such as PART−2N×N, PART−2N×nU, and PART−2N×nD. The same rule is applied to the vertical type case such as PART−N×2N, PART−nL×2N, and PART−nR×2N. Although the syntax of SRQT and NSRQT is the same, as depicted in Table III, the shapes of TUs at each transform tree depth are defined differently for SRQT and NSRQT. Fig. 9 illustrates an example of transform tree and corresponding TU splitting. Fig. 9(a) represents transform tree. Fig. 9(b) shows TU splitting when the PU shape is square. Fig. 9(c) shows TU splitting when the PU shape is rectangular or asymmetric. Although they share the same transform tree, the actual TU splitting is different depending on the PU splitting type.
- *Transform across Boundary*: In HEVC, both the PU size and the TU size can reach the same size of the corresponding CU. This leads to the fact that the size of TU may be larger than that of the PU in the same CU, i.e., residuals from different PUs in the same CU can be transformed together. For example, when the TU size is equal to the CU size, the transform is applied to the residual block covering the whole CU regardless of the PU splitting type. Note that this case exists only for inter-coded CU since the prediction is always coupled with the TU splitting for intra coded CU.
- *Maximum Depth of Transform Tree*: The maximum depth of transform tree is closely related to the encoding complexity. To provide the flexibility on this feature, HEVC specifies two syntax elements in the SPS which control the maximum depth of transform tree for intra coded CU and inter coded CU, respectively. The case when the maximum depth of transform tree is equal to 1 is denoted as implicit TU splitting since there

is no need to transmit any information on whether the TU is split. In this case, the transform size is automatically adjusted to be fit inside the PU rather than allowing transform across the boundary. The coding efficiency loss of implicit TU partitioning is about from 0.7% to 1% compared to the cases RQT depth is equal to 2.

Chapter 2. Fractional Motion Estimation in HEVC and Related Works on Complexity Reduction

1. Motion Estimation

- One of the most important coding tools used in the HEVC is the inter-frames prediction, where is located the Motion Estimation (ME) process. The ME explores the temporal redundancy from the previously encoded frames, called reference frames, to encode the current one. With this method, it is possible to reduce the amount of data necessary to represent each frame since it is possible to transmit and store only the difference between the reference frame and the current frame, and a motion vector.
- In ME, the picture to be coded is first divided into blocks, and for each block, an encoder searches reference pictures to find the best matching block. The best matching block is called the prediction of the corresponding block and the difference between the original and the prediction signal is coded by various means, such as transform coding, and transmitted to a decoder. The relative position of the prediction with respect to the original block is called a motion vector and it is transmitted to the decoder along with the residual signal. The true displacements of moving objects between pictures are continuous and do not follow the sampling grid of the digitized video sequence. Hence, by utilizing fractional accuracy for motion vectors instead of integer accuracy, the residual error is decreased and coding efficiency of video coders is increased. If a motion vector has a fractional value, the reference block needs to be interpolated accordingly. The interpolation filter used in video coding standards are carefully designed taking into account many factors, such as coding efficiency, implementation complexity, and visual quality.
- In HEVC, the ME process is divided into two steps: integer motion Estimation (IME) and Fractional Motion Estimation (FME). FME is a refinement process with fractional pixel accuracy level of IME.

2. Fractional Motion Estimation

- FME by increasing the precision of motion vectors enhances the compression performances of a video encoder but introduces an extra computation cost. FME process is divided into two steps: Interpolation and Sum of Absolute Transformed Difference calculation.

2.1 Interpolation

- As in H.264/AVC, HEVC standard supports motion vectors with quarter-pel accuracy. Compared to H.264/AVC, H.265/HEVC includes various modifications to the interpolation filter design. During the development of the H.265/HEVC standard, several techniques were considered, including switched interpolation filter with offset (SIFO), maximum order of interpolation with minimal support (MOMS), one-dimensional directional interpolation filter (DIF), and DCT-based interpolation filter (DCT-IF). The latest design of the H.265/HEVC interpolation filter is based on the simplified form of the DCT-IF with the addition of the high-accuracy motion compensation processing. These modifications yield an average 4.0% bitrate reduction over the H.264/AVC interpolation filter for luma and 11.3% bitrate reduction for chroma components. The coding efficiency gains become very significant for some sequences and can reach a measured maximum of 21.7%.
- H.264/AVC supports motion vectors with quarter-pel accuracy for the luma component and one-eighth pel accuracy for chroma components for video in the 4:2:0 color format. Although some video sequences may benefit from higher motion vector accuracy, it was found that quarter-pel accuracy provides the best trade-off between prediction accuracy and signaling overhead. Fig. 10 (a) denotes the fractional pel positions for the luma interpolation process of H.264/AVC. To minimize the number of filtering operations, H.264/AVC uses various combinations of separable one-dimensional filters according to the fractional sample position. For example, if one of the motion vector components is fractional but another is an integer, then interpolation is applied along only one direction (vertical or horizontal). If both motion vector components are fractional, a horizontal (vertical) interpolation filtering is done followed by vertical (horizontal) filtering while the intermediate results are stored in a buffer. The

samples at half-pel positions and are derived by applying a 6-tap filter as shown in (1) and (2). The samples at half-pel positions are computed similarly but from the non-rounded intermediate half-pel samples rather than the integer-pel samples as shown in (3) and (4).

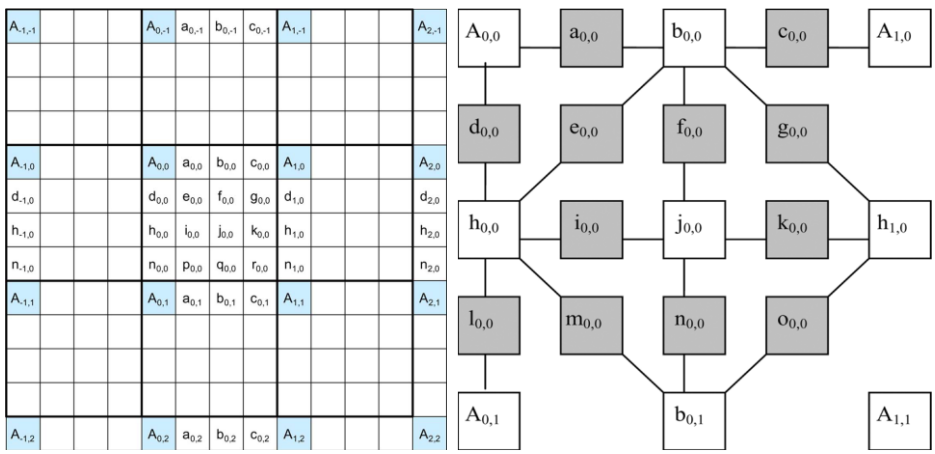


Fig. 10: (a): Fractional positions in Luma motion compensation with 1/4 pel accuracy. (b): Quarter-pel interpolation in H.264/AVC.

- The samples at quarter-pel positions are calculated by averaging the two nearest samples located at integer-pel and half-pel. positions. This is illustrated in Fig. 10 (b), where each quarter-pel position is connected to two samples at half-pel or integer-pel positions indicating the corresponding samples used in the averaging process as shown in (5), and (6).

a. Issues in Interpolation Process of H.264/AVC

- There are several issues with the interpolation process of H.264 as following:
    - **Number of filter coefficients**: a six-tap filter is used for half-pel positions of luma samples and a bi-linear filter for eighth-pel positions of chroma samples may not be sufficient for video sequences acquired with modern recording devices, which typically contain more high-frequency information than older video sequences. In addition, since the optimality of the interpolation filter is related to all other parts of the video compression system, the interpolation filter should be re-designed according to other parts of the H.265/HEVC standard.

- **Cascaded process for quarter-pel positions**: samples at quarter-pel positions are generated by averaging two neighboring samples. This cascaded process introduces an intermediate rounding step. This may introduce undesirable latency and accuracy losses.
- **Inconsistent averaging across quarter-pel positions**: samples at quarter-pel positions are derived differently according to their fractional positions.
- **Loss of accuracy from cascaded rounding operations**: the interpolation filter defined in H.264/AVC has a large number of intermediate rounding operations. The number of rounding operations can go up to 7 when specific quarter-pel positions are used with bi-directional prediction. Every rounding operation introduces an undesirable rounding error that accumulates over frames. The number of rounding operations should thus be minimized.

b. Interpolation Filter Design of HEVC

- To overcome the above issues, H.265/HEVC introduces several new features including redesigned interpolation filters for luma and chroma as well as a high-accuracy motion compensation process for uni- and bi-directional prediction which is mostly free from rounding errors. The key differences between H.264/AVC and H.265/HEVC interpolation can be summarized as:
    - **Re-designed luma and chroma interpolation filter**: to improve the filter response in the high-frequency range, luma and chroma interpolation filters are re-designed. The luma interpolation process uses a symmetric 8-tap filter for half-pel positions and an asymmetric 7-tap filter for quarter-pel positions to minimize the additional complexity of the motion compensation process. For chroma samples, a 4-tap filter is introduced.
    - **Non-cascaded process for quarter-pel positions**: rather than averaging two neighboring samples, H.265/HEVC directly derives quarter-pel samples by applying two one-dimensional filters similar to the half-pel center position in H.264/AVC. Since it is consistent with all quarter-pel positions, the

inconsistency issues for different quarter-pel positions in H.264/AVC no longer exist in H/265/HEVC.

- **High-accuracy motion compensation operation**: in H.265/HEVC, intermediate values used in interpolation are kept at a higher accuracy. In addition, the rounding of two prediction blocks used in bi-directional prediction is delayed and merged with the rounding in the bi-directional averaging process. It should be noted that the H.265/HEVC interpolation process guarantees that no 16-bit overflow occurs at any intermediate stage by controlling the accuracy according to the source bit depth.

- For fractional positions a, b and c, horizontal 1D filter is used. For fractional positions d, h and n, vertical 1D filter is used.For remaining positions, first horizontal 1D filter is applied for extended block and then vertical 1D filter is used. Half-pixel vertical and horizontal interpolation are illustrated as in Fig. 11, and quarter pixel vertical and horizontal interpolation are illustrated as in Fig. 12.
- The interpolation filter is applied in motion compensation for fractional position values generation. Current motion vector accuracy for luma components in HEVC is still quarter-pel, so 15 fractional-pel pixels will be interpolated as showed in Fig. 10 (a). In the HEVC, three types of 8-tap filters are adopted as shown in equation (7), (8), and (9). According to the fractional position to be predicted, one of the three filters is applied for.

$$b_{0,0} = (A_{-2,0} - 5A_{-1,0} + 20A_{0,0} + 20A_{1,0} - 5A_{2,0} + A_{3,0} + 16) \gg 5 \quad (1)$$

$$h_{0,0} = (A_{0,-2} - 5A_{0,-1} + 20A_{0,0} + 20A_{0,1} - 5A_{0,2} + A_{0,3} + 16) \gg 5 \quad (2)$$

$$h'_n = A_{n,-2} - 5A_{n,-1} + 20A_{n,0} + 20A_{n,1} - 5A_{n,2} + A_{n,3} \quad (3)$$

$$j_{0,0} = h'_{-2} - 5h'_{-1} + 20h'_0 + 20h'_1 - 5h'_2 + h'_3 + 512) \gg 10 \quad (4)$$

$$a_{0,0} = (A_{0,0} + b_{0,0} + 1) \gg 1 \quad (5)$$

$$f_{0,0} = (b_{0,0} + j_{0,0} + 1) \gg 1 \quad (6)$$

$$a_{0,0} = (-A_{-3,0} + 4A_{-2,0} - 10A_{-1,0} + 58A_{0,0} + 17A_{1,0} - 5A_{2,0} + A_{3,0} + + 32) \gg 6 \quad (7)$$

$$b_{0,0} = (-A_{-3,0} + 4A_{-2,0} - 11A_{-1,0} + 40A_{0,0} + 40A_{1,0} - 11A_{2,0} + +4A_{3,0} - A_{4,0} + 64) \gg 7 \quad (8)$$

$$c_{0,0} = (A_{-2,0} - 5A_{-1,0} + 17A_{0,0} + 58A_{1,0} - 10A_{2,0} + 4A_{3,0} - A_{4,0} + +32) \gg 6 \quad (9)$$

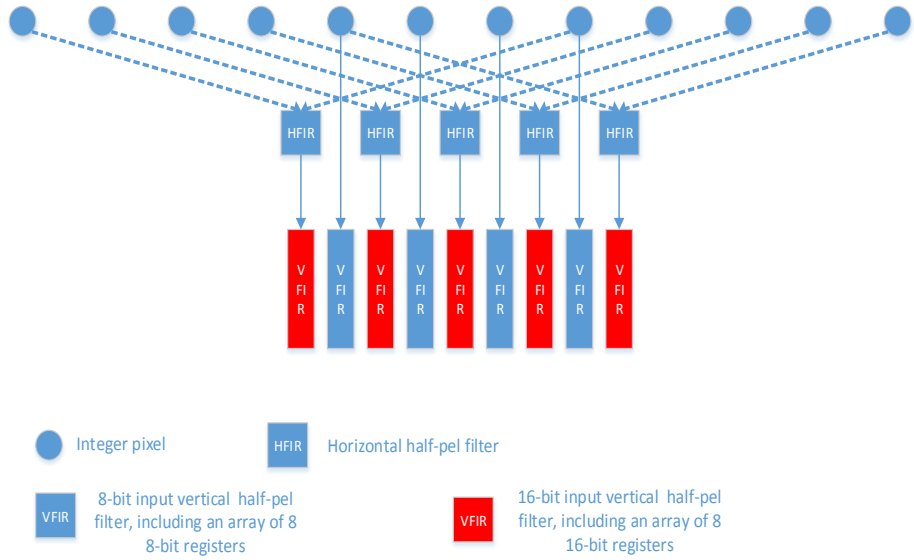


Fig. 11: Half-pixel horizontal and vertical interpolation

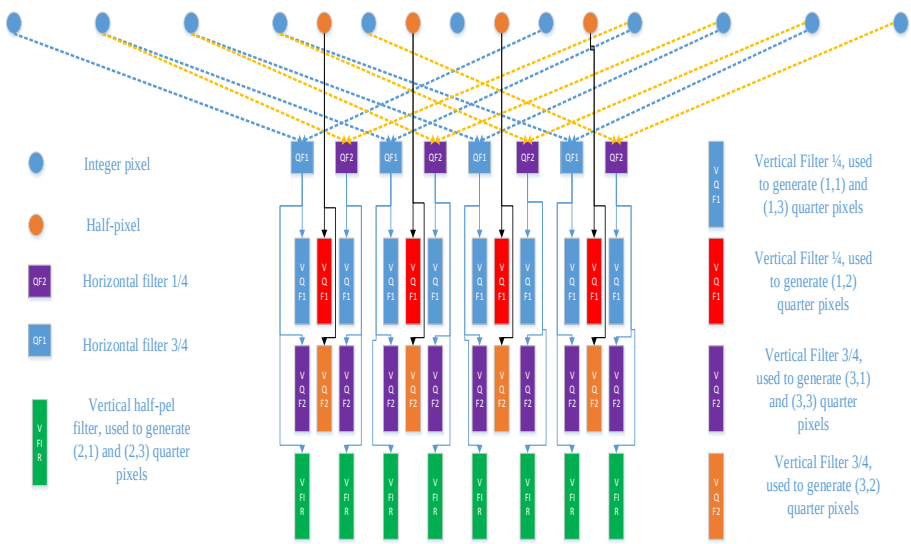


Fig. 12: Quarter pixel horizontal and vertical interpolation

2.2 Sum of Absolute Transformed Difference Calculation

- The latest video coding standard HEVC coder utilizes several advanced coding techniques to attain significantly higher compression ratio than the previous video coding standards. Among these, the rate-distortion optimization (RDO), which is the procedure conducted to select the best coding mode from all possible modes in both intra and inter-prediction, is considered to be one of the most important factors contributing to the success of HEVC in terms of compression ratio and visual quality. Nevertheless, this technique increases computational complexity remarkably. To lower the computation burden, the HEVC reference software provides a simplified way to estimate the rate-distortion cost with the prediction error and a simple bit cost estimate for a prediction mode, instead of obtaining the exact value by going through the whole encoding/decoding processes. It is illustrated as (10):

  $$J_{motion} = D_{motion} + \lambda_{motion} R_{motion} \quad (10)$$

  where λ_{motion} is the Lagrangian multiplier, D_{motion} is an error measure between the candidate macroblock taken from the reference frame(s) and the current macroblock and R_{motion} stands for the number of bits required to encode the difference between the motion vector(s) and its prediction from the neighboring macroblocks (differential coding). A similar function to the equation (10) is used to decide the optimal block size for motion estimation.
- Two distortion metrics are suggested to measure the prediction error; one being the sum of absolute differences (SAD), and the other sum of absolute Hadamard-transformed differences (SATD). In particular, for any given block of pixels, the SAD between the current macroblock and the reference candidate macroblock is computed using the following equation (11):

  $$SAD = \sum_{ij} |C_{ij} - R_{ij}| \quad (11)$$

  where C_{ij} is a pixel of the current macroblock and R_{ij} is a pixel of the reference candidate macroblock.
- The Lagrangian cost can also be minimized in the frequency domain, in a very similar manner to the pixel domain. As mentioned above, SATD can be used in equation (10) instead of SAD. Central to the calculation of SATD is the 4x4 Hadamard transform

which is an approximation to the 4x4 DCT transform. The transform matrix used is shown in equation (12) below (not normalized):

$$H = \begin{array}{|c|c|c|c|}\hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & -1 & -1 \\ \hline 1 & -1 & -1 & 1 \\ \hline 1 & -1 & 1 & -1 \\ \hline \end{array} \quad (12)$$

Since H is a symmetric matrix, it is equal to its own transpose. By using this matrix, the (SATD) is computed using equation (13) below:

$$SATD = \left(\sum_{ij} |H * (C_{ij} - R_{ij}| H^T\right)/2 \quad (13)$$

where C_{ij} is a pixel of the current macroblock and R_{ij} is a pixel of the reference candidate macroblock.

- The sum of absolute transformed differences (SATD) is a widely used block matching criteria used in fractional motion estimation for video compression. Especially, it is used for fractional motion estimation in HEVC

2.3 Fractional Motion Estimation Procedure

- In HEVC, the algorithm of fractional pixel interpolation for motion compensation (MC) is defined in the coding standard. However, how to produce the fractional MV, or FME procedure, including the interpolation scheme on the reference frame and the FME searching algorithm, can be decided by the designer. Therefore, there are several FME procedures have been introduced, which can be roughly classified into two groups:
    - Two-iteration FME
    - Single-iteration FME

a. Two-iteration FME

- After the integer pixel motion search finds the best match, the values at half-pixel positions around the best match are interpolated by applying a one-dimensional 8-tap FIR filter horizontally and vertically. Then the SATD value of each half-pixel is calculated and compare to find the best match half-pixel. Then the values of the quarter-pixel positions are generated around the best match half-pixel by applying a one-dimensional 7-tap FIR filter horizontally and vertically. Fig. 13 illustrates an example of

interpolated fractional pixel positions of the two-iteration FME, where half-pixels are generated around the best integer pixel position and quarter pixels are generated around the best half pixel.

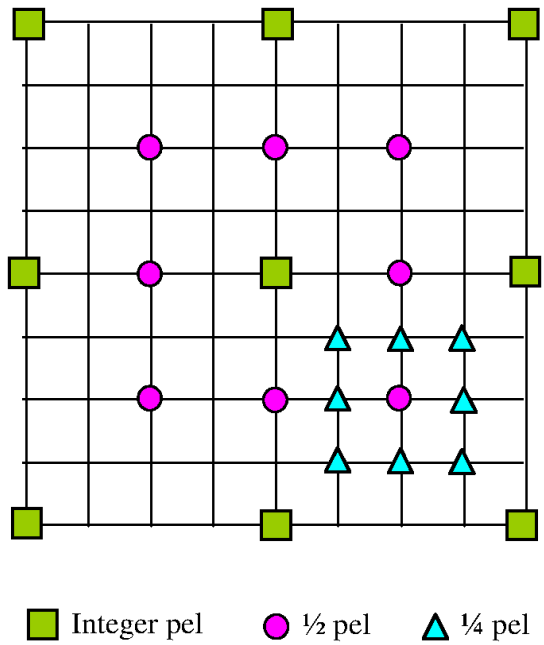


Fig. 13: Two-iteration FME

b. Single-iteration FME

- One of the drawbacks of two-iteration FME is that quarter-pixel interpolation and search can be processed only after half-pixel search is finished. This results in huge timing constraint for real-time application. Therefore, a number of designs have adopted the single-iteration FME scheme. In the single-iteration FME scheme, half pixels and quarter pixels are generated around the best matching integer pixel position at the same time. And then SATD of all fractional pixels or a certain number of fractional pixels around the best integer pixel are calculated. After getting all necessary SATD values, SATD comparison is processed to find the best matching pixel. Fig. 14 illustrates an example of interpolated fractional pixel positions of a single-iteration FME, where half-pixels and quarter pixels are generated around the best integer pixel.

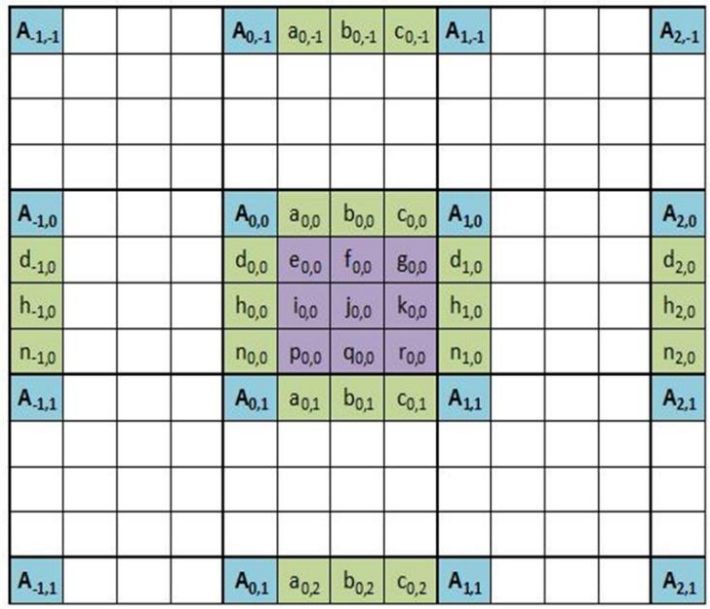


Fig. 14: Single-iteration FME

Chapter 3. Complexity Reduction for FME

1. Problem Statement and Previous Studies

1.1. Problem Statement

- HEVC employs the hierarchical quad-tree structure based on the coding tree unit (CTU), using the coding unit (CU), a prediction unit (PU), and transform unit (TU) as the basic processing unit of coding, prediction, and transform, respectively. This new structure can be adaptively adjusted between the large homogeneous region and highly textured region, which accounts for HEVC's high encoding efficiency compared to H.264. However, it comes with the price of about 40% encoding complexity increase [2].
- Inter mode prediction with motion estimation is the bottleneck of HEVC because of the abounding amount of computation, in which Motion Estimation (ME) which consists of integer motion estimation (IME) and fractional motion estimation (FME) is its main core. In HM, HEVC's reference software, motion estimation (ME) alone occupies up to 51.32 % of execution time, in which IME takes 18.17 % and FME takes around 32.16% [3]. This huge timing constraint imposed by FME is the result of complicated and time-consuming interpolation processes and that the two-step FMEs for half- and quarter-pixel precisions should be performed sequentially.
- There are a number of fast algorithms for IME have been introduced, which helped the complexity of IME reduced significantly. It results in the fact that FME remains to be more complicated and time-consuming than IME. The FME's enormous computational complexity can be explained by two following reasons:
    - *A large number of FME refinements processed*: In HEVC, a frame is divided into CTU, whose size is usually 64x64 pixels. One 64x64 CTU consists of 85 CUs including one 64x64 CU at depth 0, four 32x32 CUs at depth 1, 16 16x16 CUs at depth 2, and 64 8x8 CUs at depth 3. Each CU can be partitioned into PUs according to a set of 8 allowable partition types. An HEVC encoder processes FME refinement for all possible PUs with usually 4 reference frames before deciding the best

configuration for a CTU. As a result, typically in HEVC's reference software, HM, for one CTU, it has to process 2,372 FME refinements, which consumes a lot of computational resources.

- *A complicated and redundant interpolation process*: Conventionally, FME refinement, which consists of interpolation and sum of absolute transformed difference (SATD), is processed for every PU in 4 reference frames. As a result, for a 64x64 CTU, in order to process fractional pixel refinement, FME needs to interpolate 6,232,900 fractional pixels. In addition, In HEVC, fractional pixels which consist half fractional pixels and quarter fractional pixels, are interpolated by 8-tap filters and 7-tap filters instead of 6-tap filters and bilinear filters as previous standards. As a result, interpolation process in FME imposes an extreme computational burden on HEVC encoders.

- For the above reasons, FME is the computational bottleneck of real-time HEVC encoder. Therefore, reducing the computational complexity of FME is a very critical task in order to assure real-time operation. However, there are not many efficient algorithms which help to reduce the complexity of FME in HEVC has been introduced so far. That is why an efficient algorithm which effectively reduce the complexity of FME without significantly degrading the encoding performance could be extremely valuable for HEVC encoders.

1.2. Previous Studies

- There are a number of fast algorithms targeting FME have been introduced to overcome the above problems. These algorithms can be roughly classified into two categories: FME procedure modification and advanced PU partitions decision. The first category, [4]-[6] tried to reduce timing constraint and computational complexity of FME by improving interpolation process, reducing searching point or exploiting single iterative refinement method. However, this kind of algorithms do not exploit the relationship of IME and FME and have to pay the trade-off of coding performance decrease to obtain complexity reduction. Moreover, none of the so far proposed algorithms attempt to reduce the redundant interpolation caused by the

similarity in motion among neighboring PUs. Therefore, this paper proposes an efficient range-based algorithm that reduces a large amount of redundant interpolation calculation by avoiding repeatedly interpolating overlapped regions caused by the motional similarity among neighboring PUs. In the second category, instead of doing FME for all PU partitions, based on IME result to filter unnecessary skippable partitions. This is based on the fact that FME is the refinement process of IME result, therefore according to IME result, there are redundant partitions can be skipped doing FME refinement without significantly affecting the encoding efficiency. Several research [7], [8] were successfully proposed to H.264 but the direct application of these algorithms to HEVC results in significantly encoding efficiency degrade due to the difference of encoding structure between HEVC and H.264 [3]. The algorithm in [7] is modified and applied to HEVC [3], however for simplicity, it does not take into account asymmetric partition which is one of the key factors accounting for coding efficiency improvement in HEVC compared to H. 264. In addition, it also does not take into account the variation of temporal correlation among frames and treats all the reference frames equally which can cause extra computational complexity. Therefore, this paper proposes an efficient algorithm that takes into account all asymmetric partitions as well as exploits the variation of temporal correlation between the current frame and different reference frames to predetermine PU partition type for FME to reduce the complexity without significantly degrading encoding performance. In addition, the proposed algorithm also takes advantages of IME result of CUs at CTU level to predetermine CU depth for FME in order to further reduce complexity.

2. Proposed Algorithms

- As discussed in the previous section, FME in HEVC is very complicated because of the two following reasons:
    - A large number of FME refinements processed
    - A complicated and redundant interpolation process
- Therefore, in order to reduce FME's complexity, it is necessary to tackle one of or both the reasons. In this work, two following efficient algorithms which tackle each one of the two reasons of FME's complexity are proposed:
    - Advanced decision of PU partitions and CU depths for Fractional Motion Estimation in HEVC
    - A Reduction of the Interpolation Redundancy for Fractional Motion Estimation in HEVC
- The first algorithm tackles the first reason to reduce FME's complexity b reducing the number of FME refinements while the second algorithm aims to solve the second reason to reduce FME's complexity by reducing interpolation redundancy in FME. The two algorithms are discussed more in detail in the following sections.

2.1. Advanced Decision of PU Partitions and CU Depths for Fractional Motion Estimation in HEVC

a. Reference Frame Selection Analysis

- As mentioned in the problem the statement section, to the best of the author's knowledge, all previous works in this category do not take into account the variation in the temporal correlation of current frame and different reference frames. All reference frames are treated equally in the manner of choosing some PUs or CU depths having smallest IME costs, which are the RD costs after IME, and process FME for these modes for all reference frames. This causes possible encoding performance decrease and unnecessary complexity increase because usually reference frames with higher temporal correlation with current frame have a higher probability to be chosen as a final reference frame by the best mode after doing motion estimation. Table I shows reference frames' probability of being chosen as a final reference frame by the best mode. In which, the first column represents the sequence simulated and the second, third and fourth

column show the probability to be chosen as the final reference frame of reference frame 0, 1, 2, and 3 respectively.

Table 3: Reference frames' probability of being chosen as a final reference frame by the best mode

	Ref. Frame 0	Ref. Frame 1	Ref. Frame 2	Ref. Frame 3
C1 Keiba	65.38%	23.34%	7.67%	3.60%
C2 BQMail	87.17%	8.11%	2.66%	2.06%
C3 BasketballDrill	79.88%	12.82%	3.85%	3.45%
C4 Flowercase	87.26%	7.83%	2.79%	2.13%
C5 PartyScene	79.15%	11.04%	5.23%	4.57%
C6 RaceHorses	77.65%	14.35%	5.31%	2.67%
Average	79.73%	12.47%	4.66%	3.14%

- As shown in Table 3, almost 80% of the best modes choose reference frame 0 which is the temporally nearest frame to current frame as a final reference frame, and if referent frame 0, reference frame 1, and reference frame 2 are taken into account, roughly 97% of the best modes choose one of these three reference frames as a final reference frame. Based on this observation, this paper proposes an efficient algorithm that exploits temporal correlation of current frame and reference frames in order to further reduce complexity without significantly degrading encoding efficiency.

b. Advanced PU Partition Decision

❖ The Idea

- In HEVC, in order to find the best specification for a CTU, motion estimation will be processed for all PUs in 4 reference frames according to 8 PU partition types of a CU, and recursively processed for all CUs in a CTU. Then the results will be compared to determine the best reference frame for every PU, the optimal PU partition type for every CU, and optimal CU splitting in a CTU. This means that for a CU, before FME, if it can predetermine the PU partition types which are less likely to be chosen as the optimal PU partition type, and discard them from doing FME refinement, we can significantly reduce the number of FME refinement. The proposed algorithm based on two following observations

- Since FME is just a refinement of IME. PU partition types having smallest cost after IME are more likely to have the smallest cost after FME as well
- Temporally closer reference frames' probabilities to be chosen as the best reference frame is higher than the distant ones.

- Previous studies just considered the first observation to predetermine PU partition types for CUs and ignore the second observation. The proposed algorithm exploits both of them to effectively reduce computational complexity with insignificant performance degrade. The proposed algorithm is processed as follow. For a given CU, the PU partition types for FME is selected by comparing the IME costs of the corresponding PU partitions. To this end, the IME costs of all partition types are calculated, and then the PU partition types are sorted according to the IME cost in each reference frame. In order to exploit the first observation, it discards a certain number of PU partition types having largest IME costs from doing FME. In order to exploit the second observation, it adaptively chooses a smaller number of discarded PU partition types for temporally closer reference frames, and a larger number discarded PU partition types for distant ones. By this, it can get maximum complexity reduction without significantly degrading the encoding performance.

❖ The algorithm

- In a CU, for each reference frame, instead of processing FME for all partition types, the algorithm independently choose a certain number of PU partitions having smallest IME cost considering in that reference frame only to do FME. The number of modes selected to do FME is also different and independent for different reference frames. The optimal number of PU partition for FME process for each frame are determined by experiment results presented in section IV. Fig. 15 illustrates the proposed algorithm to predetermine reduced PU partition modes for FME. The algorithm can be divided into two main tasks: IME cost comparison and skippable PU partition modes filter.
- In the first task, IME cost comparison, first, get IME costs of all 7 partition types (3 if it is 8x8 CU) for each reference frame, for partition type 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N, the IME cost is the sum of two partition's IME cost. Then for each reference frame, sort all the partition types in increasing order of IME cost.

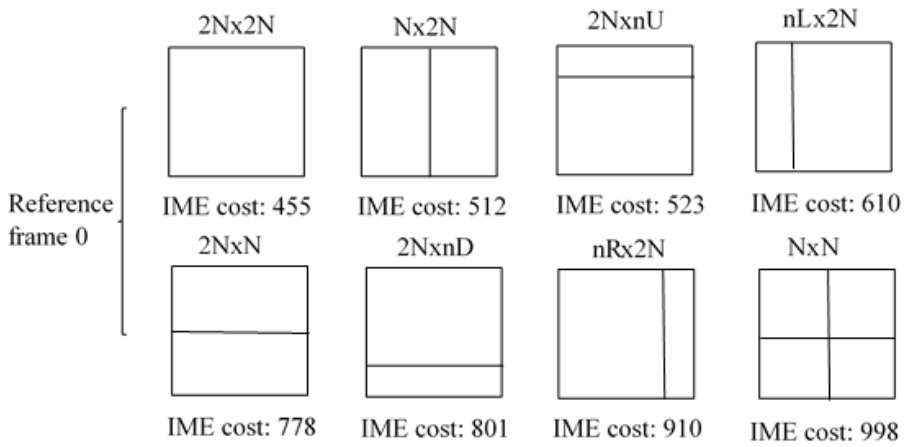


Fig. 15: PU partition modes in IME cost increasing order

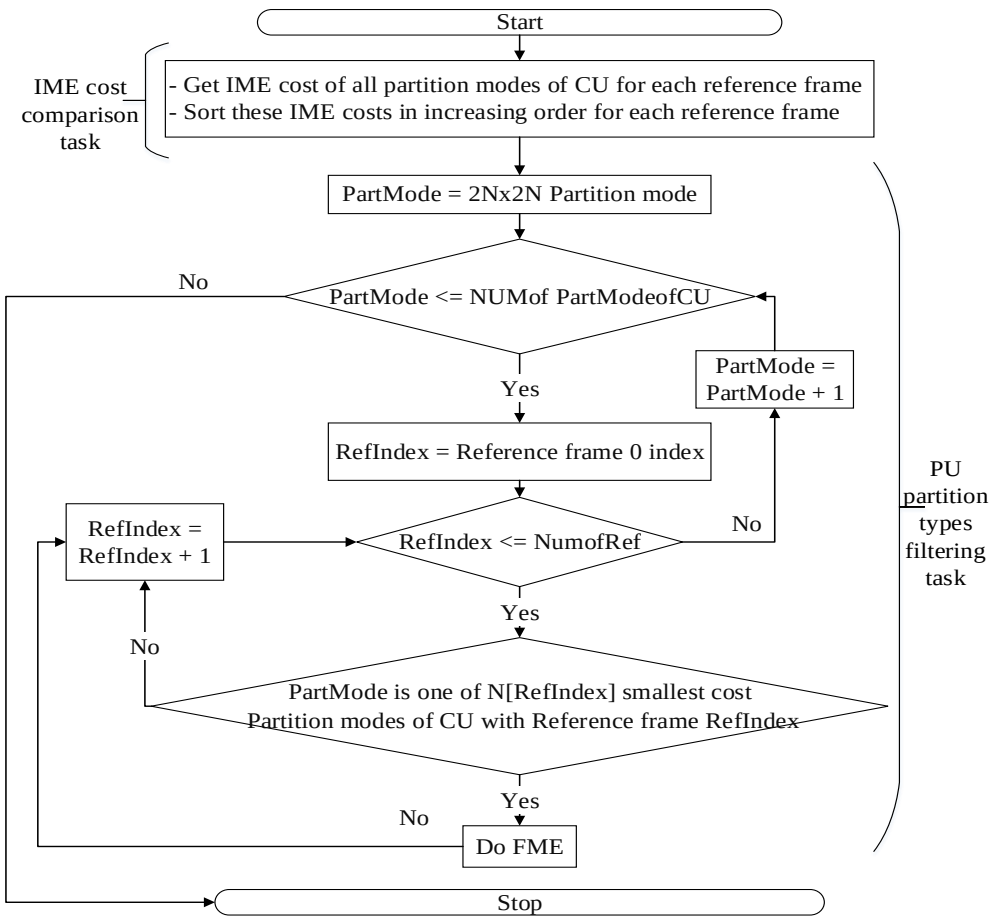


Fig. 16: The flow-chart of the advanced PU partition decision algorithm

- In the second task, skippable PU partition types filtering, based on the sorted list of all PU partition types of each reference frame, determine which PU partition types are selected to do FME and which ones are filtered from doing FME independently for each reference frame. In order to exploit the fact that the more temporally distant frames are less likely to be chosen as final reference frame by best mode, the proposed algorithm adaptively adjusts the number of PU partition types selected to do FME different for each frame. For simplicity, N{RefIndex} presents the number of PU partition types selected to do FME for reference frame RefIndex, where RefIndex is reference index whose value is 0, 1, 2 and 3. In this task, for each PU partition type, for each reference frame RefIndex, check whether the PU partition types is one of the N{RefIndex} smallest cost PU partition types of reference frame RefIndex or not. If yes, do FME for this PU partition type, otherwise filter it from doing FME.

c. Advanced CU Depth Decision

❖ The idea

- In HEVC, one CTU consists of 85 CUs including one 64x64 CU at depth 0, four 32x32 CUs at depth 1, 16 16x16 CUs at depth 2, and 64 8x8 CUs at depth 3. Instead of doing FME for every CU depth, based on IME result.
- Because FME is a refinement process of IME, best CU depths after IME and final best CU depths are highly correlated.
- Table 4 shows the probability of the correlation of best IME depth which is the best CU depth decided based on IME cost and final best CU depth. In which, the first row represents final best CU depth and the first column represents best CU depth after IME. The remaining rows and columns represent the probability of that best IME depth is d1 and the final best CU depth is d2, in which d1 and d2 get the value of 0, 1, 2 and 3. The result shows that more than 50% of best IME depths are also final best CU depths. And if best IME depth and 2 depths surrounding it are taken into account, roughly more than 91% of final best CU depths belong to this group of three depths. In which, in case best IME depth is depth 0 or depth 3, two surrounding depths are CU depths which are one depth and two depths far away from best IME depth. And if best IME depth is depth 1 or depth 2, two surrounding depths are CU depths which is one depth larger and one

depth smaller than best IME depth. Based on this observation, an efficient algorithm predetermining CU depth for FME is proposed.

Table 4: Probability of correlation of best CU depth after IME and final best CU depth

Final best CU depth / Best IME depth	Depth 0	Depth 1	Depth 2	Depth 3
Depth 0	73.20%	18.46%	5.70%	2.64%
Depth 1	8.19%	65.87%	20.23%	5.71%
Depth 2	1.21%	13.52%	50.67%	34.60%
Depth 3	0.02%	1.00%	9.04%	89.94%

❖ The algorithm

- To reduce FME complexity, the CU depth for FME can be selected by comparing the IME costs. This is possible because FME is a refinement process of IME so that the best CU depths after IME and the final best CU depths are highly correlated. To this end, the IME costs of all CUs in a CTU are compared. Then, the best IME CU depths are determined. From the selected depth, the CU depth(s) for FME are finally chosen in four different options: the best IME depth only (best_only), the best IME depth plus one more depth along one direction (best_plus_one), the best IME depth plus one more depth along two directions (best_plus_minus), and best IME depth plus two additional depths (best_plus_two). In the case of depth 1 chosen from IME, FME is performed only for depth 1 in best_only option, depths 1 and 2 in best_plus one option, depths 0, 1, and 2 in both best_plus_minus and best_plus_two option. The remaining depths are kept with the IME cost and used for CU depth decision in a later stage.

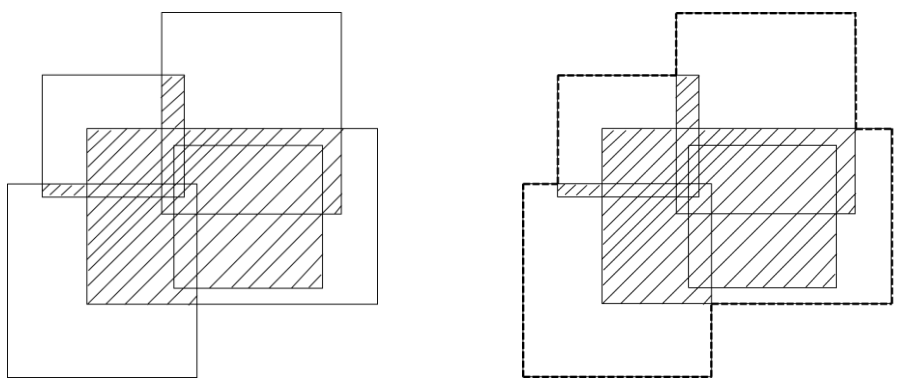
d. Combination of Advanced PU Partition and Cu depth Decision

- This combined algorithm predetermines both skippable CU depths and PU partition type for FME in order to get maximum complexity reduction with an acceptable BD-BR increase. First, it filters skippable CUs among 85 CUs in a CTU using the process of advanced CU depth decision algorithm. Then, for CUs which are determined to do FME, it discards unnecessary PU partitions from doing FME by using the process of skippable PU partition decision

2.2. Range-based interpolation algorithm

a. Motional Similarity among Neighboring PUs

- Conventionally, each PU is processed FME separately. This means that for a CTU, interpolated sub-pixels are generated separately for each PU. However, since spatially-neighboring PUs in one CTU usually have similar motion, as a result, after IME, predicted blocks of PUs have overlapped regions as illustrated in the Fig. 17 (a) below. As mentioned previously, in conventional FME algorithms, because each PU is processed FME separately, sub-pixels in overlapped regions are repeatedly calculated for different PUs which causes a great amount of redundant computation. Based on this observation, a new FME algorithm called range-based algorithm is proposed to reduce redundant interpolation computation caused by repeatedly interpolating pixels in overlapped regions.



(a) (b)

Fig. 17: (a) Predicted PUs with overlapped regions which are slashed. (b) The dot-line union of predicted PUs

b. Range-based Interpolation Algorithm

- The idea of the range-base algorithm is instead of processing each PU separately, finding the union region of PUs as illustrated in Fig. 17 (b) and then do interpolation for the whole united region. In order to find the union region of PUs, the algorithm partition the whole union into non-overlapped rectangles which are defined by key points as illustrated in Fig. 18 (a), where a rectangle (rec) contains information on the right and left x-coordinates (rec.left, rec.right) and top and bottom y-coordinates (rec.ytop, rec.ybottom) and can be defined by

two vertices X0, Y0 and X1, Y1 as illustrated in Fig. 18 (b). The algorithm process is illustrated in Fig. 19, in which N_y is the number of y-coordinates obtain by ytop and ybottom of all rectangles, and N_{rec} is the number of all rectangles.

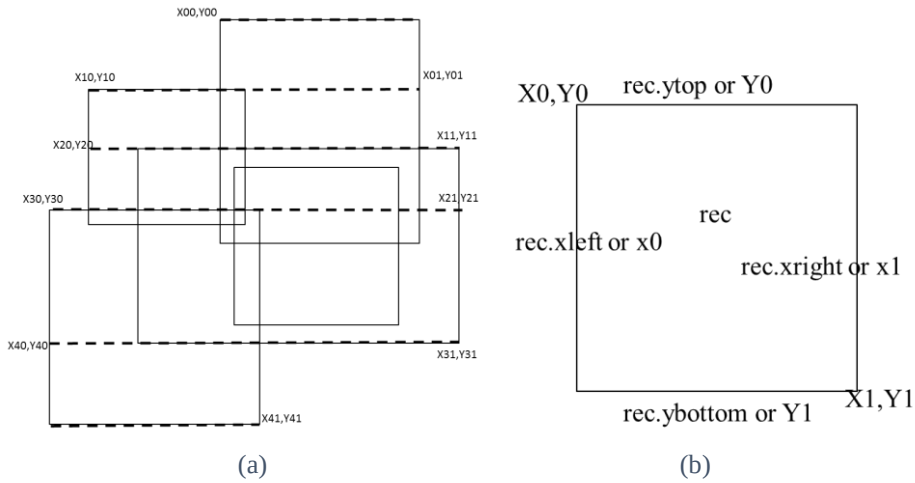


Fig. 18: (a) Defining the union of rectangles by key points. (b) Defining a rectangle

- In HEVC, to do FME for one CTU in one reference frame, 593 PUs need to be processed. If the range-based algorithm is applied for this level, which means $N_{rec} = 593$, the highest interpolation calculation reduction can be obtained while the complexity of the algorithm itself is also the highest. In order to limit the complexity of the algorithm, PUs are grouped into spatial groups and each group is represented by only one rectangle which is the smallest rectangle containing all PUs in that group. Then the range-based algorithm is applied for representing rectangles only. There are three levels for the algorithm. In level 1, the simplest, by considering 593 PUs in one group and representing it by one rectangle, then N_{rec} is 1. In level 2, 593 PUs are divided into five groups including four groups of 32x32 block and a one group of depth-0 PUs, where a group of 32x32 block includes all PUs from depth 1 to depth 3, which are spatially inside that 32x32 block, and the group of depth-0 PUs includes all PU of the depth 0 CU. Representing the five groups by five rectangles, and then the algorithm is applied for that 5 rectangles, which means Nrec=5.

Similarly, in level 3, dividing into 16x16 block level, there are 33 groups with 33 representing rectangles.

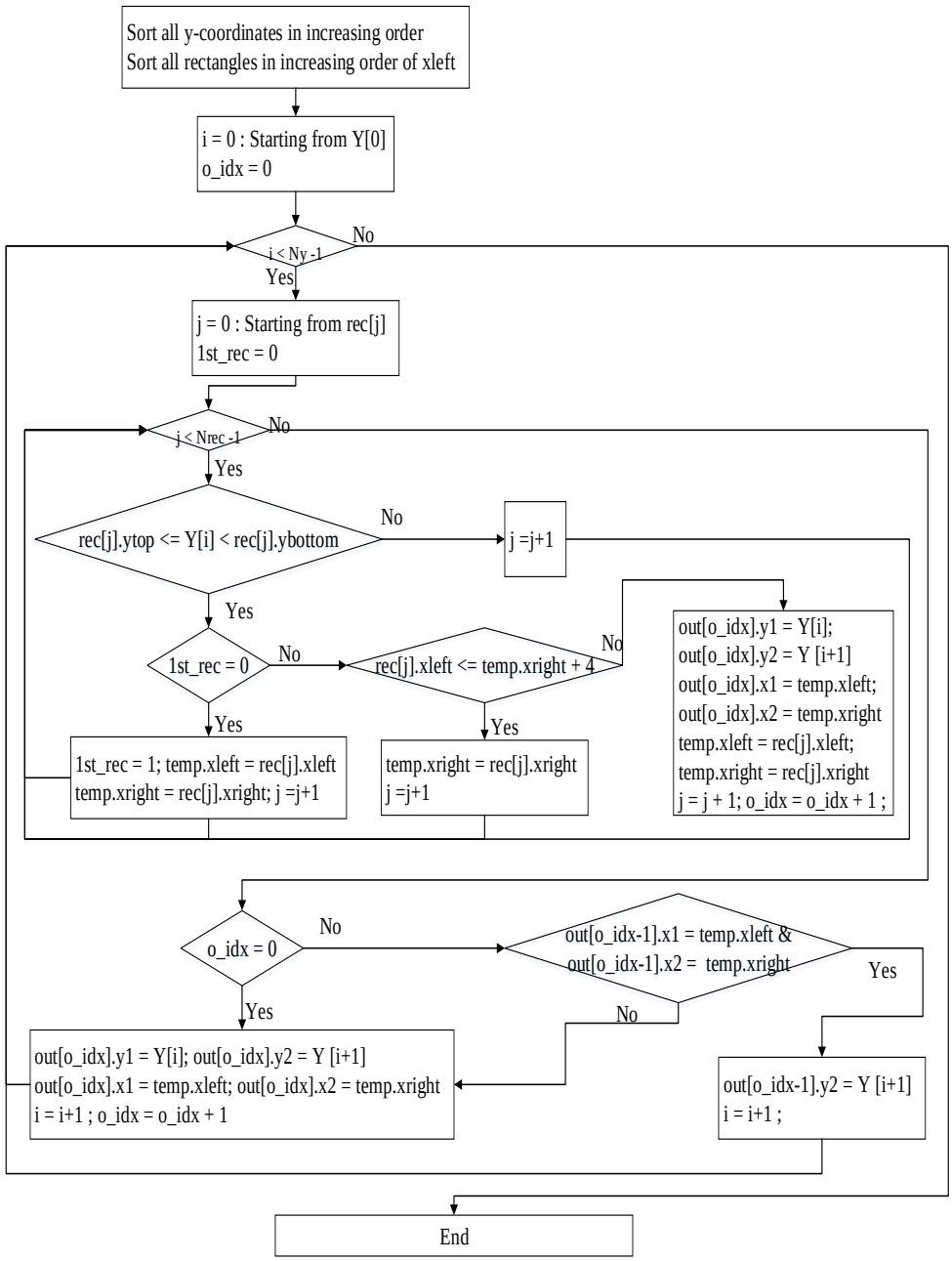


Fig. 19: Flow-chart of the range-based algorithm

Chapter 5. Experiment Results

- The proposed algorithms were integrated into the reference software, HM-13.0. Six of class C test sequences are encoded in Low-delay P with quantization parameters (QP) 22, 27, 32, 37. And all hardware implementations were implemented standard 130nm CMOS technology

1. Advanced Decision of PU Partitions and CU Depths for Fractional Motion Estimation in HEVC Algorithms

- The number of FME calculation for a 4x4 block is defined as complexity unit. Then the complexity of doing FME of a block can be calculated:

$$C_{block} = (block_{height} * block\_width)/16 \quad (13)$$

The total complexity is accumulated by all complexity of blocks selected to do FME.

1.1. Advanced Decision of PU Partitions

- In this algorithm, the number of PU partitions for FME is adaptive and not the same for all reference frame. It uses a higher number of PU partition for temporally close reference frames, which are more important and correlated with the current frame. In order to find optimal numbers of PU partition types selected for each reference frame, it is necessary to analyze how selecting a certain number of PU partition affects each reference frame. In order to do that, the advanced PU decision algorithm is run for each reference frame only, for example for reference frame 0 only, to get the statistical results of how BD-BR changes according to the number of PU partition selected for FME in each reference frame.

- The results are illustrated in Fig. 20, in which horizontal and vertical axes are complexity reduction and BD-BR increase respectively. Six markers illustrate 1, 2, 3, 4, 5, 6 and 7 partition types selected to do FME respectively. The results show that while complexity reduction is almost similar to all frames, BD-BR increase is most vulnerable to the PU partition type predetermination of reference frame 0, then reference frame 1. Reference frame 2 and 3 have less effect on BD-

BR increase compared to the other two. For each reference frame, the numbers which give the best tradeoff between complexity reduction and BD-BR increase are chosen as optimal numbers. In Fig. 2, the result shows that 4 and 5 are optimal numbers of PU partition types predetermined for FME of reference frame 0; 3 and 4 are optimal numbers of reference frame 1; 2 and 3 are optimal numbers of reference frame 2 and reference frame 3. Based on this result, the algorithm is simulated for different combinations of a number of PU partition types selected for each frame.

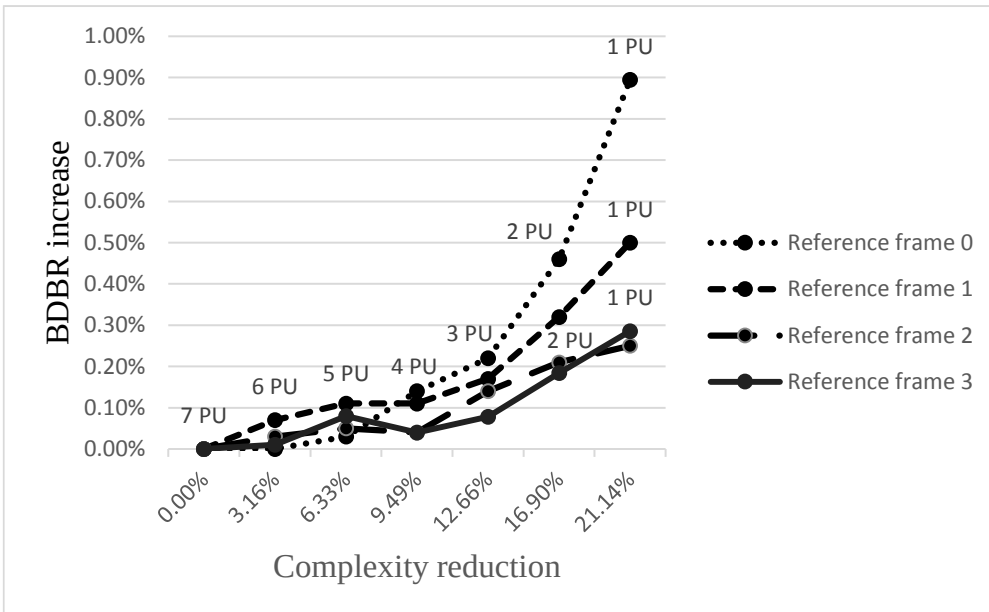


Fig. 20: PU partition decision algorithm for each reference frame alone

a. Results of fixed number of PU partition for all reference frames

- First, we run the experiments of algorithms with the number of PU partition selected to process FME in a CU is fixed the same for all reference frames. The experiment results are presented in Table 5 below, where No. of PU partition type selected to do FME refers to the number of PU partition type selected to do FME. The other two columns are BD-BR increase and complexity reduction.

Table 5: Experiment results of PU algorithms with number of PU partition selected to do FME the same for all reference frames

No. of PU partition type selected to do FME	BDBR increase	Complexity reduce
1	2.278%	83.286%
2	1.270%	66.572%
3	1.230%	49.858%
4	0.446%	37.394%
5	0.283%	24.929%
6	0.169%	12.465%
7	0.000%	0.000%

- This results will be used to compare with the advanced PU partition decision algorithm, in which the number of PU partition selected is different and adaptive for each reference frame.

b. Results of Advanced PU Partition Decision Algorithm

- The results is illustrated in Table 6 in which the first column represents the combination of all reference frame whose b-c-d-e PU selection means the algorithm predetermines b, c, d, and e number of PU partition types predetermined for FME for reference frame 0, reference frame 1, reference frame 2, and reference frame 3 respectively.
- Fig. 21 shows complexity reduction and BD-BR tradeoff of the proposed algorithm with an adaptive combination of the number of PU partitions and it with the number of PU partitions fixed the same for all reference frames. As illustrated in Fig. 3, the result shows that adaptive combination of the number of PU partitions predetermined for FME for each reference frame gives better performance compared with the number of PU partitions fixed the same for all reference frames. One of the best tradeoffs is the 5-3-2-2 PU selection combination which reduces 51.80% FME complexity with 0.60% BD-BR increase

Table 6: Experiment results of Advanced PU Partition Decision with different combinations of different PU partition selected for each reference frame

Combination of all reference frames	BD-BR increase	Complexity reduction
7-6-6-5 PU selection	-0.01%	12.37%
6-6-6-4 PU selection	0.14%	18.60%
5-4-3-3 PU selection	0.41%	40.40%
5-3-2-2 PU selection	0.60%	51.80%
4-3-2-2 PU selection	0.73%	54.97%
3-2-1-1 PU selection	1.24%	70.60%

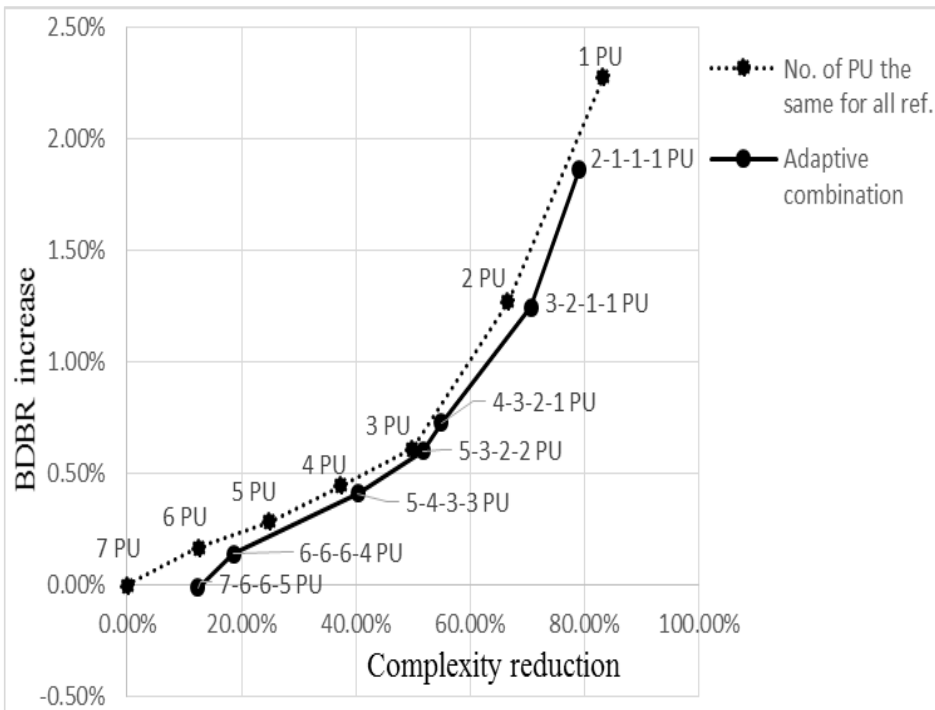


Fig. 21: Comparison of Advance PU partition decision algorithm and fixed number of PU partition algorithm

1.2. Advanced Decision of CU Partitions

- At CTU level, the proposed algorithm determines which CUs among 85 CUs are selected for FME. Table 7 shows the BD-BR increase and complexity reduction of the algorithm. The first column represents the CU depth selection option whereas the second and third columns represent BD-BR increase and complexity reduction, respectively. As shown in the table, the best trade-off is the best_plus_minus option, in which complexity reduction is 27.95% and BD-BR increase is 0.27%.

Table 7: Results of advanced CU depth decision

Option	BD-BR increase	Complexity reduction
best_only	2.94%	70.34%
best_plus_one	0.66%	45.56%
best_plus_minus_one	0.27%	27.95%
best_plus_two	0.13%	17.19%

1.3. Combination of Advanced PU Partition and CU Depth Decision

Table 8: Results of Advanced PU Partition and CU Depth Decision

Combination of PU partition type and CU depth predetermination	BD-BR increase	Complexity reduction
best_plus_minus and 5-3-2-2 PU	1.08%	67.47%
best_plus_minus and 4-3-2-2 PU	1.14%	69.94%
best_plus_minus and 3-2-1-1 PU	1.66%	80.64%
best_plus_two and 5-3-2-2 PU	0.95%	62.72%
best_plus_two and 4-3-2-2 PU	1.06%	65.69%
best_plus_two and 3-2-1-1 PU	1.53%	78.06%

- The two proposed algorithms: Advanced PU Partition Decision and Advanced CU Depth Decision are combined into one algorithm called Advance PU Partition and CU Depth Decision Algorithm with the options which give reasonable BD-BR and complexity tradeoff. Therefore, the combination of CU depth selection with best_plus_minus and best_plus_2 options and PU partition type with 5-3-2-2 PU, 4-3-2-2 PU, and 3-2-1-1 PU selections are evaluated. The

results are presented in Table 8, in which the first column represents the CU depth and PU partition type combination option, the second and third columns are BD-BR increase and complexity reduction respectively. The combination of best_plus_minus and 5-3-2-2 PU selection gives the most reasonable trade-off of 67.47% complexity reduction with 1.08% BD-BR increase.

1.4. Comparison with Other Similar Works

- There are some similar researches have been proposed previously [5], [9]. In [5], the algorithm uses PU Size-Dependent FME, in which it adopts interpolation free FME for depth 0/1 and full search for depth 2 and skips FME for depth 3. Interpolation free FME is an algorithm to do FME refinement without interpolation based on IME results and mathematical model. In [9], the authors proposes an algorithm that reduces the complexity by restricting the Prediction Units (PUs) - among a total of 24 sizes - to the 4 square-shaped size. Those two are the most recent works that reduce FME's complexity by reducing the number of FME refinements.
- Table 9 shows the comparison of Advanced PU Partition and CU Depth Decision with that two algorithms. It is very clear that the proposed algorithm performs much better than the other two algorithm

Table 9: Comparison of the proposed algorithm and other algorithms

Algorithm	BD-BR increase	Complexity reduction
Advanced PU partition and CU Depth Decision	1.66%	80.64%
[5]	2.7%	62.4%
[9]	4%	74%

2. Range-based Algorithm

- The algorithm is implemented in both HM software and hardware in a standard 130nm CMOS technology. The algorithm is implemented for three level:
    - Level 1: Only one rectangle as a union representing for all 593 PUs in a CTU
    - Level 2: 593 PUs are divided into five groups including four groups of 32x32 block and a one group of depth-0 PUs, representing the five groups by five rectangles. Applying the algorithm for five rectangles to find the union to reduce redundancy caused by overlapping
    - Level 3: 593 PUs are divided into 16x16 blocks, which means that there are 33 groups with 33 representing rectangles. Applying the algorithm for five rectangles to find the union to reduce redundancy caused by overlapping

2.1. Software Implementation

Table 10: Interpolation Reduction Percentage for Each Level

	Interpolation Reduction Percentage		
	Level 1	Level 2	Level 3
Seq. C1	68.98%	71.78%	74.87%
Seq. C2	74.94%	77.17%	91.27%
Seq. C3	80.25%	81.97%	89.65%
Seq. C4	85.06%	86.31%	95.70%
Seq. C5	89.40%	90.24%	91.73%
Seq. C6	88.79%	89.67%	82.70%
Average	84.92%	86.46%	87.65%

- The algorithm is integrated into HEVC reference software HM and run for each level. The results are illustrated in Table 10 below, in which, Seq. C1 -> Seq. C6 refer to the video test sequence in class C, from sequence 1 to sequence 6. Level 1, Level 2, Level 3 and Interpolation Reduction Percentage show the percentage of interpolation calculation reduction when applying the algorithm for each level. It is demonstrated very clear that Sequence C1 gives the

lowest interpolation reduction percentage in all three level while sequence C4 and C4 give the highest interpolation reduction percentage. This is because Sequence C1 contains a lot of fast motion, while Sequence C4 and C4 does not. And from Level 1 to Level 3, the interpolation calculation reduction increases, however, the complexity added by the algorithm also increases from level 1 to level 3. To see the trade-off between interpolation calculation reduction and complexity added by the algorithm, it is necessary to implement the algorithm in hardware.

2.2. Hardware Implementation of the Algorithm

a. Trade-off between efficiency and complexity of the algorithm

- After implementing in hardware, the results of both interpolation calculation reduction and complexity added by the algorithm for each level are out. Table 11 illustrates the results for each level. Apparently, the results show that from level 1 to level 3, the interpolation calculation reduction significantly compared to level 1, which requires the highest number of interpolation calculation. However, it comes with the price that the complexity representing by gate count added by the algorithm is also increased from level 1 to level 3. Based on the tradeoff between the complexity of the algorithm and number of interpolation calculation it requires, it is clear that level two seems to be the best one to apply this algorithm. Level 3 requires the lowest number of interpolation which is apparently is the best for FME, but it cannot compensate the complexity added by the algorithm which is too large compared with level 2 and level 1.

Table 11: Complexity and Interpolation reduction of the algorithm for each level

Level	Number of Cycles	Gate count	Number of Interpolated Pixels/CTU	Interpolation Reduction compared to level 1
1	13	25,175	259,304	0%
2	23	38,677	232,962	10.16%
3	79	331,472	212,382	18.10%

b. Internal Memory Requirement

- To apply the algorithm, huge internal memory requirements is another problem of this algorithm. Table 12 illustrates internal memory requirement for different cases in each level. It is apparent that the internal memory requirement is too large for all level. From the trade-off between complexity and interpolation reduction and also considering the memory requirement, it is safe to say that level 1 is the best choice for range-based interpolation.

Table 12: Internal Memory Requirement for Range-based algorithm

	Level 1	Level 2	Level 3
Bad cases	609.44 KB	554.37 KB	504.66 KB
Good cases	76.24 KB	74.20 KB	69.44 KB
Average	253.23 KB	231.93 KB	213.21 KB

- However, the memory requirements are too large, it is critical to limit the internal memory requirement and adapt the algorithm with the restriction

c. Memory Restriction

- As discussed above, it is the best to apply the algorithm for level 2, which means that 593 PUs are divided into five groups including four groups of 32x32 block and a one group of depth-0 PUs as illustrated in Fig. 22 (a), where a group of 32x32 block includes all PUs from depth 1 to depth 3, which are spatially inside that 32x32 block, and the group of depth-0 PUs includes all PU of the depth 0 CU. Representing the five groups by five rectangles, and then the algorithm is applied for that 5 rectangles, which means N_{rec}=5 as illustrated in Fig. 22 (b). Then, the interpolators do interpolation at once inside that united region only.

- After interpolating the whole rectangle region, the interpolated pixels need to be stored in internal memory. As the result, when the rectangle size is large, the number of interpolated pixels need to be stored is large, the internal buffer memory is large. Therefore, it is necessary to restrict the size of the internal buffer size or the size of the rectangles.

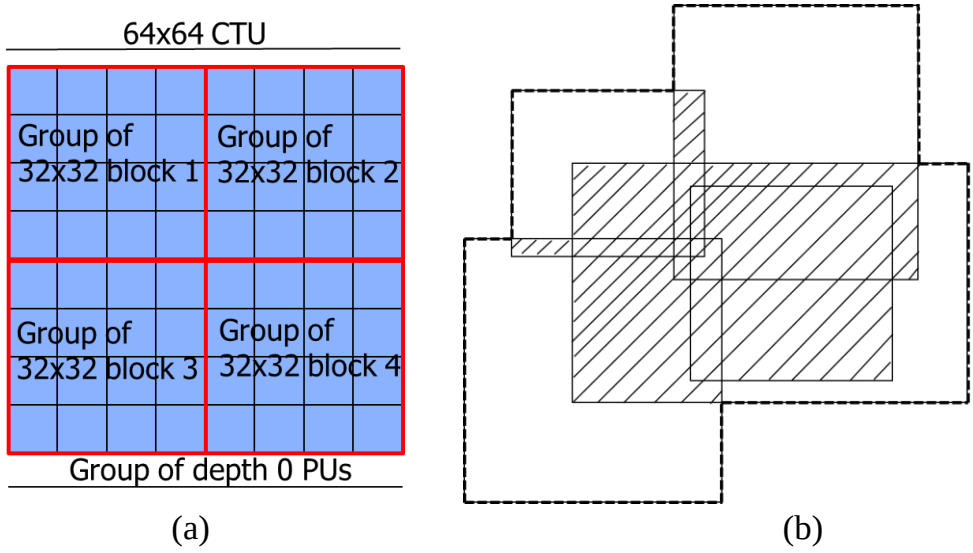


Fig. 22: (a) Dividing 593 PUs of a CTU into five groups. (b) Finding the union of the five rectangles representing the five groups

Table 13: SRAM requirement and rectangle size

Rectangle size	80*80	96*96	112*112	128*128	160*160	192*192
SRAM Requirement (KB)	100.63	144.75	196.88	257.00	401.25	577.50

Table 14: Percentage of Out-range CTUs for each of SRAM restricted size

Rectangle size	80*80	96*96	112*112	128*128	160*160	192*192
Percentage of Out-range CTUs	46.79%	39.52%	31.88%	25.96%	16.36%	9.26%

- The internal buffer size should be restricted to 100.63 KB, which means that the rectangle size is restricted to be smaller than 80*80. Then CTUs whose the rectangle boundary is smaller than or equal to 80*80, can be processed without any problem, but CTUs whose rectangle boundary is larger than 80*80, can be processed by the

algorithm. These CTUs are called Out-range CTUs. Table 14 illustrates the percentage of Out-range CTUs per total CTUs for each of internal memory's size restriction. It is clear that the smaller the internal memory size is restricted to, the bigger the percentage of Out-range CTUs is

d. Divide and Conquer Algorithm for Memory Restriction.

- As mention above, to process Out-range CTUs, it is unavoidable to adjust the algorithm. There are 593 PUs at CTU level, enclosing all of them in one rectangle makes the rectangle size too big and therefore CTUs Out-range CTU. To avoid it, for Out-range CTU, 593 PUs are divided into five groups including four groups of 32x32 block and a one group of depth-0 PUs, where a group of 32x32 block includes all PUs from depth 1 to depth 3, which are spatially inside that 32x32 block, and the group of depth-0 PUs includes all PU of the depth 0 CU. Each one of the five groups is processed individually the same process as at CTU level. This means that for each group, all of the PUs in that group are enclosed by a rectangle. if the size of the rectangle boundary is small, it is processed by the range-base algorithm normally, if the size of the rectangle boundary is still big, it is further divided into smaller groups and recursively process them the same.
- Run the algorithm with different memory size restriction, the experimental results are illustrated in Table 15, Table 16, Table 17, Table 18, and Table 19 for 80*80, 96*96, 112*112, 128*128, and 192*192 memory size restriction, respectively
- As illustrated in Table 20, where Number of interpolated pixels/CTU is the average number of fractional pixels interpolated for a CTU, using range-based algorithm with divide and conquer for a certain memory size restriction, where Number of interpolated pixels/CTU (Original algorithm in HM) refers to the average number of fraction pixels interpolated for a CTU, using the conventional method as in HM reference software. Interpolation calculation reduction illustrates the percentage of interpolation calculation reduction of range-based algorithm compared with the original algorithm in HM. As predicted, the larger the memory size is, the bigger the percentage of interpolation calculation reduction is.

Table 15: Interpolation Calculation Reduction of the Algorithm with 80*80 memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
Seq. C1	631,118	1,720,025	63.31%
Seq. C2	555,293	1,720,025	67.72%
Seq. C3	482,281	1,720,025	71.96%
Seq. C4	410,164	1,720,025	76.15%
Seq. C5	338,598	1,720,025	80.31%
Seq. C6	354,789	1,720,025	79.37%
Average	398,139	1,720,025	76.85%

Table 16: Interpolation Calculation Reduction of the Algorithm with 96*96 memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
Seq. C1	568,295	1,720,025	66.96%
Seq. C2	495,153	1,720,025	71.21%
Seq. C3	425,558	1,720,025	75.26%
Seq. C4	357,059	1,720,025	79.24%
Seq. C5	289,441	1,720,025	83.17%
Seq. C6	304,833	1,720,025	82.28%
Average	350,012	1,720,025	79.65%

Table 17: Interpolation Calculation Reduction of the Algorithm with 112*112 memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
Seq. C1	535,181	1,720,025	68.89%
Seq. C2	463,631	1,720,025	73.05%
Seq. C3	395,002	1,720,025	77.04%
Seq. C4	326,994	1,720,025	80.99%
Seq. C5	259,927	1,720,025	84.89%
Seq. C6	273,735	1,720,025	84.09%
Average	322,347	1,720,025	81.26%

Table 18: Interpolation Calculation Reduction of the Algorithm with 128*128 memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
Seq. C1	518,291	1,720,025	69.87%
Seq. C2	446,842	1,720,025	74.02%
Seq. C3	377,545	1,720,025	78.05%
Seq. C4	308,785	1,720,025	82.05%
Seq. C5	240,405	1,720,025	86.02%
Seq. C6	253,106	1,720,025	85.28%
Average	305,183	1,720,025	82.26%

Table 19: Interpolation Calculation Reduction of the Algorithm with 192*192 memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
Seq. C1	448,162	1,720,025	73.94%
Seq. C2	376,783	1,720,025	78.09%
Seq. C3	309,357	1,720,025	82.01%
Seq. C4	245,047	1,720,025	85.75%
Seq. C5	183,981	1,720,025	89.30%
Seq. C6	194,719	1,720,025	88.68%
Average	247,753	1,720,025	85.60%

Table 20: Interpolation Calculation Reduction of the Algorithm with different memory size restriction

	Number of interpolated pixels/CTU	Number of interpolated pixels/CTU (Original algorithm in HM)	Interpolation calculation reduction
80*80	398,139	1,720,025	76.85%
96*96	350,012	1,720,025	79.65%
112*112	322,347	1,720,025	81.26%
128*128	305,183	1,720,025	82.26%
192*192	247,753	1,720,025	85.60%

e. Interpolator's size decision

- There are several interpolator sizes to consider for FME's interpolator design. The most used ones are 4xN, 8xN, and 16xN interpolators, where 4, 8 and 16 are horizontal bandwidths of each interpolator respectively. To decide which interpolator is the most reasonable one to use, it is critical to consider the target throughput and the throughput each interpolator can offer.

- The target throughput is 3,200 cycle per CTU. Based on the throughput in the best case, worst case and in average each interpolator size offers as illustrated in Table 21. In order to meet the throughput requirement, the 16xN size interpolator, which is illustrated in Fig. 23 is the best to use.

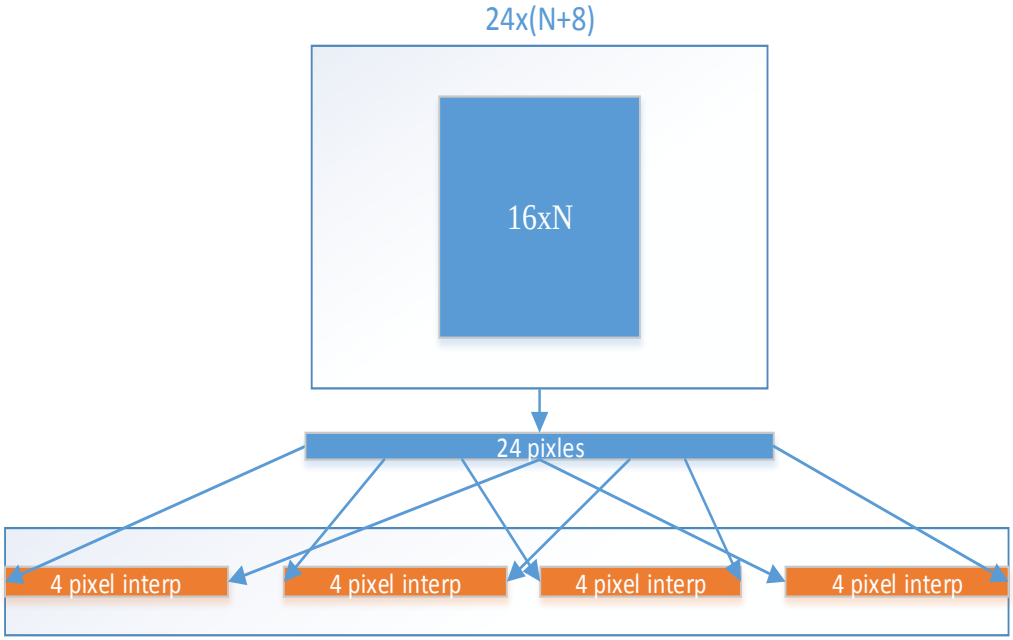


Fig. 23: an example of 16xN interpolator

Table 21: Throughput of each interpolator size

	4xN	8xN	16xN
Throughput-worst case	8,795 cycle/CTU	4,539 cycle/CTU	2,307 cycle/CTU
Throughput-best case	1,108 cycle/CTU	572 cycle/CTU	291 cycle/CTU
Throughput-average	3,640 cycle/CTU	1,879 cycle/CTU	955 cycle /CTU

- As illustrated in Fig. 23, a 16xN interpolator can be comprised of four 4xN interpolator work in parallel. After range-based algorithm, the union information is sent to interpolators, based on that information,

the 16xN interpolators start to interpolate from the top to the bottom of the region, and then save the interpolated pixels to SRAM.

f. Internal Memory Reduction with Quarter Pixel's Bilinear Estimation

Table 22: Rectangle size, SRAM size requirement, and percentage of Out-range CTU with Quarter Pixel Bilinear Estimation

Rectangle size	80*80	96*96	112*112	128*128	160*160	192*192
Percentage of Out-range CTUs	46.79 %	39.52 %	31.88%	25.96%	16.36%	9.26%
SRAM (KB)	25.31	36.38	49.44	64.50	100.63	144.75

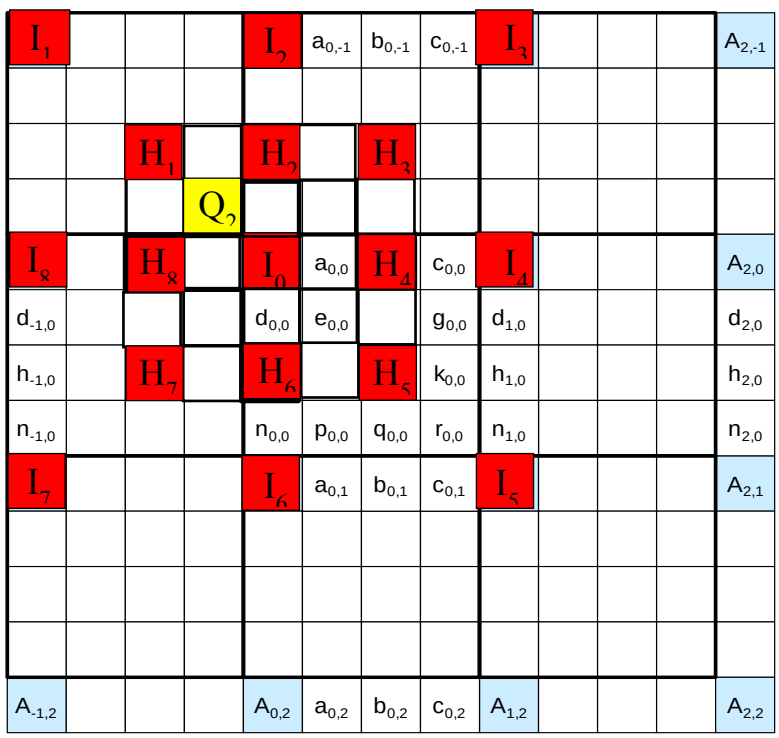


Fig. 24: Quarter pixels Bilinear Estimation

- The memory restriction is applied with the restriction of rectangle boundary size is smaller than 80*80. However, it is still large. To further reduce the memory size, the quarter pixel's bilinear estimation.
- In HEVC, quarter pixels are generated by 7-tap filters, which is one of the reasons why the interpolation process is very complicated. If quarter pixels are generated by bilinear filters as in H.264, then interpolation of quarter-pixels can be exempted because the Hadamard transform coefficients can be calculated from Hadamard transform coefficients of half pixels and integer pixel as the following equation:

$$HT(O - Q_2) = HT\left(O - \frac{I_0 + H_1 + 1}{2}\right)$$

  where O represents original block's pixels, Q_2, I_0, and H_1 represent quarter pixels, integer pixels and half pixels as illustrated in Fig. 24.

- With Quarter Pixel's Bilinear Estimation, the internal memory can be reduced significantly as illustrated in Table 22. The trade-off for it is the BD-BR increase of BDBR increase: 0.55%

- If the 80*80 rectangle size restriction still holds, the n with Quarter Pixel's Bilinear Estimation, the internal buffer size is just 25.31 KB.

g. Comparison with other similar works

Table 23: Comparison of the range-based algorithm and other algorithms

Algorithm	BD-BR increase	Complexity reduction
Range-based algorithm	0 %	86.46% %
[5]	1.1%	34.87%
[16]	2.07%	75%

- There are some similar researches have been proposed previously [5], [16]. In [5] adopts interpolation free FME for depth 0 and depth 1. Interpolation free FME is an algorithm to do FME refinement without interpolation based on IME results and mathematical model. In [16], the authors utilize Quarter Pixel's Bilinear Estimation scheme to

avoid interpolate quarter pixels and use 5T12S search pattern to reduce the number of search candidates from 25 to 12.

- Table 23 shows the comparison of the range-based algorithm with that two algorithms. It is very clear that the proposed algorithm performs much better than the other two algorithm

Chapter 6. Conclusion

- This work proposed two efficient algorithms to tackle two main causes of FME's enormous computational complexity. As stated previously, FME is the computational bottleneck of real-time HEVC encoders due to two reasons: a large number of FME refinements processed and a complicated and redundant interpolation process. Normally, previous works attempt to solve one of the two reasons to reduce the complexity of FME in HEVC. However, this work proposes algorithms to solve both of the two reason in order to form a combined method which gives a high complexity reduction with minimal encoding performance decrease.
- The first proposed algorithm, called the Advanced PU Partition and CU Depth Decision algorithm, attempts to reduce the number of FME refinements for PUs in a CTU. In previous works, several attempts have been successfully proposed for H.264 but direct application of these algorithms to HEVC results in a significant degradation of the encoding efficiency due to the difference of coding structures between HEVC and H.264 [3]. For HEVC, an efficient advanced PU partitions decision for FME is proposed in [3]. However, it does not take into account asymmetric partition which is one of the key tools improving the coding efficiency in HEVC compared to H. 264. In addition, it also does not take into account the variation of temporal correlation among frames and treats all the reference frames equally which can cause extra computational complexity. Therefore, this work proposes Advanced PU Partition and CU Depth Decision algorithm that takes into account all asymmetric partitions as well as exploits the variation of temporal correlation between the current frame and multiple reference frames to efficiently predetermine PU partition types and CU depths for FME. The algorithm is divided into two parts: Advanced PU Partition Decision, and Advanced CU Depth Decision. In the first part, for a given CU, the PU partition type for FME is selected by comparing the IME costs of the corresponding PU partitions. To this end, the IME costs of all 7 partition types (3 if it is 8x8 CU) are compared for every reference frame. Note that each CU consists of two PUs for partition type 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N. The IME costs of the two PUs constituting a CU are

summed and then compared with each other. Then, the PU partition types are sorted according to the IME cost in each reference frame and the PU types with the smallest Nref_idx IME costs are selected for FME. The number Nref_idx is predefined for each reference frame. In general, a closer reference frame is more important than a distant reference frame. Therefore, a larger number is assigned as Nref_idx for a closer reference frame than a distant frame. All discarded partitions are kept with IME costs and used for best mode decision in a later stage. In the second part, IME costs of all CUs in a CTU are compared. Then, the best IME CU depths are determined. From the selected depth, the CU depth(s) for FME are finally chosen in four different options: the best IME depth only (best_only), the best IME depth plus one more depth along one direction (best_plus_one), the best IME depth plus one more depth along two directions (best_plus_minus_one), and best IME depth plus two additional depths (best_plus_two). In the case of depth 1 chosen from IME, FME is performed only for depth 1 in best_only option, depths 1 and 2 in best_plus one option, depths 0, 1, and 2 in both best_plus_minus and best_plus_two option. The remaining depths are kept with the IME cost and used for CU depth decision in a later stage. The Advanced PU Partition and CU Depth Decision, consisting of the two parts, reduce dramatically the complexity of FME without significantly degrading the encoding performance. The experimental results show that the algorithm reduces up to 67.47% with a BD-BR increase 1.08%. The second algorithm, the Range-based algorithm, attempts to reduce redundancy in interpolation process of FME. In previous works, none of the so far proposed algorithms attempt to reduce the redundant interpolation caused by the similarity in motion among neighboring PUs. Therefore, this work proposes the Range-based algorithm that reduces a large amount of redundant interpolation calculation by avoiding repeatedly interpolating overlapped regions caused by the motional similarity among neighboring PUs. The algorithm divides 593 PUs of a CTU into five spatial groups and each group is represented by only one rectangle which is the smallest rectangle containing all PUs in that group. The union region of the five representing rectangles is calculated, and then interpolation is processed inside that union region from the top to the bottom. By this manner, repeated interpolation calculations in overlapped regions

among PUs are reduced significantly. Experimental results show that the algorithm reduces up to 86.46% interpolation computation without any encoding performance decrease. And the Range-based algorithm for dividing PUs into spatial groups and finding the union of 5 representing rectangles only requires 25k gates in a standard 130nm CMOS technology at an operating frequency of 647MHz.

- The combination of the two algorithms creates a coherent solution to reduce the complexity of FME. Considering interpolation is a half of the complexity of an FME refinement, then the complexity of FME could be reduced more than 90% with a BD-BR increase of 1.66%

Bibliography

[1] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas 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.

[2] Jarno Vanne, Marko Viitanen, and Timo D. Hämäläinen, "Efficient Mode Decision Schemes for HEVC Inter Prediction," IEEE Trans. Circuits Syst. Video Technol., vol. 24, no. 9, pp. 1579 – 1593, Sep. 2014.

[3] Shiau-Yu Jou and Tian-Sheuan Chang, "Fast Prediction Unit Selection for HEVC Fractional pel Motion Estimation Design," 2013 IEEE Workshop on Signal Processing Systems, pp. 247 – 250, Oct. 2013.

[4] Chun-Yu Lung and Chung-An Shen, "A High-Throughput interpolator for Fractional Motion Estimation in High Efficient Video Coding (HEVC) systems," Circuits and Systems (APCCAS), 2014 IEEE Asia Pacific Conference, pp. 268 – 271, Nov. 2014.

[5] Shiaw-Yu Jou, Shan-Jung Chang, and Tian-Sheuan Chang, "Fast Motion Estimation Algorithm and Design for Real Time QFHD High Efficiency Video Coding," IEEE Trans. Circuits Syst. Video Technol., vol. 25, no. 9, pp. 1533 – 1544, Jan. 2015.

[6] Yu-Jen Wang, Chao-Chung Cheng, and Tian-Sheuan Chang, "A Fast Algorithm and Its VLSI Architecture for Fractional Motion Estimation for H.264/MPEG-4 AVC Video Coding," IEEE Trans. Circuits Syst. Video Technol., vol.17, no.5, pp.578-583, May 2007.

[7] C.-C. Yang, K.-J. Tan, Y.-C. Yang and J.-I. Guo, "Low complexity fractional motion estimation with adaptive mode selection for H.264/AVC," in Proceeding of IEEE International Conference on Multimedia and Expo., pp.673-678, July 2010.

[8] C.-C. Lin, Y.-K. Lin, and T.-S. Chang, "A fast algorithm and itsarchitecture for motion estimation in MPEG-4 AVC/H.264," in proceedings of Asia Pacific Conference on Circuits and Systems, pp.1250-1253, Dec. 2006.

[9] Henrique Maich, Vladimir Afonso, Bruno Zatt, Luciano Agostini, Marcelo Porto, " HEVC Fractional Motion Estimation complexity reduction

for real-time applications," Circuits and Systems (LASCAS), 2014 IEEE 5th Latin American Symposium, 25-28 Feb. 2014.

[10] Nguyen Ngoc Luong, Tae Sung Kim, Hyuk-Jae Lee, and Soo-Ik Chae, "Advanced Decision of PU Partitions and CU Depths for Fractional Motion Estimation in HEVC," International Conference on Electronics, Information and Communication, Jan. 2016.

[11] Grzegorz Pastuszak, Maciej Trochimiuk, "Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder," J Real-Time Image Proc, DOI 10.1007/s11554-015-0516-4.

[12] J.-F. Chang and J.-J. Leou, "A quadratic prediction based fractional-pixel motion estimation algorithm for H.264," inProc. 7th IEEE Int. Symp. Multimedia, Dec. 2005.

[13] Il-Koo Kim, Junghye Min, Tammy Lee, Woo-Jin Han, and JeongHoon Park," Block Partitioning Structure in the HEVC Standard," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 22, NO. 12, DECEMBER 2012.

[14] Kemal Ugur, Alexander Alshin, Elena Alshina, Frank Bossen, Woo-Jin Han, Jeong-Hoon Park, and Jani Lainema, "Motion Compensated Prediction and Interpolation Filter Design in H.265/HEVC," IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, VOL. 7, NO. 6, DECEMBER 2013.

[15] Chae Eun Rhee, Kyujoong Lee, Tae Sung Kim and Hyuk-Jae Lee, "A Survey of Fast Mode Decision Algorithms for Inter-Prediction and Their Applications to High Efficiency Video Coding," IEEE Transactions on Consumer Electronics(SCI, IF 0.941), Volume : 58, Issue : 4, Pages : 1375-1383, November, 2012.

[16] Gang He, Dajiang Zhou, Yunsong Li, Zhixiang Chen, Tianruo Zhang, and Satoshi Goto, "High-Throughput Power-Efficient VLSI Architecture of Fractional Motion Estimation for Ultra-HD HEVC Video Encoding," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume:23 , Issue: 12, Pages: 3138 – 3142, March 2015.

Abstract in Korean

High-Efficiency Video Coding (HEVC) [1]은 최신의 영상 coding 표준으로 Joint Collaborative Team on Video Coding (JCT-VC)에 의해 만들어졌으며, 전신인 H.264 표준 대비 2 배의 부호화 효율과 상대적으로 높은 영상 품질을 달성하는 것을 목표로 한다. Motion Estimation (ME)은 integer motion estimation (IME)과 fractional motion estimation (FME)로 이루어지는데, HEVC 연산의 병목으로 작용한다. HM 참조 소프트웨어 수행 시에 ME 단독으로 수행 시간의 50 % 가량을 차지하며, IME 가 대략 20 %, FME 가 30 % 가량을 구성한다[2]. FME 의 막대한 연산 복잡도는 다음의 두 가지 이유로 설명될 수 있다:

- 많은 FME refinement 실행: HEVC 에서는 크기가 대개 64x64 픽셀인 CTU 단위로 프레임이 분할된다. 하나의 64x64 CTU 는 depth 0 의 64x64 CU 하나, depth 1 의 32x32 CU 4 개, depth 2 의 16x16 CU 16 개, depth 3 의 8x8 CU 64 개를 포함하여 총 85 개의 CU 로 구성된다. 각 CU 는 8 가지의 허용되는 partition type 에 따라 PU 로 쪼개진다. HEVC 부호화기는 대개 4 장의 참조 프레임을 사용하여, 모든 가능한 PU 에 대해 FME refinement 를 수행해본 후, CTU 의 최적 configuration 을 결정한다. 결국 HEVC 참조 소프트웨어 HM 은 일반적으로 하나의 CTU 를 위해, 매우 많은 연산 자원을 소모하는 FME refinement 를 2,372 번 수행해야 한다.
- 복잡하고 중복이 많은 보간 과정: FME refinement 는 보간과 sum of absolute transformed difference (SATD) 로 구성되어 있는데, 일반적으로 4 장의 참조 프레임의 해당되는 모든 PU 에 대해

수행된다. 결국 64x64 CTU 하나의 fractional pixel refinement 를 처리하기 위해서, FME 는 6,232,900 개의 fractional 픽셀을 처리해야 한다. 게다가 HEVC 에서는 fractional 픽셀이 half fractional 픽셀과 quarter fractional 픽셀로 구성되어 있는데, 각 fractional 픽셀에 대해 6-tap 필터와 bilinear 필터를 사용하는 이전 표준들과 달리 8-tap 필터와 7-tap 필터로 보간이 수행된다. 따라서 FME 의 보간 과정은 HEVC 부호화기에 극심한 연산 부담이 된다.

본 논문에서는 위의 두 가지 항목 각각을 해결할 수 있는 두 개의 알고리즘을 제안한다. 첫 번째 알고리즘인 Advanced Decision of PU Partitions and CU Depths for FME 는 IME 의 cost 를 추정한 후, FME 를 위하여 CU level 의 PU partition type 과 coding tree unit (CTU) level 의 CU depth 를 선택한다. 이 알고리즘은 1.08 % 의 BD-BR 저하로 복잡도를 67.47 % 감소시킬 수 있어 효과적임을 실험 결과로 확인하였다. 두 번째 알고리즘인 A Reduction of the Interpolation Redundancy for FME 는 아무런 부호화 성능 저하 없이 보간 연산을 최대 86.46 % 감소시킨다. 두 가지 알고리즘의 조합은 FME 의 복잡도를 줄이기 위한 완전한 해결책이라고 할 수 있다. 보간이 FME refinement 복잡도의 절반을 차지하는 것을 고려할 때, BD-BR 1.66 % 증가로 FME 의 복잡도를 85% 이상 감소시킬 수 있다.

주요어 : High-Efficiency Video Coding; Motion estimation; Fractional motion estimation; Interpolation; Complexity Reduction

학 번 : 2014-25271