|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  38th Meeting: Brussels, BE, 10–17 January 2020 | Document: JCTVC-AL1002 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **High Efficiency Video Coding (HEVC) Test Model 16 (HM 16)** Improved Encoder Description Update 13 | | |
| *Status:* | Output Document of JCT-VC | | |
| *Purpose:* | Report | | |
| Author(s) or Contact(s): | C. Rosewarne  K. Sharman  R. Sjöberg  G. Sullivan | Email: | chris.rosewarne@canon.com.au  [karl.sharman@eu.sony.com](mailto:karl.sharman@eu.sony.com)  [rickard.sjoberg@ericsson.com](mailto:rickard.sjoberg@ericsson.com)  garysull@microsoft.com |
| *Source:* | Editors | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Abstract**

The JCT-VC released HEVC test model (HM) 16.21 software following its 37th meeting in Geneva, with no further update resulting from the 38th meeting in Brussels. This document serves as an overview of HEVC Version 1 and the Range Extensions of HEVC Version 2, and also provides an encoder-side description of the HM-16.21 software.

**CONTENTS**

Contents

[List of acronyms 1](#_Toc27473080)

[1 Introduction 1](#_Toc27473081)

[1.1 Overview of coding structures 1](#_Toc27473082)

[1.2 Obtaining the HEVC test model software 3](#_Toc27473083)

[1.3 Obtaining the HEVC Standard 3](#_Toc27473084)

[2 Scope 3](#_Toc27473085)

[3 Encoder control 3](#_Toc27473086)

[3.1 Encoder configuration options 3](#_Toc27473087)

[3.1.1 File, format and profile/level/tier configuration options 3](#_Toc27473088)

[3.1.2 Coding tool configuration options 3](#_Toc27473089)

[3.1.3 Slice coding parameters 4](#_Toc27473090)

[3.1.4 Motion search options 4](#_Toc27473091)

[3.1.5 Mode decision parameters 5](#_Toc27473092)

[3.1.6 Quantization parameters 5](#_Toc27473093)

[3.1.7 Pre-filtering control parameters 6](#_Toc27473094)

[3.1.8 Rate control parameters 6](#_Toc27473095)

[3.1.9 SEI message configuration options 6](#_Toc27473096)

[3.1.10 Supplemental Motion Vector Estimation hint configuration options 7](#_Toc27473097)

[4 Overview of tools in HEVC Version 1 and HEVC Version 2 RExt 7](#_Toc27473098)

[4.1 High level syntax 7](#_Toc27473099)

[4.1.1 Bitstream organization 7](#_Toc27473100)

[4.1.2 Parameter sets 7](#_Toc27473101)

[4.1.3 Picture types 7](#_Toc27473102)

[4.1.4 Reference picture set 8](#_Toc27473103)

[4.2 Picture partitioning 8](#_Toc27473104)

[4.2.1 Coding tree unit (CTU) partitioning 8](#_Toc27473105)

[4.2.2 Slice and tile structures 9](#_Toc27473106)

[4.2.3 Coding unit (CU) and coding tree structure 10](#_Toc27473107)

[4.2.4 Prediction unit (PU) structure 11](#_Toc27473108)

[4.2.5 Transform unit (TU) and transform tree structure 12](#_Toc27473109)

[4.3 Intra prediction 12](#_Toc27473110)

[4.3.1 Prediction modes 12](#_Toc27473111)

[4.3.2 Filtering of neighbouring samples 14](#_Toc27473112)

[4.3.3 Intra boundary filter 14](#_Toc27473113)

[4.3.4 4:2:2 chroma format mode adjustment 15](#_Toc27473114)

[4.4 Inter prediction 17](#_Toc27473115)

[4.4.1 Prediction modes 17](#_Toc27473116)

[4.4.2 Motion vector prediction 21](#_Toc27473117)

[4.4.3 Interpolation filter 22](#_Toc27473118)

[4.4.4 Weighted Prediction 23](#_Toc27473119)

[4.5 Transform and quantization (scaling) 24](#_Toc27473120)

[4.5.1 Inverse transforms 24](#_Toc27473121)

[4.5.2 1D inverse transform matrices 24](#_Toc27473122)

[4.5.3 Scaling and quantization 25](#_Toc27473123)

[4.5.4 Scaling lists 25](#_Toc27473124)

[4.5.5 Scaling lists for transform skipped TUs 26](#_Toc27473125)

[4.5.6 Transform selection for the 4:2:2 chroma format 26](#_Toc27473126)

[4.5.7 Chroma QP initialization offset table 26](#_Toc27473127)

[4.5.8 Extended precision processing 26](#_Toc27473128)

[4.5.9 CU-adaptive chroma QP offset 27](#_Toc27473129)

[4.6 Residual prediction in case of transquant bypass and transform skip 27](#_Toc27473130)

[4.7 Entropy coding 28](#_Toc27473131)

[4.7.1 CABAC alignment 28](#_Toc27473132)

[4.8 Coefficient Coding 28](#_Toc27473133)

[4.8.1 Transform skip residual rotation 28](#_Toc27473134)

[4.8.2 Significance map context modelling 28](#_Toc27473135)

[4.8.3 Rice parameter adaptation 28](#_Toc27473136)

[4.8.4 Maximum coeff\_abs\_level\_remaining codeword length restriction 29](#_Toc27473137)

[4.9 Cross-component prediction 29](#_Toc27473138)

[4.10 Loop Filtering 30](#_Toc27473139)

[4.10.1 Overview of Loop filtering 30](#_Toc27473140)

[4.10.2 Deblocking filter 30](#_Toc27473141)

[4.10.3 Sample adaptive offset filter 34](#_Toc27473142)

[4.11 Wavefront parallel processing 35](#_Toc27473143)

[5 Profiles, Levels and Tiers 36](#_Toc27473144)

[6 Description of the HM encoder and encoding methods 36](#_Toc27473145)

[6.1 Encoder configurations 37](#_Toc27473146)

[6.1.1 Overview of encoder configurations 37](#_Toc27473147)

[6.1.2 Intra-only configuration 37](#_Toc27473148)

[6.1.3 Low-delay configurations 37](#_Toc27473149)

[6.1.4 Random-access configuration 38](#_Toc27473150)

[6.2 Cost mode 40](#_Toc27473151)

[6.3 Cost Functions 40](#_Toc27473152)

[6.3.1 Sum of Square Error (SSE) 40](#_Toc27473153)

[6.3.2 Sum of Absolute Difference (SAD) 40](#_Toc27473154)

[6.3.3 Hadamard transformed SAD (SATD) 40](#_Toc27473155)

[6.3.4 RD cost functions 41](#_Toc27473156)

[6.3.5 Lambda modifiers 42](#_Toc27473157)

[6.4 Slice and tile partitioning operation 42](#_Toc27473158)

[6.5 Derivation process for CU-level and PU-level coding parameters 43](#_Toc27473159)

[6.5.1 Intra prediction mode and parameters 43](#_Toc27473160)

[6.5.2 Inter prediction mode and parameters 43](#_Toc27473161)

[6.5.3 Intra/Inter/PCM mode decision 50](#_Toc27473162)

[6.5.4 Adaptive QP 52](#_Toc27473163)

[6.6 Derivation process for TU-level coding parameters 53](#_Toc27473164)

[6.6.1 Residual quadtree partitioning 53](#_Toc27473165)

[6.6.2 Rate-distortion optimized quantization 53](#_Toc27473166)

[6.6.3 Quantization rounding for residual DPCM 54](#_Toc27473167)

[6.6.4 Cross-component prediction 55](#_Toc27473168)

[6.6.5 Transform skip selection 55](#_Toc27473169)

[6.6.6 Sign data hiding 55](#_Toc27473170)

[6.7 Inter-prediction residual quadtree derivation 56](#_Toc27473171)

[6.8 Quantization control 56](#_Toc27473172)

[6.9 Rate control 56](#_Toc27473173)

[6.9.1 Workflow for bit allocation and lambda estimation 59](#_Toc27473174)

[6.9.2 Workflow for parameters update 62](#_Toc27473175)

[6.9.3 Target bits saturation for rate control 63](#_Toc27473176)

[6.10 Derivation process for slice-level coding parameters 64](#_Toc27473177)

[6.10.1 Sample Adaptive Offset (SAO) parameters 64](#_Toc27473178)

[6.10.2 Adaptive QP selection 64](#_Toc27473179)

[6.10.3 Adaptive search range for motion estimation 65](#_Toc27473180)

[6.10.4 Weighted prediction control 65](#_Toc27473181)

[6.11 Pre-encoding GOP-based temporal filter 66](#_Toc27473182)

[6.12 Encoder-only supplemental motion vector estimation for point cloud coding content 68](#_Toc27473183)

[6.12.1 Supplemental motion vector estimation file formats 69](#_Toc27473184)

[7 References 69](#_Toc27473185)

List of figures

[Figure 1‑1. Simplified block diagram of HM encoder. 2](#_Toc13594798)

[Figure 4‑1. Example of a picture divided into CTUs. 9](#_Toc13594799)

[Figure 4‑2. Example of slices and slice segments. 9](#_Toc13594800)

[Figure 4‑3. Examples of tiles and slices. 10](#_Toc13594801)

[Figure 4‑4. Example of coding tree structure. 11](#_Toc13594802)

[Figure 4‑5. 8 partition modes for inter PU. 11](#_Toc13594803)

[Figure 4‑6. Example of transform tree structure within CU. 12](#_Toc13594804)

[Figure 4‑7. The 33 intra prediction directions. 13](#_Toc13594805)

[Figure 4‑8. Mapping between intra prediction direction and intra prediction mode. 13](#_Toc13594806)

[Figure 4‑9. Intra boundary filter example. 15](#_Toc13594807)

[Figure 4‑10. Intra prediction directions in luma for example 16x16 PB. 16](#_Toc13594808)

[Figure 4‑11. Intra prediction modes for chroma PBs in the 4:2:2 chroma format. 17](#_Toc13594809)

[Figure 4‑12. Derivation process for merge candidates list construction. 18](#_Toc13594810)

[Figure 4‑13. Positions of spatial merge candidates. 19](#_Toc13594811)

[Figure 4‑14. Candidate pairs considered for redundancy check of spatial merge candidates. 19](#_Toc13594812)

[Figure 4‑15. Positions for the second PU of N×2N and 2N×N partitions. 19](#_Toc13594813)

[Figure 4‑16. Illustration of motion vector scaling for temporal merge candidate. 20](#_Toc13594814)

[Figure 4‑17. Candidate positions for temporal merge candidate, C0 and C1. 20](#_Toc13594815)

[Figure 4‑18. Example of combined bi-predictive merge candidate. 20](#_Toc13594816)

[Figure 4‑19. Derivation process for motion vector prediction candidates. 21](#_Toc13594817)

[Figure 4‑20. Illustration of motion vector scaling for spatial motion vector candidate. 22](#_Toc13594818)

[Figure 4‑21. Lossless scaling and transformation process (quantities illustrated for HEVC version 1 profiles). 24](#_Toc13594819)

[Figure 4‑22. Transform skip scaling and transformation process (quantities illustrated for HEVC version 1 profiles). 24](#_Toc13594820)

[Figure 4‑23. Inverse transformation process (quantities illustrated for HEVC version 1 profiles). 24](#_Toc13594821)

[Figure 4‑24. Scaling lists. 26](#_Toc13594822)

[Figure 4‑25. Square transform arrangement for the 4:2:2 chroma format.. 26](#_Toc13594823)

[Figure 4‑26. Diagram showing magnitude bit depths in HEVC encoding path. 27](#_Toc13594824)

[Figure 4‑27. Overall processing flow of deblocking filter process. 31](#_Toc13594825)

[Figure 4‑28. Flow diagram for Bs calculation. 31](#_Toc13594826)

[Figure 4‑29. Referred information for Bs calculation at CTU boundary. 32](#_Toc13594827)

[Figure 4‑30. Pixels involved in filter on/off decision and strong/weak filter selection. 32](#_Toc13594828)

[Figure 4‑31. Deblocking behaviour in the 4:2:2 chroma format. 34](#_Toc13594829)

[Figure 4‑32. Four 1-D 3-pixel patterns for the pixel classification in EO. 35](#_Toc13594830)

[Figure 4‑33. Four bands are grouped together and represented by its starting band position. 35](#_Toc13594831)

[Figure 6‑1. Graphical presentation of intra-only configuration. 37](#_Toc13594832)

[Figure 6‑2. Graphical presentation of low-delay configuration. 38](#_Toc13594833)

[Figure 6‑3. Graphical presentation of random-access configuration. 39](#_Toc13594834)

[Figure 6‑4. Diamond and enhanced diamond search flowchart. 46](#_Toc13594835)

[Figure 6‑5. 8-point search with iDist equal to one. 47](#_Toc13594836)

[Figure 6‑6. 8-point search with iDist from two to eight. 47](#_Toc13594837)

[Figure 6‑7. 8-point search with iDist greater than eight. 47](#_Toc13594838)

[Figure 6‑8. The schematic of Intra/Inter/PCM mode decision. 52](#_Toc13594839)

[Figure 6‑9. Dead zone uniform quantizer with rounding offset. 55](#_Toc13594840)

[Figure 6‑10. Example CPB behavior. 63](#_Toc13594841)

[Figure 6‑11. The different layers of the hierarchical motion estimation. L0 is the original resolution. L1 is a subsampled version of L0. L2 is a subsampled version of L1. 66](#_Toc13594842)

List of tables

[Table 3‑1. Encoder configuration options for control of coding tools. 3](#_Toc27473186)

[Table 3‑2. Encoder configuration options for slice coding. 4](#_Toc27473187)

[Table 3‑3. Encoder configuration options for motion search. 4](#_Toc27473188)

[Table 3‑4. Encoder configuration options for mode decisions. 5](#_Toc27473189)

[Table 3‑5. Encoder configuration options for quantization parameter control. 5](#_Toc27473190)

[Table 3‑6. Encoder configuration options for pre-filtering input frames. 6](#_Toc27473191)

[Table 3‑7. Encoder configuration options for rate control. 6](#_Toc27473192)

[Table 3‑8. Encoder configuration options for motion Vector Estimation hints 7](#_Toc27473193)

[Table 4‑1. Mapping between intra prediction direction and intra prediction mode for chroma. 14](#_Toc27473194)

[Table 4‑2. Specification of predefined threshold for various transform block sizes. 14](#_Toc27473195)

[Table 4‑3. Specification of intra prediction mode for 4:2:2 chroma. 15](#_Toc27473196)

[Table 4‑4. 8-tap DCT-IF coefficients for 1/4th luma interpolation. 22](#_Toc27473197)

[Table 4‑5. 4-tap DCT-IF coefficients for 1/8th chroma interpolation. 23](#_Toc27473198)

[Table 4‑6. g\_maxTrDynamicRange[channel]. 27](#_Toc27473199)

[Table 4‑7. α Mapping Table. 29](#_Toc27473200)

[Table 4‑8. Derivation of threshold variables from input Q. 32](#_Toc27473201)

[Table 4‑9. Specification of SAO type. 34](#_Toc27473202)

[Table 4‑10. Pixel classification rule for EO. 35](#_Toc27473203)

[Table 5‑1. Bitstream indications for format range extensions profiles. 36](#_Toc27473204)

[Table 6‑1. Derivation of Wk. 41](#_Toc27473205)

[Table 6‑2. Fast encoder mode summary. 48](#_Toc27473206)

[Table 6‑3. Conditions and actions for fast AMP mode evaluation. 50](#_Toc27473207)

[Table 6‑4. Values for *ωCurrPic* for random access GOP 8-picture configuration. 57](#_Toc27473208)

[Table 6‑5. Values for *ωCurrPic* for low delay GOP configuration. 57](#_Toc27473209)

[Table 6‑6. Configuration options for rate control algorithm. 58](#_Toc27473210)

[Table 6‑7: Values for *lambdaRatio* used to derive the *bitsRatio* for a GOP. 60](#_Toc27473211)

[Table 6‑8. 3D motion vector derivation process based in 3D information 69](#_Toc27473212)

# List of acronyms

BO Band Offset

CABAC Context Adaptive Arithmetic Coding

CBF Coded Block Flag

CCP Cross Component Prediction

CTC Common Test Conditions

CTU Coding Tree Unit

CU Coding Unit

DPB Decoded Picture Buffer

EO Edge Offset

GOP Group Of Pictures

IDR Instantaneous Decoding Refresh

MPM Most Probable Mode

NAL Network Abstraction Layer

PU Prediction Unit

QP Quantization Parameter

RDO Rate Distortion Optimization

RDOQ Rate Distortion Optimized Quantization

RDPCM Residual Differential Pulse Code Mode

RQT Residual QuadTree

TS Transform Skip

TU Transform Unit

SAO Sample Adaptive Offset

# Introduction

This document provides an overview of the coding tools defined in HEVC Version 1 and the Range Extension (RExt) of HEVC Version 2, both of which are implemented in the HEVC test model (HM) software. In addition, some of the algorithms that the HM encoder uses to control these tools and user-controlled configuration options are also described.

## Overview of coding structures

HEVC has a block-based hybrid coding architecture, combining inter and intra prediction and transform coding with high efficiency entropy coding. However, in contrast to previous video coding standards, HEVC employs a quadtree coding block partitioning structure that enables a flexible use of large and small coding, prediction, and transform blocks. HEVC also allows for improved intra prediction and coding, adaptive motion parameter prediction and coding, a new loop filter and an enhanced version of context-adaptive binary arithmetic coding (CABAC) entropy coding over that defined by previous standards. New high-level structures have also been designed to aid parallel processing.

Figure 1‑1 shows a (simplified) general block diagram of the HM encoder.



Figure 1‑1. Simplified block diagram of HM encoder.

The picture partitioning structure, which is further described in Section 4.2, divides the input video into blocks called coding tree units (CTUs). These CTUs have a role that is broadly analogous to that of macroblocks in previous standards. A CTU is split using a quadtree into coding units (CUs), with a leaf coding unit (CU) defining a region sharing the same prediction mode (intra, inter or skip). The leaf CU also defines the shape of prediction units (PUs) present, with each PU detailing the prediction information to be used for the respective picture region. Leaf CUs also define another quadtree that defines the residual-quadtree (RQT) containing transform units (TUs), which define a region sharing the same transformation and quantization process. The term ‘unit’ defines a region of an image covering all components; the term ‘block’ is used to define a region covering a particular component (e.g. luma), and may differ in spatial location when considering chroma sub-sampling such as 4:2:0.

The intra prediction processes, which are further described in Section 4.3, provide 35 modes (Planar, DC and 33 angular directions) for each prediction block. A luma intra prediction mode is selected for a luma prediction block and a chroma intra prediction mode is selected for the chroma prediction blocks. Mode-dependent reference and prediction sample smoothing is applied to increase prediction efficiency and intra prediction mode is coded using either one of the three most probable modes (MPM) or one of the 32 remaining modes.

The inter picture prediction processes, which are further described in Section 4.4, select the motion parameters, which includes the option of a skip mode, a merge mode, or a regular inter prediction mode with motion vectors being coded relative to predictors and able to describe spatial offsets with ¼ pixel accuracy to select a reference block from a reference picture.

The transform and quantization processes, which are further described in Section 4.5, take the residuals generated by subtracting the prediction from the encoder input and spatially transform and quantize them. In the transform process, matrices which are approximations to the DCT are used. In the case of 4x4 intra predicted residuals, an approximation to DST is used for the luma residual. For 8-bit video, 52-level quantization steps are permitted (the number of steps increases by 6 for each additional video bit depth). Reconstructed samples are created by inverse quantization and inverse transform.

Entropy coding, which is described in Section 4.7, is applied to the generated symbols and quantized transform coefficients in the encoding process using a Context-based Adaptive Binary Arithmetic Coding (CABAC) process.

Loop filtering is described in Section 4.10, which applies two in-loop filtering processes (namely deblocking filtering and sample adaptive offset (SAO) filtering) after the reconstructed pixel data is formed. The resulting image is stored in the decoded picture buffer (DPB) and may be used for inter coding predictions for future frames in coding order.

## Obtaining the HEVC test model software

The current version of the 16th HEVC test model software (HM-16.21) is available from the following location:

<https://vcgit.hhi.fraunhofer.de/jct-vc/HM>

## Obtaining the HEVC standard

The HEVC standard may be obtained from the ITU and ISO/IEC Standards setting organizations:

<http://www.itu.int/rec/T-REC-H.265>

<http://www.iso.org/iso/home/search.htm?qt=23008-2&published=on&active_tab=standards&sort_by=rel>

# Scope

This document provides an encoder-side description of the HEVC test model 16 (HM), serving as a tutorial on the encoding algorithms implemented in the HM software. The purpose of this text is to establish a common understanding on reference encoding methods supported in the HM software, in order to facilitate the assessment of the technical impact of proposed new technologies during the HEVC standardization process. Although brief descriptions of the HEVC design are provided to help understanding of the HM, the corresponding sections of the HEVC specification [2] should be referred to for any descriptions regarding normative processes. Document [1] provides a summary of configuration options for the HM encoder, without going into detail of encoder algorithms. The document [3] defines the common test conditions and software reference configurations that should be used for experimental work.

# Encoder control

## Encoder configuration options

### File, format and profile/level/tier configuration options

Configuration options for the HEVC test model software associated with files, formats and profiles/levels/tiers are described in the JCTVC HM software manual [1].

### Coding tool configuration options

Table 3‑1 provides a list of encoder configuration options for the HEVC test model associated with coding tool configuration and enablement.

Table 3‑1. Encoder configuration options for control of coding tools.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| AlignCABACBeforeBypass | 4.7.1 |
| ChromaFormatIDC | 4.1.1 |
| CrossComponentPrediction | 4.9 |
| ExplicitResidualDPCM | 4.6 |
| ExtendedPrecision | 4.5.8 |
| GolombRiceParameterAdaptation | 4.8.3 |
| HighPrecisionPredictionWeighting | 4.4.4.1 |
| ImplicitResidualDPCM | 4.6 |
| IntraReferenceSmoothing | 4.3.2 |
| LowerBitRateConstraintFlag | 5 |
| MaxBitDepthConstraint | 5 |
| MaxCUChromaQpAdjustmentDepth | 4.5.9 |
| MaxCUWidth | 4.2.3 |
| MaxCUHeight | 4.2.3 |
| MaxCUSize | 4.2.3 |
| MaxPartitionDepth | 4.2.3 |
| QuadtreeTULog2MaxSize | 4.2.5 |
| QuadtreeTULog2MinSize | 4.2.5 |
| QuadtreeTUMaxDepthIntra | 4.2.5 |
| QuadtreeTUMaxDepthInter | 4.2.5 |
| ReconBasedCrossCPredictionEstimate | 4.9 |
| ResidualRotation | 4.8.1 |
| SaoLumaOffsetBitShift | 4.10.3 |
| SaoChromaOffsetBitShift | 4.10.3 |
| SingleSignificanceMapContext | 4.8.2 |
| TransformSkipLog2MaxSize | 4.5 |
| WaveFrontSynchro | 4.11 |

### Tile and slice coding parameters

Table 3‑2. Encoder configuration options for slice coding.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| SliceMode | 6.4 |
| SliceArgument | 6.4 |
| SliceSegmentMode | 6.4 |
| SliceSegmentArgument | 6.4 |
| TileUniformSpacing | 6.4 |
| NumTileColumnsMinus1 | 6.4 |
| NumTileRowsMinus1 | 6.4 |
| TileColumnWidthArray | 6.4 |
| TileRowHeightArray | 6.4 |

### Motion search options

Table 3‑3. Encoder configuration options for motion search.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| DisableIntraInInter | 6.5.2 |
| FastSearch | 6.5.2 |
| SearchRange | 6.5.2 |
| BipredSearchRange | 6.5.2 |
| MinSearchWindow | 6.5.2 |
| RestrictMESampling | 6.5.2 |
| ClipForBiPredMEEnabled | 6.5.2 |
| FastMEAssumingSmootherMVEnabled | 6.5.2 |
| HadamardME | 6.5.2 |
| ASR | 6.5.2 |

### Mode cost evaluation parameters

Table 3‑4. Encoder configuration options for mode decisions.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| CostMode | 6.2 |
| LambdaModifier0 | 6.3.5 |
| LambdaModifier1 | 6.3.5 |
| LambdaModifier2 | 6.3.5 |
| LambdaModifier3 | 6.3.5 |
| LambdaModifier4 | 6.3.5 |
| LambdaModifier5 | 6.3.5 |
| LambdaModifier6 | 6.3.5 |
| LambdaModifierI | 6.3.5 |
| IQPFactor | 6.3.5 |

### Quantization parameters

Table 3‑5. Encoder configuration options for quantization parameter control.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| QP,q | 6.8 |
| QPIncrementFrame,qpif | 6.8 |
| IntraQPOffset | 6.8 |
| MaxDeltaQP,d | 6.8 |
| MaxCuDQPDepth,dqd | 6.8 |
| MaxCUChromaQpAdjustmentDepth | 6.8 |
| FastDeltaQP | 6.8 |
| LumaLevelToDeltaQPMode | 6.8 |
| LumaLevelToDeltaQPMaxValWeight | 6.8 |
| LumaLevelToDeltaQPMappingDQP | 6.8 |
| CbQpOffset,-cbqpofs | 6.8 |
| CrQpOffset,-crqpofs | 6.8 |
| WCGPPSCbQpScale | 6.8 |
| WCGPPSCrQpScale | 6.8 |
| WCGPPSChromaQpScale | 6.8 |
| WCGPPSChromaQpOffset | 6.8 |
| SliceChromQPOffsetPeriodicity | 6.8 |
| SliceCbQpOffsetIntraOrPeriodic | 6.8 |
| SliceCrQpOffsetIntraOrPeriodic | 6.8 |
| AdaptiveQpSelection,-aqps | 6.8 |
| AdaptiveQP,-aq | 6.8 |
| MaxQPAdaptationRange,-aqr | 6.8 |
| dQPFile,m | 6.8 |
| RDOQ | 6.8 |
| RDOQTS | 6.8 |
| SelectiveRDOQ | 6.8 |
| RDpenalty | 6.8 |
| ScalingList | 4.5.4 |
| ScalingListFile | 4.5.4 |

### Pre-filtering control parameters

Table 3‑6. Encoder configuration options for pre-filtering input frames.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| TemporalFilter | 6.11 |
| TemporalFilterFutureReference | 6.11 |
| TemporalFilterStrengthFrame | 6.11 |

### Rate control parameters

Table 3‑7. Encoder configuration options for rate control.

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| RateControl | 6.9 |
| TargetBitrate | 6.9 |
| KeepHierarchicalBit | 6.9 |
| LCULevelRateControl | 6.9 |
| RCLCUSeparateModel | 6.9 |
| InitialQP | 6.9 |
| RCForceIntraQP | 6.9 |
| RCCpbSaturation | 6.9 |
| RCCpbSize | 6.9 |
| RCInitialCpbFullness | 6.9 |

### SEI message configuration options

SEI message configuration options are described in [1].

### Supplemental Motion Vector Estimation hint configuration options

*Note: This section describes functionality anticipated for a future release of HM.*

Table 3‑8. Encoder configuration options for motion vector estimation hints

|  |  |
| --- | --- |
| **Configuration option** | **Section reference** |
| UsePccMotionEstimation | 6.12 |
| BlockToPatchFile | 6.12 |
| OccupancyMapFile | 6.12 |
| PatchInfoFile | 6.12 |

# Overview of tools in HEVC Version 1 and HEVC Version 2 RExt

## High level syntax

This section summarizes the main features of the high level syntax of HEVC. A more detailed overview of the topic is provided in a tutorial paper [5].

### Bitstream organization

Any bitstream compliant with HEVC is organized into network abstraction layer (NAL) units, which are self-contained packets that allow the video layer to be identical for different transmission environments. Each NAL unit should not exceed the maximum transfer unit size associated with the transmission environment and can be of two types: video coding layer (VCL) and non-video coding layer (non-VCL). The former type carries information associated with coded video data while the latter contains data shared by different pictures. A NAL unit consists of a header, followed by the NAL unit payload. The header has a fixed two byte length, facilitating processing by media aware network elements. Finally, each NAL unit is associated with a particular temporal layer, as indicated by the TemporalId. HEVC mandates that all NAL units associated with a particular picture have the same TemporaId, hence a picture has one and only one TemporalId. Moreover, the syntax of HEVC prohibits any data dependency on data belonging to a higher temporal sublayer when decoding the data associated with a lower temporal sublayer. This restriction is needed to support temporal scalability, as a decoder capable of decoding at a given frame rate is expected to discard NAL units associated with intermediate pictures used for higher frame rate decoding.

### Parameter sets

HEVC specifies parameter sets to allow for bitstream robustness over unreliable transmission links. Parameter sets contain information associated with a frame, a coded video sequence (CVS) or shared among several layers. Each parameter set is contained in a non-VCL NAL unit and can be duplicated or transmitted via a separate, reliable, channel to improve bitstream error robustness. HEVC specifies three parameter sets:

* **Video parameter set (VPS)**: Contains information applicable to multiple layers, avoiding replication in each layer. Examples of information contained in the VPS are: profile, level and Hypothetical Reference Decoder (HRD) parameters.
* **Sequence parameter set (SPS)**: Contains information applicable to all slices of a CVS. Examples of information conveyed by the SPS are: picture size, profile and level.
* **Picture parameter set (PPS)**: Contains information which may vary on a per-picture basis. Examples of information contained in the PPS are: quantization parameter and flags indicating the use of particular coding tools (e.g. Transform Skip).

Parameter sets may reference other parameter sets, specifically, a PPS has an ID indicating the associated SPS and an SPS has an ID indicating the associated VPS. Regardless of these associations, to facilitate parsing robustness, each parameter set can be parsed independently, i.e. there is no conditional dependency in the syntax parsing on information present in any associated parameter set.

### Picture types

Random access functionality is provided using intra random access point (IRAP) pictures. An IRAP picture can only contain one or more I-slices. HEVC defines three types of IRAP pictures. These three picture types are:

* Instantaneous decoding refresh (IDR)
* Clean random access (CRA)
* Broken link access (BLA)

An IDR picture, when encountered, results in flushing of the decoded picture buffer (DPB). IDR pictures provide RAP functionality but sacrifice coding performance because frames decoded prior to an IDR picture are no longer available for reference in inter coding. In order to allow random access to the content and maintain the coding performance, HEVC defines CRA pictures. CRA pictures are intra coded but, when encountered, do not empty the DPB. Consequently, pictures following a CRA picture in decoding order can still use reference pictures that precede the CRA picture in decoding order. Leading pictures may follow a CRA picture; leading pictures can either be decoded or skipped. Pictures following a CRA in decoding order and correctly decodable are called random access decodable leading (RADL) pictures. Pictures that follow a CRA picture in decoding order but cannot be correctly decoded without preceding reference frames having also been decoded are called random access skipped leading (RASL) pictures.

One example use case for this functionality is splicing bitstreams to insert advertisements in a television programme. Consider the case when Bitstream 1 (B1) and Bitstream 2 (B2) are concatenated as B1·B2 and the picture which starts the segment associated to B2 is a CRA. All RASL pictures following the CRA picture in decoding order in B2 cannot be correctly decoded because their associated reference pictures are not present in the DPB. These RASL pictures should be discarded from the decoder output and this is accomplished by the splicing operation declaring the CRA picture in B2 to be a BLA picture. In this case the decoder knows that all RASL associated with this BLA picture will not be displayed.

Finally, HEVC also defines two additional types of pictures to support temporal scalability:

* Temporal sublayer access (TSA)
* Step-wise temporal sub-layer access (STSA)

These pictures impose restrictions on the reference used between different temporal layers so that temporal down-switching and up-switching operations can be made possible (see Figure 5 in [5] for an example on the use of TSA and STSA pictures).

### Reference picture set

The reference picture set (RPS) has been introduced in HEVC to handle reference pictures in the DPB. In fact, when a picture is no longer used for reference by other pictures, it should be discarded from the DPB. If instead a picture is used as reference for future pictures it must be kept in the DPB to correctly decode the bitstream. The RPS contains information on the status of the DPB and may be signalled in the SPS, and additionally signalled, or overridden, in the slice header. The signalling is absolute, i.e. each RPS describes the DPB status and does not refer to any previous status for its description. In this way, bitstream error resilience is improved even when some NAL units are lost.

## Picture partitioning

### Coding tree unit (CTU) partitioning

Pictures are divided into a sequence of coding tree units (CTUs), all being the same size, and each covering a square pixel region of the picture, except at the right and bottom edge of a picture. At the right edge and bottom edge of a picture CTUs may be forcibly quadtree split to ensure CUs to not span the picture boundary, and only those CUs within the picture boundary are coded. An example of a picture divided into CTUs is shown in Figure 4‑1. The size of a CTU is specified with respect to the luma channel, to prevent ambiguity when considering chroma formats.

The size of the CTU is configured as one of 16×16, 32×32 or 64×64 luma samples.



Figure 4‑1. Example of a picture divided into CTUs.

### Slice and tile structures

A slice is a data structure that can be decoded independently from other slices of the same picture, in terms of entropy coding, signal prediction, and residual signal reconstruction. A slice can either be the entire picture or a region of a picture, which is not necessarily rectangular. A slice consists of a sequence of one or more slice segments starting with an independent slice segment and containing all subsequent dependent slice segments (if any) that precede the next independent slice segment (if any) within the same access unit.

A slice segment consists of a sequence of CTUs. An independent slice segment is a slice segment for which the values of the syntax elements of the slice segment header are not inferred from the values for a preceding slice segment. A dependent slice segment is a slice segment for which the values of some syntax elements of the slice segment header are inferred from the values for the preceding independent slice segment in decoding order. For dependent slice segments, prediction can be performed across dependent slice segment boundaries, and entropy coding is not initialized at the starting of the dependent slice segment parsing process.

An example of picture with 11 by 9 CTUs that is partitioned into two slices is shown in Figure 4‑2, below. In this example, the first slice is composed of an independent slice segment containing 4 CTUs, a dependent slice segment containing 32 CTUs, and another dependent slice segment containing 24 CTUs. The second slice consists of a single independent slice segment containing the remaining 39 CTUs of the picture.



Figure 4‑2. Example of slices and slice segments.

A tile is a rectangular region containing an integer number of CTUs. CTUs are ordered in raster scan within a tile, and tiles in a picture are ordered consecutively in a raster scan of the tiles of the picture. This defines the coding order of CTUs, which is referred to as the tile scan order.

A tile may consist of CTUs contained in more than one slice. Similarly, a slice may consist of CTUs contained in more than one tile. Note that within the same picture, there may be both slices that contain multiple tiles and tiles that contain multiple slices. However, one or both of the following conditions must be fulfilled for each slice and tile:

– All coding tree units in a slice belong to the same tile.

– All coding tree units in a tile belong to the same slice.

In addition, one or both of the following conditions must be fulfilled for each slice segment and tile:

– All coding tree units in a slice segment belong to the same tile.

– All coding tree units in a tile belong to the same slice segment.

Two examples of possible slice and tile structures for a picture with 11 by 9 coding tree units are shown in Figure 4‑3, below. In both examples, the picture is partitioned into two tiles, separated by a vertical tile boundary. The left-hand example shows a case in which the picture only contains one slice, starting with an independent slice segment and followed by four dependent slice segments. The right-hand example illustrates an alternative case in which the picture contains two slices in the first tile and one slice in the second tile.



Figure 4‑3. Examples of tiles and slices.

### Coding unit (CU) and coding tree structure

The coding unit (CU) is a square region (in terms of pixels/luma-samples) and is a leaf node in a ‘coding tree’, where the CTU is subject to a quadtree partitioning into one or more CUs. The quadtree partitioning structure allows recursive splitting into four equally sized nodes, starting from the CTU and stopping when no further splitting is signalled in the bit stream (as determined by an encoder) or when the minimum CU size is reached. The minimum CU size is configured in the SPS to be 32×32, 16×16, or 8×8 luma samples; 8×8 is used in the common test conditions [3] for HEVC development. Each leaf node CU is configured to use a particular prediction mode, that being either intra prediction or inter-prediction. Figure 4‑4 shows a CTU divided into multiple CUs.



Figure 4‑4. Example of coding tree structure.

The quadtree partitioning thus allows a content-adaptive coding tree structure comprised of CUs, each of which may be as large as the CTU or as small as 8×8.

The CTU size and minimum CU size are configured using the syntax elements *log2\_min\_luma\_coding\_block\_size\_minus3* and *log2\_diff\_max\_min\_luma\_coding\_block\_size*. The HM encoder sets these syntax elements according to MaxCUWidth, MaxCUHeight, MaxCUSize and MaxPartitionDepth. The maximum partition depth is defined as the maximum number of splits in the CU quadtree and one split resulting from dividing a CU into partitions for prediction (to be described below). Thus, a CTU size of 64×64 and a maximum partition depth of 4 implies a minimum CU size of 8×8 (3 quadtree splits) with one partitioning of the CU.

### Prediction unit (PU) structure

Each leaf CU is associated with one or more prediction units (PU) according to a partition mode and all PUs associated with a given CU have the same prediction mode. Each CU includes one, two or four PUs, depending on the partition mode of the CU. Figure 4‑5 shows the eight partition modes that may be used to define the PUs for a CU.



Figure 4‑5. 8 partition modes for inter PU.

For a CU configured to use intra prediction, only square PUs are available and thus only partition modes PART\_NxN and PART\_2Nx2N are available; the latter is only available when the CU size is equal to the configured minimum CU size.

For a CU configured to use inter-prediction, the following partition modes are always available: PART\_2Nx2N, PART\_2NxN, and PART\_Nx2N. When the CU is larger than the minimum CU size (MinCbLog2SizeY) then non-square PU partitionings PART\_2NxnU, PART\_2NxnD, PART\_nLx2N, and PART\_nRxN are also available. The availability of non-square PU sizes permits improved matching of boundaries of real objects in the picture. The PART\_NxN partition mode is only available when the CU size is equal to MinCbLog2SizeY and the CU size is larger than 8×8, preventing the occurance of 4×4 PUs, contributing to reducing worst-case memory bandwidth of motion compensation. Also, for reduction of worst-case memory bandwidth of motion compensation, 4×8 and 8×4 PU sizes may only use one reference picture. Other PU sizes can reference one or two reference pictures.

For inter prediction, a PU spans all colour channels and is generally associated with one prediction block (PB) for each colour channel, with motion parameters common to all colour channels. For intra prediction, separate intra prediction modes are coded for the luma PB vs the chroma PBs. An 8×8 CU is divided into 4×4 PUs in each colour channel when PART\_NxN is used. As a consequence, when a 4:2:0 or a 4:2:2 chroma format is used, the boundaries of luma PUs within the 8×8 CU are not all aligned to chroma PU boundaries. For the 4:2:0 chroma format, an 8×8 CU that is split according to the PART\_NxN partition mode will have four luma 4×4 PBs, but only one 4×4 PB per chroma channel for intra coded blocks. Similarly, for the 4:2:2 chroma format, there will be two 4×4 PBs per chroma channel for intra coded blocks. Fortunately for this exception, it will be seen that the transform block (TB) term is used instead.

### Transform unit (TU) and transform tree structure

The transform unit (TU) is a square region of size 8×8, 16×16 or 32×32 luma samples/pixels defined by a quadtree partitioning of a CU. The quadtree partitioning of the CU into one or more TUs is known as a ‘residual quadtree’ (RQT). In general, each TU is associated with one transform block (TB) per colour channel. However, for the 4:2:2 chroma format (permitted in some RExt profiles) there must always be two TBs per chroma channel in order to ensure that the TBs are square arrays of samples (to match the defined transforms). In addition, in all chroma formats, the 8x8 TU may be split into four 4×4 luma TBs, but for the 4:2:0 format, the 8×8 TU will continue to have just one 4x4 TB (occupying 8×8 pixels) per chroma channel; similarly two 4×4 TBs (each occupying 8×4 pixels) TBs per chroma channel for 4:2:2. For 4:4:4, there are four 4×4 TBs (each occupying 4×4 pixels) for each chroma channel. Figure 4‑6 shows an example RQT.



Figure 4‑6. Example of transform tree structure within CU.

The range of supported transform sizes is signalled in the bitstream using *log2\_min\_luma\_transform\_block\_size\_minus2* and *log2\_diff\_max\_min\_luma\_transform\_block\_size*. These values are specified using QuadtreeTULog2MaxSize and QuadtreeTULog2MinSize.

The maximum depth of the RQT is signalled in the bitstream independently for inter-predicted CUs and intra-predicted CUs using *max\_transform\_hierarchy\_depth\_inter* and *max\_transform\_hierarchy\_depth\_intra*, respectively in the SPS. These values are controlled using QuadtreeTUMaxDepthInter and QuadtreeTUMaxDepthIntra, respectively, in the HM encoder.

For a CU configured to use inter-prediction, PU boundaries may occur within a given TU. For a CU configured to use intra prediction, PU boundaries cannot occur within a given TU, except for when an 8×8 TU is split into four 4×4 luma TBs (which effectively makes a virtual TU of 4×4 luma samples, with 4×4 luma samples also being the smallest PU size).

## Intra prediction

### Prediction modes

Intra prediction involves producing samples for a given TB using samples previously reconstructed in the considered colour channel. Regardless of the RQT, the intra prediction mode is separately once for each luma partition of a CU (i.e. once for PART\_2Nx2N and four times for PART\_NxN) and once for common application to the chroma channels in 4:2:2 and 4:2:0 or four times for application to each pair of chroma blocks when 4:4:4 is used. A chroma channel intra prediction mode is optionally dependant on the intra prediction mode of the collocated luma block via the ‘DM\_CHROMA’ mode. Although the intra prediction mode is signalled at the PB level, the intra prediction and reconstruction process is applied at the TB level, in accordance with the residual quadtree hierarchy for the CU. Accordingly, the residual of one TB contributes to the reference samples of later TBs within the CU, improving the accuracy of the later PBs within the CU.

HEVC includes 35 intra prediction modes – a DC mode, a planar mode and 33 directional, or ‘angular’ intra prediction modes. The 33 angular intra prediction modes are illustrated in Figure 4‑7 below.



Figure 4‑7. The 33 intra prediction directions.

The mapping between the direction of each of the angular intra prediction modes and the intra prediction mode number is specified in Figure 4‑8, below.



Figure 4‑8. Mapping between intra prediction direction and intra prediction mode.

For PBs associated with chroma colour channels, the intra prediction mode is specified as either planar, DC, horizontal, vertical, ‘DM\_CHROMA’ mode or sometimes diagonal mode ‘34’. Table 4‑1 shows the rule specifying the chroma colour channel PB intra prediction mode given the luma colour channel PB intra prediction mode and the ‘*intra\_chroma\_pred\_mode*’ syntax element.

Note for chroma formats 4:2:2 and 4:2:0, the chroma PB may overlap two or four (respectively) luma PBs; in this case the luma direction for DM\_CHROMA is taken from the top left of these luma PBs.

The DM\_CHROMA mode indicates that the intra prediction mode of the luma colour channel PB is applied to the chroma colour channel PBs. Since this is relatively common, the binarization of intra\_chroma\_pred\_mode allocates a short bin string to DM\_CHROMA.

Table 4‑1. Mapping between intra prediction direction and intra prediction mode for chroma.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **intra\_chroma\_pred\_mode** | **Luma intra prediction direction, X** | | | | |
| **0** | **26** | **10** | **1** | **Otherwise ( 0 <= X <= 34 )** |
| 0 | 34 | 0 | 0 | 0 | 0 |
| 1 | 26 | 34 | 26 | 26 | 26 |
| 2 | 10 | 10 | 34 | 10 | 10 |
| 3 | 1 | 1 | 1 | 34 | 1 |
| 4 (DM\_CHROMA) | 0 | 26 | 10 | 1 | X |

### Filtering of neighbouring samples

The neighbouring samples filtering process for intra prediction is skipped when *intra\_smoothing\_disabled\_flag* is set to 1 (as configured by the IntraReferenceSmoothing enable flag). The intra reference smoothing filter is disabled in common test conditions [5] only when sequence-level lossless coding is used.

If the intra reference smoothing filter is enabled, then for the luma component, the neighbouring samples used for generation of intra-predicted samples are filtered. The filtering further is controlled by the given intra prediction mode and transform block size. If the intra prediction mode is DC or the transform block size is equal to 4×4, neighbouring samples are not filtered. If the distance between the given intra prediction mode and vertical mode (or horizontal mode) is larger than predefined threshold, the filtering process remains enabled (otherwise the filtering process becomes disabled). The predefined threshold is specified in Table 4‑2, where nT represents the TB size.

Table 4‑2. Specification of predefined threshold for various transform block sizes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **nT = 8** | **nT = 16** | **nT = 32** |
| **Threshold** | 7 | 1 | 0 |

If filtering remains enabled, then either a neighbouring sample filtering, [1, 2, 1] or a bi-linear filter are used. The bi-linear filtering is used if all of the following conditions are true (otherwise the neighbouring sample filtering is used):

– strong\_intra\_smoothing\_enabled\_flag is equal to 1

– luma channel under consideration

– transform block size is equal to 32

– Abs( p[ −1 ][ −1 ] + p[ nT\*2−1 ][ −1 ] − 2\*p[ nT−1 ][ −1 ] ) < (1 << ( BitDepthY − 5 ))

– Abs( p[ −1 ][ −1 ] + p[ −1 ][ nT\*2−1 ] − 2\*p[ −1 ][ nT−1 ] ) < (1 << ( BitDepthY − 5 ))

### Intra boundary filter

When reconstructing intra-predicted TBs an intra-boundary filter (IBF) may be used when predicting samples along the left and/or top edges of the TB for PBs using horizontal, vertical and DC intra prediction modes, as shown in Figure 4‑9. For horizontal and vertical intra prediction modes, the IBF is disabled when implicit RDPCM and transquant bypass are enabled. For the DC intra prediction mode, the IBF is applied to the luma channel of TBs smaller than 32×32.



Figure 4‑9. Intra boundary filter example.

The intra boundary filter is defined with respect to an array of predicted samples p as input and predSamples as output as follows:

– For horizontal intra-prediction applied to luma transform blocks of size less than 32×32, and disableIntraBoundaryFilter is equal to 0, the following filtering applies with x = 0..nTbS − 1, y = 0:

predSamples[ x ][ y ] = Clip1Y( p[ −1 ][ y ] + ( ( p[ x ][ −1 ] − p[ −1 ][ −1 ] )  >>  1 ) )

– For vertical intra-prediction applied to luma transform blocks of size less than 32x32, and disableIntraBoundaryFilter is equal to 0, the following filtering applies with x = 0..nTbS − 1, y = 0:

predSamples[ x ][ y ] = Clip1Y( p[ x ][ −1 ] + ( ( p[ −1 ][ y ] − p[ −1 ][ −1 ] )  >>  1 ) )

– For DC intra-prediction applied to luma transform blocks of size less than 32x32 the following filtering applies with x = 0..nTbS − 1, y = 0 (where dcVal is the DC predictor):

predSamples[ 0 ][ 0 ] = ( p[ −1 ][ 0 ] + 2 \* dcVal + p[ 0 ][ −1 ] + 2 )  >>  2

predSamples[ x ][ 0 ] = ( p[ x ][ −1 ] + 3 \* dcVal + 2 )  >>  2, with x = 1..nTbS − 1

predSamples[ 0 ][ y ] = ( p[ −1 ][ y ] + 3 \* dcVal + 2 )  >>  2, with y = 1..nTbS − 1

### 4:2:2 chroma format mode adjustment

When the 4:2:2 chroma format is in use, the intra prediction mode for a chroma PB (**intra\_chroma\_pred\_mode**) is coded in the same manner as for 4:2:0 and 4:4:4 formats as set out in section 4.3.1, except that a mapping is applied to the final modes that accounts for the non-square sample aspect ratio. The mapping is as specified in Table 4‑3.

Table 4‑3. Specification of intra prediction mode for 4:2:2 chroma.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **intra pred mode** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| **intra pred mode for 4:2:2 chroma** | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 18 | 18 | 18 |
| **intra pred mode** |  | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** |
| **intra pred mode for 4:2:2 chroma** |  | 22 | 22 | 23 | 23 | 24 | 24 | 25 | 25 | 26 | 27 | 27 | 28 | 28 | 29 | 29 | 30 | 30 |

The mapping between luma intra prediction mode and chroma intra prediction mode is illustrated in Figure 4‑10 and Figure 4‑11. Figure 4‑10 shows the 33 angular intra prediction modes (modes 2-34) for an example 16x16 PB in the luma channel.



Figure 4‑10. Intra prediction directions in luma for example 16x16 PB.

A luma PB is collocated with a pair of chroma PBs when the 4:2:2 chroma format is in use. Figure 4‑11 shows the example of a pair of 8x8 chroma PBs, with the 34 angular intra prediction modes shown for the upper PB. As the upper PB occupies only half of the area occupied by the corresponding luma PB, the angular intra prediction process results in different directions in chroma compared to luma. When an intra prediction mode is selected for 4:2:2, the mapping in Table 4‑3 results in the 33 angular intra prediction modes available for luma being mapped onto a subset of 24 of the 33 defined angular intra prediction modes (blue arrows). The remaining nine angular intra prediction modes (dark grey arrows) are not available for use in chroma PBs for 4:2:2.



Figure 4‑11. Intra prediction modes for chroma PBs in the 4:2:2 chroma format.

## Inter prediction

### Prediction modes

Each inter-predicted PU has motion parameters for one or two reference picture lists. Motion parameters include a motion vector and a reference picture index. Usage of one of the two reference picture lists may also be signalled using *inter\_pred\_idc*. Motion vectors may be explicitly coded as deltas relative to predictors.

When a CU is coded with skip mode, one PU is associated with the CU, and there are no significant residual coefficients, no coded motion vector delta or reference picture index. A merge mode is specified whereby the motion parameters for the current PU are obtained from neighbouring PUs, including spatial and temporal candidates. The merge mode can be applied to any inter-predicted PU, not only for skip mode. The alternative to merge mode is the explicit transmission of motion parameters, where motion vector, corresponding reference picture index for each reference picture list and reference picture list usage are signalled explicitly per each PU.

When signalling indicates that one of the two reference picture lists is to be used, the PU is produced from one block of samples. This is referred to as ‘uni-prediction’. Uni-prediction is available both for P-slices and B-slices.

When signalling indicates that both of the reference picture lists are to be used, the PU is produced from two blocks of samples. This is referred to as ‘bi-prediction’. Bi-prediction is available for B-slices only.

The following text provides the details on the inter prediction modes specified in HEVC. The description will start with the merge mode.

#### Derivation of candidates for merge mode

When a PU is predicted using merge mode, an index pointing to an entry in the *merge candidates list* is parsed from the bitstream and used to retrieve the motion information. The construction of this list is specified in the HEVC standard and can be summarized according to the following sequence of steps:

* Step 1: Initial candidates derivation
  + Step 1.1: Spatial candidates derivation
  + Step 1.2: Redundancy check for spatial candidates
  + Step 1.3: Temporal candidates derivation
* Step 2: Additional candidates insertion
  + Step 2.1: Creation of bi-predictive candidates
  + Step 2.2: Insertion of zero motion candidates

These steps are also schematically depicted in Figure 4‑12. For spatial merge candidate derivation, a maximum of four merge candidates are selected among candidates that are located in five different positions. For temporal merge candidate derivation, a maximum of one merge candidate is selected among two candidates. Since constant number of candidates for each PU is assumed at decoder, additional candidates are generated when the number of candidates does not reach the maximum number of merge candidate (MaxNumMergeCand) which is signalled in slice header. Since the number of candidates is constant, index of best merge candidate is encoded using truncated unary binarization (TUB). If the size of CU is equal to 8, all the PUs of the current CU share a single merge candidate list, which is identical to the merge candidate list of the 2N×2N prediction unit.

In the following, the operations associated with the aforementioned steps are detailed.



Figure 4‑12. Derivation process for merge candidates list construction.

#### Spatial candidates derivation

In the derivation of spatial merge candidates, a maximum of four merge candidates are selected among candidates located in the positions depicted in Figure 4‑13. The order of derivation is A1, B1, B0, A0 and B2. Position B2 is considered onlywhen any PU of position A1, B1, B0, A0 is not available (e.g. because it belongs to another slice or tile) or is intra coded. After candidate at position A1 is added, the addition of the remaining candidates is subject to a redundancy check which ensures that candidates with same motion information are excluded from the list so that coding efficiency is improved. To reduce computational complexity, not all possible candidate pairs are considered in the mentioned redundancy check. Instead only the pairs linked with an arrow in Figure 4‑14are considered and a candidate is only added to the list if the corresponding candidate used for redundancy check has not the same motion information. Another source of duplicate motion information is the “*second PU*” associated with partitions different from 2Nx2N. As an example, Figure 4‑15 depicts the second PU for the case of N×2N and 2N×N, respectively. When the current PU is partitioned as N×2N, candidate at position A1 is not considered for list construction. In fact, by adding this candidate will lead to two prediction units having the same motion information, which is redundant to just have one PU in a coding unit. Similarly, position B1 is not considered when the current PU is partitioned as 2N×N.



Figure 4‑13. Positions of spatial merge candidates.



Figure 4‑14. Candidate pairs considered for redundancy check of spatial merge candidates.



Figure 4‑15. Positions for the second PU of N×2N and 2N×N partitions.

#### Temporal candidates derivation

In this step, only one candidate is added to the list. Particularly, in the derivation of this temporal merge candidate, a scaled motion vector is derived based on co-located PU belonging to the picture which has the smallest POC difference with current picture within the given reference picture list. The reference picture list to be used for derivation of the co-located PU is explicitly signalled in the slice header. The scaled motion vector for temporal merge candidate is obtained as illustrated by the dotted line in Figure 4‑16, which is scaled from the motion vector of the co-located PU using the POC distances, tb and td, where tb is defined to be the POC difference between the reference picture of the current picture and the current picture and td is defined to be the POC difference between the reference picture of the co-located picture and the co-located picture. The reference picture index of temporal merge candidate is set equal to zero. A practical realization of the scaling process is described in the HEVC specification [2]. For a B-slice, two motion vectors, one is for reference picture list 0 and the other is for reference picture list 1, are obtained and combined to make the bi-predictive merge candidate.



Figure 4‑16. Illustration of motion vector scaling for temporal merge candidate.

In the co-located PU (Y) belonging to the reference frame, the position for the temporal candidate is selected between candidates C0 and C1, as depicted in Figure 4‑17. If PU at position C0 is not available, is intra coded, or is outside of the current CTU, position C1 is used. Otherwise, position C0 is used in the derivation of the temporal merge candidate.



Figure 4‑17. Candidate positions for temporal merge candidate, C0 and C1.

#### Additional candidates insertion

Besides spatio-temporal merge candidates, there are two additional types of merge candidates: combined bi-predictive merge candidate and zero merge candidate. Combined bi-predictive merge candidates are generated by utilizing spatio-temporal merge candidates. Combined bi-predictive merge candidate is used for B-Slice only. The combined bi-predictive candidates are generated by combining the first reference picture list motion parameters of an initial candidate with the second reference picture list motion parameters of another. If these two tuples provide different motion hypotheses, they will form a new bi-predictive candidate. As an example, Figure 4‑18 depicts the case when two candidates in the original list (on the left), which have mvL0 and refIdxL0 or mvL1 and refIdxL1, are used to create a combined bi-predictive merge candidate added to the final list (on the right). There are numerous rules regarding the combinations which are considered to generate these additional merge candidates, defined in [2].

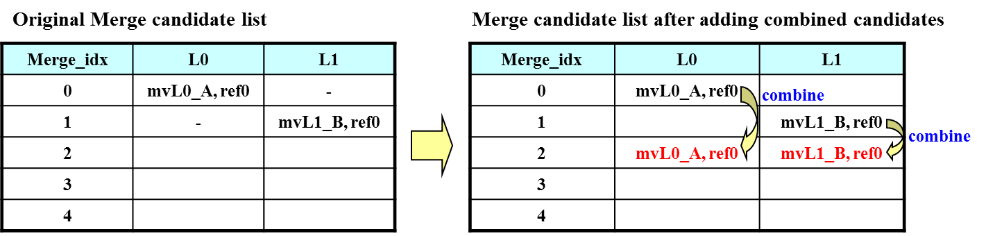


Figure 4‑18. Example of combined bi-predictive merge candidate.

Zero motion candidates are inserted to fill the remaining entries in the merge candidates list and therefore hit the MaxNumMergeCand capacity. These candidates have zero spatial displacement and a reference picture index which starts from zero and increases every time a new zero motion candidate is added to the list. The number of reference frames used by these candidates is one and two for uni and bi-directional prediction, respectively. Finally, no redundancy check is performed on these candidates.

#### Motion estimation regions for parallel processing

To speed up the encoding process, motion estimation can be performed in parallel whereby the motion vectors for all prediction units inside a given region are derived simultaneously. The derivation of merge candidates from spatial neighbourhood may interfere with parallel processing as one prediction unit cannot derive the motion parameters from an adjacent PU until its associated motion estimation is completed. To mitigate the trade-off between coding efficiency and processing latency, HEVC defines the motion estimation region (MER) whose size is signalled in the picture parameter set using the “log2\_parallel\_merge\_level\_minus2” syntax element [2]. When a MER is defined, merge candidates falling in the same region are marked as unavailable and therefore not considered in the list construction.

### Motion vector prediction

Motion vector prediction exploits spatio-temporal correlation of motion vector with neighbouring PUs, which is used for explicit transmission of motion parameters. It constructs a motion vector candidate list by firstly checking availability of left, above temporally neighbouring PU positions, removing redundant candidates and adding zero vector to make the candidate list to be constant length. Then, the encoder can select the best predictor from the candidate list and transmit the corresponding index indicating the chosen candidate. Similarly with merge index signalling, the index of the best motion vector candidate is encoded using truncated unary. The maximum value to be encoded in this case is 2 (see Figure 4‑19). In the following sections, details about derivation process of motion vector prediction candidate are provided.

#### Derivation of motion vector prediction candidates

Figure 4‑19 summarizes derivation process for motion vector prediction candidate.

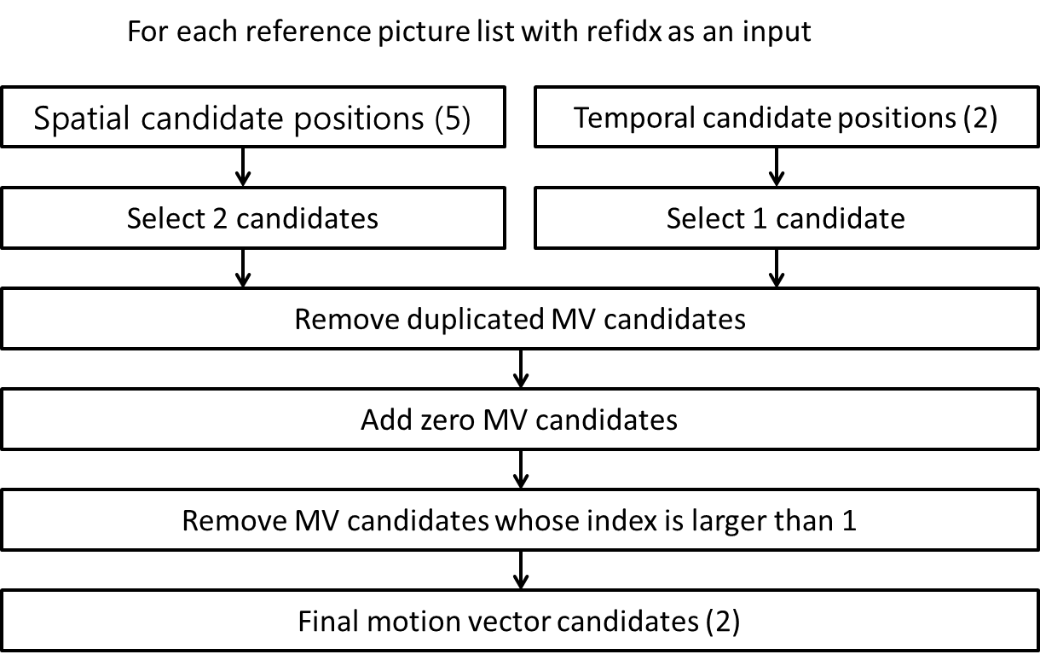


Figure 4‑19. Derivation process for motion vector prediction candidates.

In motion vector prediction, two types of motion vector candidates are considered: spatial motion vector candidate and temporal motion vector candidate. For spatial motion vector candidate derivation, two motion vector candidates are eventually derived based on motion vectors of each PU located in five different positions as depicted in Figure 4‑13.

For temporal motion vector candidate derivation, one motion vector candidate is selected from two candidates, which are derived based on two different co-located positions. After the first list of spatio-temporal candidates is made, duplicated motion vector candidates in the list are removed. If the number of potential candidates is larger than two, motion vector candidates whose reference picture index within the associated reference picture list is larger than 1 are removed from the list. If the number of spatio-temporal motion vector candidates is smaller than two, additional zero motion vector candidates is added to the list.

#### Spatial motion vector candidates

In the derivation of spatial motion vector candidates, a maximum of two candidates are considered among five potential candidates, which are derived from PUs located in positions as depicted in Figure 4‑13, those positions being the same as those of motion merge. The order of derivation for the left side of the current PU is defined as A0, A1,and scaled A0,scaled A1. The order of derivation for the above side of the current PU is defined as B0, B1, B2, scaled B0, scaled B1, scaled B2. For each side there are therefore four cases that can be used as motion vector candidate, with two cases not required to use spatial scaling, and two cases where spatial scaling is used. The four different cases are summarized as follows.

* No spatial scaling
  + (1) Same reference picture list, and same reference picture index (same POC)
  + (2) Different reference picture list, but same reference picture (same POC)
* Spatial scaling
  + (3) Same reference picture list, but different reference picture (different POC)
  + (4) Different reference picture list, and different reference picture (different POC)

The no-spatial-scaling cases are checked first followed by the spatial scaling. Spatial scaling is considered when the POC is different between the reference picture of the neighbouring PU and that of the current PU regardless of reference picture list. If all PUs of left candidates are not available or are intra coded, scaling for the above motion vector is allowed to help parallel derivation of left and above MV candidates. Otherwise, spatial scaling is not allowed for the above motion vector.



Figure 4‑20. Illustration of motion vector scaling for spatial motion vector candidate.

In a spatial scaling process, the motion vector of the neighbouring PU is scaled in a similar manner as for temporal scaling, as depicted as Figure 4‑20. The main difference is that the reference picture list and index of current PU is given as input; the actual scaling process is the same as that of temporal scaling.

#### Temporal motion vector candidates

Apart for the reference picture index derivation, all processes for the derivation of temporal merge candidates are the same as for the derivation of spatial motion vector candidates (see Figure 4‑17). The reference picture index is signalled to the decoder.

### Interpolation filter

For the luma interpolation filtering, an 8-tap separable DCT-based interpolation filter is used for 2/4 precision samples and a 7-tap separable DCT-based interpolation filter is used for 1/4 precisions samples, as shown in Table 4‑4.

Table 4‑4. 8-tap DCT-IF coefficients for 1/4th luma interpolation.

|  |  |
| --- | --- |
| **Position** | **Filter coefficients** |
| 1/4 | { -1, 4, -10, 58, 17, -5, 1 } |
| 2/4 | { -1, 4, -11, 40, 40, -11, 4, -1 } |
| 3/4 | { 1, -5, 17, 58, -10, 4, -1 } |

Similarly, a 4-tap separable DCT-based interpolation filter is used for the chroma interpolation filter, as shown in Table 4‑5.

Table 4‑5. 4-tap DCT-IF coefficients for 1/8th chroma interpolation.

|  |  |
| --- | --- |
| **Position** | **Filter coefficients** |
| 1/8 | { -2, 58, 10, -2 } |
| 2/8 | { -4, 54, 16, -2 } |
| 3/8 | { -6, 46, 28, -4 } |
| 4/8 | { -4, 36, 36, -4 } |
| 5/8 | { -4, 28, 46, -6 } |
| 6/8 | { -2, 16, 54, -4 } |
| 7/8 | { -2, 10, 58, -2 } |

For the vertical interpolation for 4:2:2 and the horizontal and vertical interpolation for 4:4:4 chroma channels, the odd positions in Table 4‑5 are not used, resulting in 1/4th chroma interpolation.

For the bi-directional prediction, the bit-depth of the output of the interpolation filter is maintained to 14-bit accuracy, regardless of the source bit-depth, before the averaging of the two prediction signals. The actual averaging process is done implicitly with the bit-depth reduction process as:

predSamples[ x, y ] = ( predSamplesL0[ x, y ] + predSamplesL1[ x, y ] + offset ) >> shift

where

shift = ( 15 − BitDepth ) and offset = 1 << ( shift − 1 )

### Weighted Prediction

A weighted prediction (WP) tool is provided by HEVC. WP corresponds to the equivalent tool present in AVC and is intended to improve the performance of inter prediction when the source material is subject to illumination variations, e.g. when using fading or cross-fading. It should be noted that WP is not enabled in the HM common test conditions [3].

The principle of WP is to replace the inter prediction signal *P* by a linear weighted prediction signal *P’* (with weight *w* andoffset *o*):

Uni-prediction: P’ = w × P + o

Bi-prediction: P’ = (w0 × P0 + o0 + w1 × P1 + o1) / 2

The applicable weights and offsets are selected by the encoder and are conveyed within the bitstream. L0 and L1 suffixes define List0 and List1 of the reference pictures list, respectively. Bit depth is maintained to 14 bit accuracy (in HEVC Version 1) before averaging the prediction signals, as for interpolation filters.

In the case of bi-prediction with at least one reference picture available in each list L0 and L1, the following formula applies to the explicit signalling of weighted prediction parameters relating to the luma channel:

predSamples[ x ][ y ] =

Clip3( 0, ( 1 << bitDepth ) − 1, ( predSamplesL0 [ x ][ y ] \* w0 + predSamplesL1[ x ][ y ] \* w1 + ( (o0 + o1 + 1) << log2WD) ) >> (log2WD + 1) )

where

log2WD = luma\_log2\_weight\_denom + 14 - bitDepth

w0 = LumaWeightL0[ refIdxL0 ], w1 = LumaWeightL1[ refIdxL1 ]

o0 = luma\_offset\_l0[ refIdxL0 ] \* highPrecisionScaleFactor

o1 = luma\_offset\_l1[ refIdxL1 ] \* highPrecisionScaleFactor

highPrecisionScaleFactor = (1 << ( bitDepth − 8 ) ) (except seeSection 4.4.4.1)

A corresponding formula applies to the chroma channel and to the case of uni-prediction.

#### High precision offsets

A highPrecisionScaleFactor of ( 1 << ( bitDepth − 8 ) ) is applied to the weighted prediction offsets o0 and o1 when *high\_precision\_offsets\_enabled\_flag* is equal to zero. At higher bit depths, this factor increases in magnitude, reducing the performance of weighted prediction.

When the *high\_precision\_offsets\_enabled\_flag* is equal to one (permitted in RExt profiles), the offsets (o0 and o1) have the same precision as the input (i.e. the factor above is removed) in order to provide enough precision for the weighted prediction process.

The *high\_precision\_offsets\_enabled\_flag* is configured by HighPrecisionPredictionWeighting in the encoder.

## Transform and quantization (scaling)

In HEVC, each TB (which are 4×4, 8×8, 16×16 or 32×32 samples) is transformed, transform skipped or coded losslessly via a *trans-quant-bypass* mode.

The scaling and transformation processes at the decoder side for a transformed block are shown in Figure 4‑21.



Figure 4‑21. Lossless scaling and transformation process (quantities illustrated for HEVC version 1 profiles).

When transform skip (TS) is used, a bit-shift is applied instead of a transform. TS may be applied to 4×4 TBs for Main/Main10 profiles and TBs of any size for range extensions profiles. In this case, the scaling and transformation processes at the decoder side are as shown in Figure 4‑22.

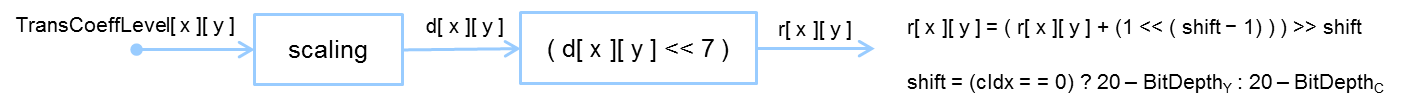


Figure 4‑22. Transform skip scaling and transformation process (quantities illustrated for HEVC version 1 profiles).

For the range extensions profiles, transform skip is supported on all TU sizes. The maximum sized TU for which transform skip is available is signaled by *log2\_max\_transform\_skip\_block\_size\_minus2* (as configured by TransformSkipLog2MaxSize)

If lossless mode is used, scaling and transformation process at the decoder side are as follows.

r[ x ][ y ] = TransCoeffLevel[ x ][ y ]

### Inverse transforms

The inverse transform is implemented as a vertical 1D transformation step operating on each column of residual coefficients (i.e. d[ x ][ y ]), following by a clipping step operating on the output of the vertical 1D transformation step (i.e. e[ x ][ y ]) and finally a horizontal 1D transformation step operating on each row of the output of the clipping step (i.e. g[ x ][ y ]). This process is illustrated in Figure 4‑23 for HEVC version 1 profiles. The clipping of intermediate sample values g[ x ][ y ] ensures that these values can be represented with 16 bits.

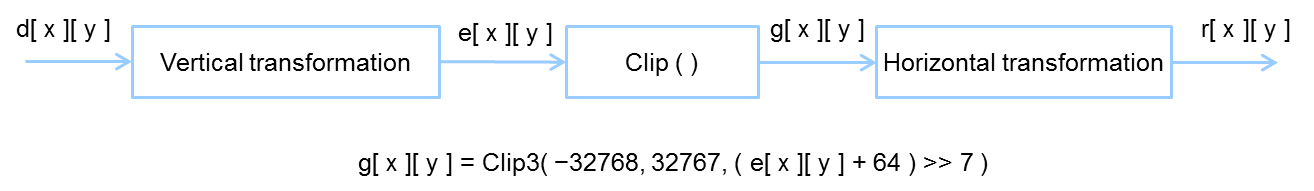


Figure 4‑23. Inverse transformation process (quantities illustrated for HEVC version 1 profiles).

### 1D inverse transform matrices

The transform matrices are an approximation of mathematical DCT matrices with 8-bit integers (including sign), utilizing 6 bits of fractional accuracy. The matrices have been optimized for maximizing orthogonality. Smaller size transform matrices are embedded in larger size transform matrices, enabling reuse of a subsets of the 32×32 matrix when performing 4×4, 8×8, 16×16, and 32×32 transforms.

A 4×4 DST is also provided and is used only for residuals of all 4×4 intra-predicted luma TBs.

In the HM implementation, the transform is applied using a partial butterfly structure for low computational complexity.

### Scaling and quantization

Quantization is performed using integer arithmetic in HEVC, with quantizer step size doubling for every increase of QP by 6. The QP remainder (QP%6) specifies a fractional scaling of the quantizer step size normalized at 16384 (corresponding to 2QUANT\_SHIFT, with QUANT\_SHIFT equal to 14) and using a table f[x]. When scaling lists are used, an additional fractional scaling is performed that depends on the position of the transform coefficient in the TB, with either a default scaling list used, or a scaling list supplied via a file. The additional fractional scaling is normalized at 16. Section 4.5.4 describes scaling list derivation for different sized TBs and prediction modes.

When scaling lists are not used, the quantized transform coefficients qij (i, j=0..nS-1) are derived from the transform coefficients dij (i, j=0..nS-1) and the scaling list sij (i, j = 0..nS-1) as

qij = ( dij \* f[QP%6] + offset ) >> ( QUANT\_SHIFT + MAX\_TR\_DYNAMIC\_RANGE + QP/6 – log2(nS) − BitDepth), with i,j = 0,...,nS-1

where

f[x] = {26214,23302,20560,18396,16384,14564}, x=0,…,5

228+QP/6−nS-BitDepth  < offset < 229+QP/6−nS-BitDepth QUANT\_SHIFT = 14

MAX\_TR\_DYNAMIC\_RANGE = 15 when *extended\_precision\_processing\_flag* is equal to 0, otherwise see Section 4.5.8.

When scaling lists are used, the quantized transform coefficients qij (i, j=0..nS-1) are derived from the transform coefficients dij (i, j=0..nS-1) and the scaling list sij (i, j = 0..nS-1) as

qij = ( dij \* ( f[QP%6] << 4 / sij ) + offset ) >> ( QUANT\_SHIFT + MAX\_TR\_DYNAMIC\_RANGE + QP/6 – log2(nS) − BitDepth), with i,j = 0,...,nS-1

The value offset is set at 171 / 512 for I slices and 85 / 512 for P or B slices.

When scaling lists are not applied, sij = 16 for all i and j.

A rate-distortion optimized quantization (RDOQ) process is also available and may be used instead, as described in Section 6.6.2.

### Scaling lists

Scaling lists (c.f. quantization matrices) can be applied during the (inverse) quantization process. The scaling values are signalled in the PPS, and each possible TB size, colour component and prediction type (intra/inter) can have its own scaling list, except for 32×32 chroma blocks, which is used only for the 4:4:4 RExt chroma format. For 16×16 and 32×32 scaling lists, the scaling lists are specified with an 8x8 grid of values which is value-repeated to the required size, along with a value used for the entry corresponding to the DC frequency location.

Application of scaling lists is via the ScalingList encoder configuration option. Either scaling lists are not applied, default scaling lists are used, or scaling lists supplied via a ScalingListFile argument are used (refer to the software manual [1] for detail on the file format).

The scaling list for 32×32 chroma blocks is derived from the 16×16 chroma scaling list. This derivation is shown in Figure 4‑24.



Figure 4‑24. Scaling lists.

### Scaling lists for transform skipped TUs

Scaling lists are not used for any transform-skipped TUs, other than 4×4.

### Transform selection for the 4:2:2 chroma format

When the 4:2:2 chroma format is in use, a TU has a rectangular chroma block. In this case, the rectangular chroma blocks are divided into two square TBs per channel and existing square transforms are used for the TBs. A separate coded block flag is signalled for each TB. Intra prediction reconstruction occurs separately for the two square blocks within a rectangular chroma block, enabling the lower block to be predicted from the reconstructed upper block.



Figure 4‑25. Square transform arrangement for the 4:2:2 chroma format..

### Chroma QP initialization offset table

When the chroma format is set to 4:2:2 or 4:4:4, the chroma QPc is initialized according to the luma qPi using the formula Min( qPi, 51 ). In particular, the mapping relationship of Table 8-10 of [2] is not used.

### Extended precision processing

An *extended\_precision\_processing\_flag* (as configured by ExtendedPrecision) is provided to allow increased internal precision, particularly for use at higher bit depths. When this flag is set to one, the internal width of the transform and the entropy coder (g\_maxTrDynamicRange[channel]in the software model) are increased according to the selected bit depth. Figure 4‑26 shows the bit depths in the HEVC encoding path.



Figure 4‑26. Diagram showing magnitude bit depths in HEVC encoding path.

Table 4‑6 shows the relationship between the internal precisions and the bit depth.

Table 4‑6. g\_maxTrDynamicRange[channel].

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***extended\_precision\_processing\_flag*** | **Bit depth[channel]** | | | | | | | | |
| **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **1** | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | |
| **0** | 15 | | | | | | | | |

### CU-adaptive chroma QP offset

A chroma QP offset adjustment may be signalled at the CU level, for CUs down to a particular depth (signalled via *diff\_cu\_chroma\_qp\_adjustment\_depth***)**. The same QP offset may also be applied to subsequent sibling or child CUs within the CTU tree if the maximum depth is reached. This is similar in principle to the operation of delta QP signalling in the current HEVC specification. To provide additional flexibility in terms of number of offsets, each CU that invokes the mode may signal an index into an offset table. This offset table contains up to five pairs of chroma QP offset values (*cb\_qp\_adjustment*, *cr\_qp\_adjustment*). The offset table is signalled in the PPS, limiting the rate cost of providing the chroma QP offsets. If the table contains one offset then no index is signalled. The chroma QP adjustment values are restricted such that the total deviation from the luma QP is limited to ±12.

For a CU where the offset is applied, each chroma QP adjustment value in a pair is applied to the corresponding chroma component. The feature is globally enabled through use of a picture parameter set flag *chroma\_qp\_adjustment\_enabled\_flag*, and locally through a slice header flag *slice\_chroma\_qp\_adjustment\_enabled\_flag*.

## Residual prediction in case of transquant bypass and transform skip

When lossless coding is used (i.e. *cu\_transquant\_bypass\_flag* is equal to one) and *implicit\_rdpcm\_enabled\_flag* is equal to one, the residues obtained from intra prediction are further predicted using DPCM. Residual DPCM is only applied when the intra prediction direction is either horizontal or vertical. For residues obtained from inter prediction, RDPCM is applied if *explicit\_rdpcm\_enabled\_flag* is equal to one. In this case the encoder selects whether to apply RDPCM on the residues and if it is applied, whether to perform RDPCM along the horizontal or vertical direction. This decision is based on the sum of absolute difference (SAD) computed over the residues. The decision which minimises the SAD is selected as the best and signalled to the decoder using two binary flags: one to signal whether RDPCM is applied and one to signal the direction in which RDPCM is applied.

When lossy coding is used (i.e. *cu\_transquant\_bypass\_flag* is equal to zero), RDPCM may be applied at TU level and only for those TUs which are transform skipped (e.g. 4×4 transform units). For intra TUs, RDPCM is always applied when *implicit\_rdpcm\_enabled flag* is equal to one for intra coded transform-skip TUs with horizontal or vertical intra prediction mode, so there is no additional signalling. For inter TUs, when *explicit\_rdpcm\_enabled\_flag* is equal to one, the same mechanism used for lossless coding is followed. For intra and inter coding, RDPCM is computed using the reconstructed residues (i.e. after inverse quantization) in order to avoid any drift between the encoder and the decoder. In particular the process can be formalized as follows. Let  and  be the residues obtained by the RDPCM application. Let  be the residue obtained after inverse quantization and *r*(*i*, *j*) the original residue (i.e. obtained by either intra or inter prediction). Over an *N*×*N* block, the residues  and  are therefore defined as follows:



,

where *Q* denotes the forward quantization operation.

The *implicit\_rdpcm\_enabled\_flag* and *explicit\_rdpcm\_enabled\_flag* are controlled using the configuration options ImplicitResidualDPCM and ExplicitResidualDPCM respectively.

## Entropy coding

HEVC uses context adaptive binary arithmetic coding (CABAC) and variable length codes. Each syntax element is binarised using a combination of context coded bins and bypass bins. Context coded bins have an associated context, indicating the probable symbol value. The cost of coding a context coded bin depends on the probability and whether the symbol to be coded is equal to the likely symbol value. Bypass coded bins have no associated context and have an equal cost for coding each of a 0 or 1 symbol. Context coding allows adaptation to the probability distribution of symbol values for a given bin. Further adaptation is provided by context selection. A given context coded bin may use one of several contexts, the context being selected based on information previously included in the bitstream.

Variable length codes are used above the slice layer and in the slice header.

### CABAC alignment

When *cabac\_bypass\_alignment\_enabled\_flag* is equal to one (as configured by AlignCABACBeforeBypass), the CABAC engine is bit-aligned (i.e. range is set to 256) prior to the coding of sign bits if there are any coefficients that require *coeff\_abs\_level\_remaining* syntax elements. Note that once the CABAC engine is bit-aligned, it will remain bit-aligned until a context coded bin is encountered. Alignment prior to decoding equi-probable CABAC bins allows those bins to be read directly from the bit-stream.

Bit alignment results in bypass bins being aligned in the bitstream, i.e. a decoder can read a given bypass bin directly from the bitstream.

## Coefficient Coding

### Transform skip residual rotation

When *transform\_skip\_rotation\_enabled\_flag* is equal to one (as configured by ResidualRotation), the residual of a 4x4 transform-skipped block or transquant-bypass block is rotated by 180 degrees. Due to the symmetry of the scans used in HEVC, this is equivalent to reversing the scan orders.

### Significance map context modelling

When *transform\_skip\_context\_enabled\_flag* is equal to one (as configured by SingleSignificanceMapContext), a separate single context is used for the *sig\_coeff\_flag* for TUs that are transform-skipped or transquant-bypassed.

### Rice parameter adaptation

When *persistent\_rice\_adaptation\_enabled\_flag* is equal to one (as configured by GolombRiceParameterAdaptation*)*, an alternative mechanism for initializing the Rice parameter used for coding *coeff\_abs\_level\_remaining* is available.

In this scheme, the 4×4 sub-blocks are divided into different categories (“sbType”). For each sub-block, the initial Rice parameter is derived based on previously coded sub-blocks in the same category. The categorization is based on whether the block is a transform-skip block (“isTSFlag”) or in trans-quant bypass (isTQBFlag) and whether it is the luma component

sbType = isLuma \* 2 + (isTSFlag | | isTQBFlag)

Stats *statCoeff* are maintained for each sub-block type (sbType) depending on the absolute coefficient value (*uiLevel*):

if (uiLevel >= ( 3 << ( statCoeff[ sbType ] / 4 ) ) ) statCoeff[ sbType ] ++;

else if ( ( 2 \* uiLevel ) < ( 1 << ( statCoeff[ sbType ] / 4 ) ) ) statCoeff[ sbType ] --;

This variable is updated at most once per 4×4 sub-block using the value of the first coded *coeff\_abs\_level\_remaining* of the sub-block. The entries of *statCoeff* are reset to 0 at the beginning of the slice (like the CABAC context variables).

The value of *statCoeff* is used to initialize the Rice parameter at the beginning of each 4×4 sub-block as:

cRiceParam = Min( maxRicePara, statCoeff/4 ).

When this mechanism for initialization is enabled, the maximum Rice parameter value is unrestricted (i.e. limited only by the maximum transform dynamic range).

### Maximum coeff\_abs\_level\_remaining codeword length restriction

When *extended\_precision\_processing\_flag* is enabled, the maximum codeword length of the *coeff\_abs\_level\_remaining* syntax element is limited to 32-bits. The maximum codeword length is achieved with the Rice parameter is equal to zero. The maximum codeword length is dependent on the dynamic range of transform and quantizer stages, referred to as ‘MAX\_TR\_DYNAMIC\_RANGE’ in the HEVC Test Model software. When *extended\_precision\_processing\_flag* is enabled, MAX\_TR\_DYNAMIC\_RANGE is set equal to the bit-depth plus six bits.

Binarization of *coeff\_abs\_level\_remaining* is modified such that the maximum prefix length is given by:

*maximumPrefixLength* = 32 − (3 + *MAX\_TR\_DYNAMIC\_RANGE*)

When this prefix length is reached, the corresponding suffix length is then given by:

*suffixLength* = *MAX\_TR\_DYNAMIC\_RANGE* − *rParam*

This results in a maximum codeword length for *coeff\_abs\_level\_remaining* of 32-bits.

## Cross-component prediction

An adaptive cross-component residual prediction scheme (i.e., between colour channels) is provided, where a prediction is performed between the luma residual signal and the chroma residual signals. The chroma residual signal is predicted from the luma residual signal at the encoder side as:

 (1)

and it is compensated at the decoder side as:

 (2)

where  denotes the chroma residual sample at a position ,  denotes the reconstructed residual sample of the luma component,  denotes the predicted signal using inter-colour prediction, denotes the reconstructed signal after coding and decoding , and  denotes the reconstructed chroma residual.

The variable α is chosen from . This set of values allows good utilization of the correlation between luma and chroma residual signals, including when they are negatively correlated. The absolute value of α, abs(α) is mapped to M(α) according to Table 4‑7before binarization. M(α) is binarized using truncated unary code and coded using CABAC, with a separate context for each bin in the TU code. If α is not zero, another bin is used to code the sign of α. At the decoder side, after CABAC decoding, M(α) is inversely mapped back to abs(α).

Table 4‑7. α Mapping Table.

|  |  |
| --- | --- |
| abs(α) | M(α) |
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 4 | 3 |
| 8 | 4 |

This prediction is performed both for intra- and inter-coded blocks. However, in case of intra-coded blocks, only those with DM chroma mode are allowed to use this prediction.

For each TU, if the coded block flag of the luma component is zero, α is not signaled and no prediction is performed. Otherwise, α is signalled separately for each chroma component.

When encoding RGB source material, the G component should be encoded as the luma component.

The use of cross-component prediction is signalled using *cross\_component\_prediction\_enabled flag* and is configured using CrossComponentPrediction.

When reconstruction based cross-component prediction estimate is enabled, use the decoded residual rather than the pre-transform encoder-side residual for determining the alpha value. This is configured using ReconBasedCrossCPredictionEstimate.

If the luma bit-depth differs from the chroma bit-depth, for cross component prediction the luma residual is scaled to align with the chroma bit-depth by either a right-shift or a left-shift operation.

## Loop Filtering

### Overview of Loop filtering

HEVC includes two processing stages in the in-loop filter: a deblocking filter and then a sample adaptive offset (SAO) filter. The deblocking filter aims to reduce the visibility of blocking artefacts and is applied only to samples located at block boundaries. The SAO filter aims to improve the accuracy of the reconstruction of the original signal amplitudes and is applied adaptively to all samples, by conditionally adding an offset value to each sample based on values in look-up tables defined by the encoder.

### Deblocking filter

A deblocking filter process is performed for each CU in the same order as the decoding process. First vertical edges are filtered (horizontal filtering) then horizontal edges are filtered (vertical filtering). Filtering is applied to 8×8 block boundaries which are determined to be filtered, both for luma and chroma components. 4×4 block boundaries are not processed in order to reduce the complexity.

Figure 4‑27 illustrates the overall flow of deblocking filter processes. A boundary can have three filtering status values: no filtering, weak filtering and strong filtering. Each filtering decision is based on boundary strength, Bs, and threshold values, β and tC.



Figure 4‑27. Overall processing flow of deblocking filter process.

#### Boundary decision

Two kinds of boundaries are involved in the deblocking filter process: TU boundaries and PU boundaries. CU boundaries are also considered, since CU boundaries are necessarily also TU and PU boundaries. When PU shape is 2NxN (N > 4) and RQT depth is equal to 1, TU boundaries at 8x8 block grid and PU boundaries between each PU inside the CU are also involved in the filtering.

#### Boundary strength calculation

The boundary strength (Bs) reflects how strong a filtering process may be needed for the boundary. A value of 2 for Bs indicates strong filtering, 1 indicates weak filtering and 0 indicates no deblocking filtering.,

Let P and Q be defined as blocks which are involved in the filtering, where P represents the block located to the left (vertical edge case) or above (horizontal edge case) the boundary and Q represents the block located to the right (vertical edge case) or above (horizontal edge case) the boundary. Figure 4‑28 illustrates how the Bs value is calculated based on the intra coding mode, the existence of non-zero transform coefficients, reference picture, number of motion vectors and motion vector difference.



Figure 4‑28. Flow diagram for Bs calculation.

Bs is calculated on a 4×4 block basis, but it is re-mapped to an 8×8 grid. The maximum of the two values of Bs which correspond to 8 pixels consisting of a line in the 4×4 grid is selected as the Bs for boundaries in the 8×8 grid.

At the CTU boundary, information on every second block (on a 4×4 grid) to the left or above is re-used as depicted in Figure 4‑29, in order to reduce line buffer memory requirement.

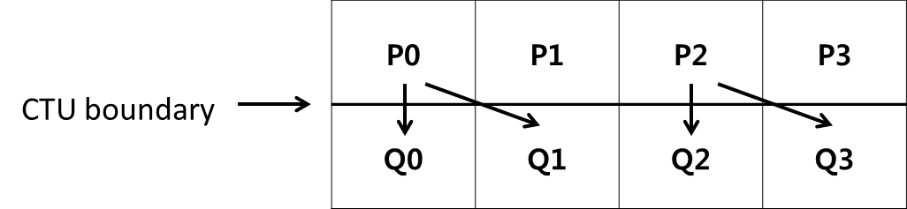


Figure 4‑29. Referred information for Bs calculation at CTU boundary.

#### Threshold variables

Threshold values β′ and tC′ are involved in the filter on/off decision, strong and weak filter selection and weak filtering process. These are derived from the value of the luma quantization parameter Q as shown in Table 4‑8.

Table 4‑8. Derivation of threshold variables from input Q.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| **β**′ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 7 | 8 |
| **tC**′ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Q** | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| **β**′ | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 |
| **tC**′ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |
| **Q** | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |  |  |  |
| **β**′ | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 | - | - |  |  |  |
| **tC**′ | 5 | 5 | 6 | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 14 | 16 | 18 | 20 | 22 | 24 |  |  |  |

The variable β is derived from β′ as follows:

β = β′ \* ( 1  <<  ( BitDepthY − 8 ) )

The variable tC is derived from tC′ as follows:

tC = tC′ \* ( 1  <<  ( BitDepthY − 8 ) )

#### Filter on/off decision for 4 lines

The filter on/off decision is made using 4 lines grouped as a unit, to reduce computational complexity. Figure 4‑30 illustrates the pixels involving in the decision. The 6 pixels in the two red boxes in the first 4 lines are used to determine whether the filter is on or off for those 4 lines. The 6 pixels in the two red boxes in the second group of 4 lines are used to determine whether the filter is on or off for the second group of 4 lines.



Figure 4‑30. Pixels involved in filter on/off decision and strong/weak filter selection.

The following variables are defined:

dp0 = | p2,0 − 2\*p1,0 + p0,0 |

dp3 = | p2,3 − 2\*p1,3 + p0,3 |

dq0 = | q2,0 − 2\*q1,0 + q0,0 |

dq3 = | q2,3 − 2\*q1,3 + q0,3 |

If dp0+dq0+dp3+dq3 < β, filtering for the first four lines is turned on and the strong/weak filter selection process is applied. If this condition is not met, no filtering is done for the first 4 lines.

Additionally, if the condition is met, the variables dE, dEp1 and dEp2 are set as follows:

dE is set equal to 1

If dp0 + dp3 < (β + ( β >> 1 )) >> 3, the variable dEp1 is set equal to 1

If dq0 + dq3 < (β + ( β >> 1 )) >> 3, the variable dEq1 is set equal to 1

A filter on/off decision is made in a similar manner as described above for the second group of 4 lines.

#### Strong/weak filter selection for 4 lines

If filtering is turned on, a decision is made between strong and weak filtering. The pixels involved are the same as those used for the filter on/off decision, as depicted in Figure 4‑20. If the following two sets of conditions are met, a strong filter is used for filtering of the first 4 lines. Otherwise, a weak filter is used.

1) 2\*(dp0+dq0) < ( β >> 2 ), | p30 − p00 | + | q00 − q30 | < ( β >> 3 ) and | p00 − q00 | < ( 5\* tC  + 1 ) >> 1

2) 2\*(dp3+dq3) < ( β >> 2 ), | p33 − p03 | + | q03 − q33 | < ( β >> 3 ) and | p03 − q03 | < ( 5\* tC  + 1 ) >> 1

The decision on whether to select strong or weak filtering for the second group of 4 lines is made in a similar manner.

#### Strong filtering

For strong filtering, the filtered pixel values are obtained by the following equations. Note that three pixels are modified using four pixels as an input for each P and Q block, respectively.

p0’ = ( p2 + 2\*p1 + 2\*p0 + 2\*q0 + q1 + 4 ) >> 3

q0’ = ( p1 + 2\*p0 + 2\*q0 + 2\*q1 + q2 + 4 ) >> 3

p1’ = ( p2 + p1 + p0 + q0 + 2 ) >> 2

q1’ = ( p0 + q0 + q1 + q2 + 2 ) >> 2

p2’ = ( 2\*p3 + 3\*p2 + p1 + p0 + q0 + 4 ) >> 3

q2’ = ( p0 + q0 + q1 + 3\*q2 + 2\*q3 + 4 ) >> 3

#### Weak filtering

Δ is defined as follows.

Δ = ( 9 \* ( q0 −  p0 ) − 3 \* ( q1 − p1 ) + 8 ) >> 4

When abs(Δ) is less than tC \*10,

Δ = Clip3( - tC , tC , Δ )

p0’ = Clip1Y( p0 + Δ )

q0’ = Clip1Y( q0 - Δ )

If dEp1 is equal to 1,

Δp = Clip3( -( tC  >> 1), tC  >> 1, ( ( ( p2 + p0 + 1 ) >> 1 ) − p1 + Δ ) >>1 )

p1’ = Clip1Y( p1 + Δp )

If dEq1 is equal to 1,

Δq = Clip3( -( tC  >> 1), tC  >> 1, ( ( ( q2 + q0 + 1 ) >> 1 ) − q1 − Δ ) >>1 )

q1’ = Clip1Y( q1 + Δq )

Note that a maximum of two pixels are modified using three pixels as an input for each P and Q block, respectively.

#### Chroma filtering

The boundary strength Bs for chroma filtering is inherited from luma. If Bs > 1, chroma filtering is performed. No filter selection process is performed for chroma, since only one filter can be applied. The filtered sample values p0’ and q0’ are derived as follows.

Δ = Clip3( -tC, tC, ( ( ( ( q0 − p0 ) << 2 ) + p1 − q1 + 4 ) >> 3 ) )

p0’ = Clip1C( p0 + Δ )

q0’ = Clip1C( q0 - Δ )

When the 4:2:2 chroma format is in use, each chroma block has a rectangular shape and is coded using up to two square transforms. This process introduces additional boundaries between the transform blocks in chroma. These boundaries are not deblocked (blue dotted lines in Figure 4‑31).



Figure 4‑31. Deblocking behaviour in the 4:2:2 chroma format.

### Sample adaptive offset filter

Sample adaptive offset (SAO) is applied to the reconstructed signal after the deblocking filter by using offsets specified for each CTB by the encoder. The HM encoder first makes the decision on whether or not the SAO process is to be applied for current slice. If SAO is applied for the slice, each CTB is classified as one of five SAO types as shown in Table 4‑9. The concept of SAO is to classify pixels into categories and reduces the distortion by adding an offset to pixels of each category. SAO operation includes edge offset (EO) which uses edge properties for pixel classification in SAO type 1-4 and band offset (BO) which uses pixel intensity for pixel classification in SAO type 5. Each applicable CTB has SAO parameters including sao\_merge\_left\_flag, sao\_merge\_up\_flag, SAO type and four offsets. If sao\_merge\_left\_flag is equal to 1, the current CTB will reuse the SAO type and offsets of the CTB to the left. If sao\_merge\_up\_flag is equal to 1, the current CTB will reuse SAO type and offsets of the CTB above.

Table 4‑9. Specification of SAO type.

|  |  |  |
| --- | --- | --- |
| **SAO type** | **sample adaptive offset type to be used** | **Number of categories** |
| 0 | None | 0 |
| 1 | 1-D 0-degree pattern edge offset | 4 |
| 2 | 1-D 90-degree pattern edge offset | 4 |
| 3 | 1-D 135-degree pattern edge offset | 4 |
| 4 | 1-D 45-degree pattern edge offset | 4 |
| 5 | band offset | 4 |

#### Operation of each SAO type

Edge offset uses four 1-D 3-pixel patterns for classification of the current pixel p by consideration of edge directional information, as shown in Figure 4‑32. From left to right these are: 0-degree, 90-degree, 135-degree and 45-degree.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **p** |  |  |  | **p** |  |  |  | **p** |  |  |  | **p** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure 4‑32. Four 1-D 3-pixel patterns for the pixel classification in EO.

Each CTB is classified into one of five categories according to Table 4‑10.

Table 4‑10. Pixel classification rule for EO.

|  |  |  |
| --- | --- | --- |
| **Category** | **Condition** | **Meaning** |
| 0 | None of the below | Largely monotonic |
| 1 | p < 2 neighbours | Local minimum |
| 2 | p < 1 neighbour && p == 1 neighbour | Edge |
| 3 | p > 1 neighbour && p == 1 neighbour | Edge |
| 4 | p > 2 neighbours | Local maximum |

Band offset (BO) classifies all pixels in one CTB region into 32 uniform bands by using the five most significant bits of the pixel value as the band index. In other words, the pixel intensity range is divided into 32 equal segments from zero to the maximum intensity value (e.g. 255 for 8-bit pixels). Four adjacent bands are grouped together and each group is indicated by its most left-hand position as shown in Figure 4‑33. The encoder searches all position to get the group with the maximum distortion reduction by compensating offset of each band.



Figure 4‑33. Four bands are grouped together and represented by its starting band position.

##### Format range extensions options for SAO

Shift values for luma and chroma for the offset value are included in the PPS. There is no change to classification and the shift values are in the range of 0 to Max(BitDepth − 10, 0). The shift values for luma and chroma are configured in the encoder using SaoLumaOffsetBitShift and SaoChromaOffsetBitShift, respectively.

## Wavefront parallel processing

Wavefront parallel processing (WPP) produces a bitstream that can be processed using one or more cores running in parallel. When WPP is used, a slice is divided into rows of CTUs (per tile). The first row is processed in an ordinary way, the second row can begin to be processed after only two CTUs have been processed in the first row, the third row can begin to be processed after only two CTUs have been processed in the second row, and so on. The context models of the entropy coder in each row are inferred from those in the preceding row with a two-CTU processing lag. WPP provides a form of processing parallelism within a slice, without the loss of compression performance that might be expected by using tiles within a slice.

The following operations are performed by the HM encoder.

- When starting the encoding of the first CTU in a CTU row, the following process is applied:

* if the last CU of the second CTU of the row above is available, the CABAC probabilities are set to the values stored in the buffer
* if not, the CABAC probabilities are reset to the default values

- When the encoding of the second CTU in a CTU row is finished, the CABAC probabilities are stored in a buffer

- If the encoding of the last CTU in a CTU row is finished and the end of a slice has not been reached, CABAC is flushed and a byte alignment is performed.

Entry point offsets are written in the slice header. Each CTU row in the slice has an entry point offset, in byte units, that indicates where the corresponding data starts in the slice data. When WPP is used, a slice that does not start at the beginning of a CTU row does not finish after the last CTU in the same row. When a slice starts at the beginning of a CTB row, there is no constraint on where it finishes.

In all but the high throughput RExt profiles, WPP and tiles cannot be used simultaneously.

WPP is enabled in the HM encoder by setting WaveFrontSynchro to 1.

# Profiles, Levels and Tiers

The HEVC test model software uses particular tools in accordance with the specified profile. To enable the use of tools available in the format range extensions profiles, the Profile encoder option is set to the value main-RExt or high-RExt. The format range extensions profiles are listed in Annex A.3.5 of the HEVC specification [2].

For the main or main10 profiles, set the Profile encoder option to the value main or main10, respectively.Further control of which profile of the format range extensions profiles is to be used when HM generates a bitstream is provided by the encoder configuration options shown in Table 5‑1.

Table 5‑1. Bitstream indications for format range extensions profiles.

|  |  |
| --- | --- |
| **Configuration option** | **Description** |
| IntraConstraintFlag | When the encoder is configured with IntraConstraintFlag, the *general\_intra\_constraint\_flag* is set accordingly.  If this flag is set to 1, only intra slices may be used. |
| LowerBitRateConstraintFlag | When the encoder is configured with LowerBitRateConstraintFlag, the *general\_lower\_bit\_rate\_constraint\_flag* is set accordingly.  Note that LowerBitRateConstraintFlag cannot be false if IntraConstraintFlag is false. |
| MaxBitDepthConstraint | Specifies the maximum allowed luma and chroma internal bit-depths.  If no value (or 0) is specified, the MaxBitDepthConstraint is assumed to be the internal bit depth. The value of MaxBitDepthConstraint controls the setting of the following syntax elements:   * *general\_max\_12bit\_constraint\_flag* * *general\_max\_10bit\_constraint\_flag* * *general\_max\_8bit\_constraint\_flag* |
| MaxChromaFormatConstraint | Specifies the maximum allowed chroma format.  If no value (or 0) is specified, the MaxChromaFormatConstraint is assumed to be that of the internal chroma format. The value of MaxChromaFormatConstraint controls the setting of the following syntax elements:   * *general\_max\_422chroma\_constraint\_flag* * *general\_max\_420chroma\_constraint\_flag* * *general\_max\_monochrome\_constraint\_flag* |
| OnePictureOnlyConstraintFlag | Value of the *general\_one\_picture\_only\_constraint\_flag* to use for RExt profiles (not used if an explicit RExt sub-profile is specified). |

# Description of the HM encoder and encoding methods

This section describes operation of HM to select parameters such as coding mode, QP, SAO offsets, etc. The section starts by describing some global settings and methods such as encoder configurations, coding mode selection for inter prediction, picture partitioning associated with tiles and rate control. The focus will then move to describe the different cost functions used by the encoder in different processing stages (e.g. motion estimation). Finally, a description on the work flow at slice, CU, PU and TU level will conclude the section.

## Encoder configurations

### Overview of encoder configurations

The HM encoder is supplied with configuration files supporting three key prediction structures, as used in the common test conditions [3]. These prediction structures are: intra-only, low-delay and random access. The reference picture list management depends on the temporal configuration.

### Intra-only configuration

For intra-only (also known as ‘all intra’ and abbreviated as ‘AI’) coding, each picture in the source material is encoded as an IDR picture. No temporal reference pictures are used. One QP value is specified in the configuration file for all slice even though this value can vary throughout the sequence because rate control and/or other perceptual optimizations are applied. Figure 6‑1 provides a graphical presentation of an intra-only configuration, where the number associated with each picture represents the encoding order.

****

Figure 6‑1. Graphical presentation of intra-only configuration.

### Low-delay configurations

Two coding configurations have been defined for testing low-delay coding performance, referred to as ‘low-delay P’ and ‘low-delay B’. For low-delay configurations, only the first picture in a video sequence is encoded as an IDR picture. Subsequent pictures are each encoded using a P-slice for low-delay P configuration or a B-slice for low-delay B configuration. For both modes, the P or B slices only reference pictures preceding the current picture in display order. For low-delay B mode, both reference lists RefPicList0 and RefPicList1 are identical.

Figure 6‑2 shows a graphical presentation of the low-delay configurations. The number associated with each picture represents the encoding order. The QP of each inter coded picture is derived by adding an offset to the QP of the intra coded picture (QPI) depending on the temporal layer.



Figure 6‑2. Graphical presentation of low-delay configuration.

### Random-access configuration

For the random-access configuration, a hierarchical B structure is used for encoding. Figure 6‑3 shows a graphical representation of a random-access configuration, where the display and encoding order for each picture is shown along with the associated QP offset relative to the QP value for intra pictures. An intra-picture is encoded at approximately one second intervals in accordance with the IntraPeriod configuration option, configured based on the frame rate of the source material. The pictures located between successive intra pictures in display order are encoded as B-pictures. The random access configuration defines a hierarchy among different B pictures whereby each hierarchical level is associated with a temporal identifier. Pictures with lower temporal id have a higher hierarchical level since they are used more often as reference for inter coding. The arrows in Figure 6‑3 depict the first reference frame (additional reference frames are not shown for the sake of simplicity) of each picture with the tip of each arrow pointing to the reference. In HEVC, the picture buffer is constrained to ensure that no more than six pictures (including the picture currently being decoded and pictures already decoded but awaiting output) are present in the DPB at any one time, as shown in Figure 6‑4. Note, however, that the HM software does not verify that this constraint is met. The picture at temporal id 0 (referred to as a ‘Generalized P and B picture’ in Figure 6‑3 below) is used as the lowest temporal layer that can refer to intra- or inter-pictures for inter prediction. The second and third temporal layers consist of referenced B pictures, while the highest temporal layer contains non-referenced B picture only. The QP of each inter-coded picture is derived by adding an offset to the QP of the intra-coded picture depending on the temporal layer as shown in the figure. The reference picture list combination is used for management and entropy coding of the reference picture index. The picture arrangement for random access in Figure 6‑3 has a group of picture length of sixteen pictures, which is currently recommended in the JCT-VC common test conditions [3].



Figure 6‑3. Graphical presentation of random-access configuration.

Figure 6‑4 shows the DPB operation for each POC in a GOP 16 random-access configuration (in accordance with ‘alternative arrangement 2’ in [15]). DPB operation is illustrated with the following coloring convention:

* Red coloured numbers indicate the frame being output after the decoding of a frame.
* Light-coloured boxes indicate the frames currently being decoded.
* Dark grey cells indicate the frames that are not used as reference frames but are waiting in the DPB to be output.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decoding order | POC |  | Frames in DPB, other than current POC | | | | | | | | | | | | | | | | | | | | | Number in DPB |
|  | -16 | -8 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 1 | 16 | -16 |  | **-3** | -2 | -1 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 | 6 |
| 2 | 8 | -16 |  |  | **-2** | -1 | 0 |  |  |  |  |  |  |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 3 | 4 | -16 |  |  |  | **-1** | 0 |  |  |  | 4 |  |  |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 4 | 2 | -16 |  |  |  |  | **0** |  | 2 |  | 4 |  |  |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 5 | 1 |  |  |  |  |  | 0 | **1** | 2 |  | 4 |  |  |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 6 | 3 |  |  |  |  |  | 0 |  | **2** | 3 | 4 |  |  |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 7 | 6 |  |  |  |  |  | 0 |  |  | **3** | 4 |  | 6 |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 8 | 5 |  |  |  |  |  | 0 |  |  |  | **4** | 5 | 6 |  | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 9 | 7 |  |  |  |  |  | 0 |  |  |  |  | **5** | 6 | 7 | 8 |  |  |  |  |  |  |  | 16 | 6 |
| 10 | 12 |  |  |  |  |  | 0 |  |  |  |  |  | **6** | 7 | 8 |  |  |  | 12 |  |  |  | 16 | 6 |
| 11 | 10 |  |  |  |  |  | 0 |  |  |  |  |  |  | **7** | 8 |  | 10 |  | 12 |  |  |  | 16 | 6 |
| 12 | 9 |  |  |  |  |  | 0 |  |  |  |  |  |  |  | **8** | 9 | 10 |  | 12 |  |  |  | 16 | 6 |
| 13 | 11 |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  | **9** | 10 | 11 | 12 |  |  |  | 16 | 6 |
| 14 | 14 |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |  | **10** | 11 | 12 |  | 14 |  | 16 | 6 |
| 15 | 13 |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |  |  | **11** | 12 | 13 | 14 |  | 16 | 6 |
| 16 | 15 |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |  |  |  | **12** | 13 | 14 | 15 | 16 | 6 |

Figure 6‑4. Reference picture lists in GOP 16 random-access configuration.

## Cost mode

The cost function minimized by the encoder to select the best coding mode for each image area can be configured using the option CostMode. This can take one of the following four strings:

* lossy – this is the standard cost equation, where cost = distortion + (bits \* lambda).
* sequence\_level\_lossless – this applies a cost in terms of bits and not distortion, where cost = (distortion / lambda) + bits. Although the cost is mathematically equivalent to the evaluation of the “lossy” cost, this is useful to ensure that when there is no distortion the cost is simply a function of bits, with no rounding errors caused by the use of lambda.
* lossless – this is equivalent to sequence\_level\_lossless, but also sets QP to the value of LOSSLESS\_AND\_MIXED\_LOSSLESS\_RD\_COST\_TEST\_QP (by default 0) (since the QP is used during the encoder search for testing intra modes). This may be deprecated in future versions in favour of the user setting the QP manually.
* mixed\_lossless\_lossy – this uses the same cost equation as sequence level lossless, but also uses a lambda evaluated at the value of the macro LOSSLESS\_AND\_MIXED\_LOSSLESS\_RD\_COST\_TEST\_QP\_PRIME (by default 4) to derive lambdas so that lossless coded blocks are not affected by QP during their encoder search. This affects the intra search during fast evaluation of intra directions and inter search during evaluation of motion vector cost.

## Cost Functions

Various cost functions are used in the HM encoder to determine costs used in making encoder decisions. This section documents the cost functions used in the encoding process of the HM software.

### Sum of Square Error (SSE)

The difference between two blocks with the same block size is produced using

Diff(i,j) = BlockA(i,j) - BlockB(i,j) (7‑1)

SEE is computed using the following equation:

 (7‑2)

### Sum of Absolute Difference (SAD)

SAD is computed using the following equation:

 (7‑3)

### Hadamard transformed SAD (SATD)

Since the transformed coefficients are coded, an improved estimation of the cost of each mode can be obtained by estimating DCT with the Hadamard transform.

SATD is computed using:

 (7‑4)

The Hadamard transform flag can be turned on or off. SA(T)D refers to either SAD or SATD depending on the status of the Hadamard transform flag.

SAD is used when computing full-pel motion estimation while SA(T)D is used for sub-pel motion estimation.

### RD cost functions

#### Lagrangian constant values

In the HM encoder, lambda values that are used for cost computation are defined as:

 (7‑5)

pred  (7‑6)

 (7‑7)

, for non-referenced pictures

, for referenced pictures

represents weighting factor dependent to encoding configuration and QP offset hierarchy level of current picture within a GOP, as specified in Table 7‑1. Note that the value of derived from Table 6‑1 is further modified by multiplying 0.95 when SATD based motion estimation is used.

Table 6‑1. Derivation of Wk.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K | QP offset hierarchy level | Slice type | Referenced |  |
| 0 | 0 | I | - | 0.57 |
| 1 | 0 | P or B | 1 | RA: 0.442  LD: 0.578 |
| 2 | 1, 2 | P or B | 1 | RA: 0.3536 \* Clip3( 2.0, 4.0, (QP-12)/6.0 )  LD: 0.4624 \* Clip3( 2.0, 4.0, (QP-12)/6.0 ) |
| 4 | 3 | B | 0 | RA: 0.68 \* Clip3( 2.0, 4.0, (QP-12)/6.0 ) |

#### Weighting factor for chroma component

The following weighting parameter *w*chroma is used to derive lambda value to be used for chroma-specific decisions in RDOQ and SAO processes.

 (7‑8)

With this parameter, is obtained by

 (7‑9)

Note that the parameter *w*chroma is also used to define the cost function used for mode decisions in order to weight the chroma part of SSE.

#### SAD based cost function for prediction parameter decision

The cost for prediction parameter decision *J*pred,SAD is specified by the following formula.

*J*pred,SAD =*SAD +* λpred \* *B*pred*,*  (7‑10)

where *B*pred specifies bit cost to be considered for making decision, which depends on each decision case. λpred and SAD are defined in the section 7.1.4.1 and 7.1.2, respectively.

#### SATD based cost function for prediction parameter decision

The cost for motion parameter decision *J*pred,SATD is specified by the following formula.

*J*pred,SATD =*SATD +* λpred \* *B*pred*,*  (7‑11)

where *B*pred specifies bit cost to be considered for making decision, which depends on each decision case. λpred and SATD are defined in the section 7.1.4.1 and 7.1.3, respectively.

#### Cost function for mode decision

The cost for mode decision *J*mode is specified by the following formula.

*J*mode =(*SSE*luma*+ w*chroma *\*SSE*chroma)*+* λmode \* *B*mode, (7‑12)

where *B*mode specifies bit cost to be considered for mode decision, which depends on each decision case. λmode and SSE are defined in the section 7.1.4.1 and 7.1.1, respectively.

### Lambda modifiers

The HM encoder supports modifying lambda for performing mode decisions. Lambda can be modified independently for each temporal layer. A separate lambda modifier for intra slices can also be specified, also independently for each temporal layer.

Lambda modifiers are specified for each temporal layer using LambdaModifier0, LambdaModifier1, LambdaModifier2, LambdaModifier3, LambdaModifier4, LambdaModifier5, and LambdaModifier6. Lambda modifiers for intra pictures are specified providing LambdaModifierI with a comma separated list of lambda modifiers, specifying the lambda modifier for each temporal layer.

IQPFactor specifies the intra QP Factor for lambda computation. If negative, use the following formula:

0.57\*(1.0 - Clip( 0.0, 0.5, 0.05\*(isField ? (GOPsize - 1)/2 : GOPsize - 1) )) (7‑13)

## Slice and tile partitioning operation

The HM encoder can partition a picture into several slices and tiles. Slice and tile boundaries are aligned to CTU boundaries. The HM encoder has three ways of determining slice size: by specifying the maximum number of CTUs in a slice, by specifying the number of bytes in a slice, and by specifying the number of tiles in a slice. This mode is controlled by SliceMode, with SliceArgument used to control the maximum number of CTUs, bytes or tiles in a slice. Additionally, the controls for a slice segment are separate to those of a slice, with SliceSegmentMode and SliceSegmentArgument used. Separate controls allow bitstreams to contain a mixture of independent and dependent slice segments. Note that if the mode is set to limit the slice size by the maximum number of bytes per slice or by the maximum number of CTUs per slice, then a given slice will always terminate at the end of every tile and in addition, if a slice starts mid-way along a CTU row of a tile then it is terminated within the same row when wavefront parallel processing is enabled; similarly for slice segments.

Tiles are configured by the number of columns and rows (NumTileColumnsMinus1 and NumTileRowsMinus1 respectively), and are either specified as having a uniform spacing (TileUnformSpacing equal to 1), or the individual tile column widths (using TileColumnWidthArray) and tile row heights (using TileRowHeightArray) are used. Tiles do not need to be enabled: they are automatically enabled if more than one tile row or column is specified. The five encoder parameters correspond to syntax elements *num\_tile\_colums\_minus1*, *num\_tile\_rows\_minus1*, *uniform\_spacing\_flag*, *col\_width\_minus1[i]*, and *row\_height\_minus1[i]*.

The top level operation occurs in the function TEncGOP::compressGOP(), as follows:

* Loop over each picture in current GOP:
  + For each slice in current picture (identified by nextCtuTsAddr):
    - Call TEncSlice::precompressSlice()
    - Call TEncSlice::compressSlice()
  + Perform loop filtering
  + Perform entropy coding
    - Encode each slice

The function TEncSlice::precompressSlice() has effect testing when delta QP rate-distortion is in use. Multiple QPs are tested. For each tested QP, TEncSlice::compressSlice() is called and the optimal QP for rate-distortion cost is selected.

The function TEncSlice::compressSlice() performs the following steps:

* Weighted prediction parameter estimation by calling WeightPredAnalysis::xCalcACDCParamSlice():
  + Loop over each colour component:
    - Calculate normalized DC value over picture.
    - Calculate normalized AC value as abs(sampleValue − normalized DC) over picture.
    - If WP enabled for P-slice or B-slice, call WeightPredAnalysis:: xEstimateWPParamSlice() as described in section 6.10.4.
* Loop over every CTU in the slice segment (may terminate sooner if byte limit on the slice segment):
  + Invoke TEncCu::compressCtu().
  + Then invoke TEncCu::encodeCtu().

## Derivation process for CU-level and PU-level coding parameters

### Intra prediction mode and parameters

The HM encoder selects an intra prediction mode for a PU as follows:

1. A candidate mode derivation step tests all possible prediction modes for the luma PB with an approximate prediction cost *J*pred,SATD specified in the section 6.5.4.4. A pre-determined number of intermediate candidates are found for each PU size (8 for 4×4 and 8×8 PUs, 3 for other PU sizes). In this step, the number of coded bits for an intra prediction mode is set to *B*pred.
2. An RD optimization step, using the coding cost *J*mode specified in the section 6.3.4.5, is applied to the previously determined candidate modes. During this step, prediction parameters and coefficients for luma component of the PU are accumulated into *B*mode. Regarding the chroma PB mode decision, all possible intra chroma prediction modes are evaluated through RD decision process, where coded bits for intra chroma prediction mode and chroma coefficient are used as *B*mode.

#### Rate-distortion penalty for intra coding

This tool is enabled with the configuration option RDpenalty and provides a fast mode decision for intra coding. Configuration option RDpenalty can take three values: 0 (i.e. disabled), 1 and 2. When set to one the encoder avoids splitting a transform unit when its size is smaller than 16×16 and its associated slice is not intra. When RDpenalty is set to 2, transform units with size 32×32 in non intra slices are not checked and the RQT search moves to the next level of recursion.

### Inter prediction mode and parameters

#### Derivation of motion parameters

An inter-predicted CU is segmented into one or more inter-predicted PUs according to the partition mode (“PartMode”) of the CU. Each PU has a set of motion parameters consisting of one motion vectors per reference picture and corresponding reference picture indices (ref\_idx\_lX) and prediction direction index (inter\_pred\_flag).

An inter-predicted CU can be encoded with one of the following coding modes (“PredMode”):

* MODE\_SKIP
* MODE\_INTER

For the MODE\_SKIP case, the partition mode of the CU is implicitly PART\_2Nx2N, and thus sub-partitioning to smaller PUs is not allowed.

For the MODE\_INTER case, up to eight further types of partitioning to smaller PUs are provided for a CU coded with MODE\_INTER (with additional restrictions according to the CU size). The PredMode and PartMode are signalled by a CU level syntax element “part\_type”, as specified in Table 7-10 of the HEVC specification. For a MODE\_INTER CU other than those having maximum depth, seven partition modes (PART\_2Nx2N, PART\_2NxN, PART\_Nx2N, PART\_2NxnU, PART\_2NxnD, PART\_nLx2N and PART\_nRx2N) can be selected. PART\_NxN can only be chosen when the CU size is greater than 8x8 and the CU depth is at the maximum configured CU depth level. For each PU, PU-based Motion Merging (merge mode) or normal inter prediction with actually-estimated motion parameters (inter mode) can be used. This section describes how luma motion parameters are obtained for each PU. The chroma motion vectors are derived from the luma motion vector of corresponding PU according to the process specified in section 8.4.2.1.10 of the HEVC specification, with the reference picture index and prediction direction index set according to the corresponding values for the luma motion parameters.

##### Motion vector prediction

For each PU, the best motion vector predictor is computed with the process specified as follows. Firstly, a set of motion vector predictor candidates for RefPicListX are derived with normative process specified in section 8.4.2.1.7 of the HEVC specification, by referring to motion parameters of neighbouring PUs. Then, the best one from the candidate set is determined by a criterion that selects a motion vector predictor candidate that minimizes the cost *J*pred,SAD specified in the section 6.3.4.3, with setting the bits for an index specifying each motion vector predictor candidate to *B*pred. The index corresponding to the selected best candidate is assigned to the mvp\_idx\_lX.

##### CU coding with MODE\_SKIP

In the case of skip mode (i.e., PredMode == “MODE\_SKIP”), motion parameters for the current CU(i.e., PART\_2Nx2N PU) are derived by using merge mode. In this case, the motion parameters are determined by checking all possible merge candidates derived by the normative process specified in section 8.4.2.1.1 to 8.4.2.1.5 of the HEVC specification, and selecting the best set of motion parameters that minimizes the cost *J*mode specified in the section . In this case, *B*mode includes coded bits for skip\_flag and merge\_idx that signals position of the PU having the best motion parameters to be used for the current PU. Since prediction residual is not transmitted for skip mode, SSE is obtained by inter prediction samples.

##### CU coding with MODE\_INTER

When a CU is coded with MODE\_INTER, motion parameter decision for each PU is performed first based on the ME cost *J*pred,SATD specified in the section 6.3.4.4.

For merge mode case, the motion parameter decision starts with checking availabilities of all neighbouring PUs to form merge candidates according to the normative process specified in the sections 8.4.2.1.1 to 8.4.2.1.5 of the HEVC specification. If there is no available merge candidate, the HM encoder skips cost computation for merge mode and does not choose merge mode for the current PU. Otherwise (i.e., if there is at least one merge candidate), the ME cost *J*pred,SATD specified in the section 6.3.4.4 is computed for all possible PUs as merge candidate and the best one is selected as the best motion parameters for the PU predicted with merge mode. SATD between source and prediction samples is used as distortion factor, and bits for merge\_idx is set to *B*pred.

For the inter mode case, the best motion parameters are derived by invoking motion estimation process specified in the section 6.5.2.2. During the motion estimation process, the best motion parameters are obtained based on the cost function *J*pred,SATD specified in the section 6.3.4.4, which is comparable with the cost of motion parameter derivation for merge mode. SATD between source and prediction samples is used as distortion factor, and bits for inter\_pred\_flag, ref\_idx\_lX, mvd\_lX and mvp\_idx\_lX are set to *B*pred.

After both of the best motion parameters are obtained, the best motion parameters are determined by comparing them and taking the better one that results in lower cost.

#### Motion estimation

To derive the motion vector(s) for each PU, a block matching algorithm is performed in the HM encoder. The motion vector accuracy supported in HEVC is quarter-pel. To generate half-pel and quarter-pel accuracy samples, interpolation filtering is performed for reference picture samples. Instead of searching all the positions for quarter-pel accuracy motion, a motion vector aligned to integer-pel positions is first obtained. For the half-pel search, the eight sample points at half-pel accuracy around the motion vector which has the minimum cost are searched. Similarly, for the quarter-pel search, the eight sample points at quarter-pel accuracy around the motion vector which has the minimum cost so far are searched. The motion vector which has the minimum cost is selected as the motion vector of the PU. To get the cost, SAD is used for integer-pel motion search and SA(T)D is used for half-pel and quarter-pel motion search. The rate for motion vector is obtained by utilizing a pre-calculated rate table. In the following sub-sections, algorithms for integer-pel motion search are described.

##### Integer-pel accuracy motion search

For AMVP, find the best candidate MV predictor for each ref\_idx and ref\_pic\_list using xEstimateMvPredAMVP(), called from predInterSearch().

One of four supported integer search algorithms is selected using to the FastSearch configuration option, with the following options being available:

* Full search
* Diamond search
* Selective search
* Enhanced diamond search

Then, aspects of the above selected search algorithm are modified according to the fast encoder modes (FEN) configuration option:

* Fast mode disabled
* Fast mode 1
* Fast mode 2
* Fast mode 3

The default search range for the first search in the HM encoder is 96 integer pixels, however the CTC [3] uses a value of 64. A search window is defined according to the search range, relative to the best candidate MV predictor.

Firstly an integer-pel search is performed, followed by a fractional-pel refinement search. These searches are described in more detail below:

**Full search**

When the full search is selected, every integer location within the defined search range is tested to find the best candidate motion vector.

When fast mode 1 or 3 is selected, a subsampled SAD is used to speed up distortion measurement. The SAD is subsampled to check only every second row for blocks with greater than eight rows.

One predictor is used in the search; set to the best candidate MV predictor for the considered reference picture.

**Diamond and extended diamond searches**

Figure 6‑5 shows the method followed when the diamond search or the enhanced diamond search is selected.



Figure 6‑5. Diamond and enhanced diamond search flowchart.

As seen in Figure 6‑5, a best starting point is selected from a set of candidates as generated by fillMvpCand(). If only one candidate was generated, that becomes the starting point. Otherwise, the candidate resulting in minimal cost, with distortion (SAD) measured in the luma channel, is selected as the optimal motion vector predictor. See the function xEstimateMvPredAMVP(). The motion vector corresponding to the PART\_2Nx2N CU is also tested for other partition modes of the CU. If the enhanced diamond search is enabled then the neighbour predictors PRED\_A, PRED\_B and PRED\_C are tested using xTZSearchHelp(). The zero motion vector is also tested using xTZSearchHelp(). The result of this testing is selection of the motion vector predictor.

Then, a first search is performed to select an integer-pel accuracy motion vector. The first search begins with an iterative 8-point search.

The 8-point search uses a distance parameter *iDist* to control the distance of the tested points relative to the chosen starting location. Additionally, the pattern of the test points also depends on the value of *iDist*. When *iDist* is equal to one, relative to the starting location (red), eight neighbour points (light and dark blue) are tested, as shown in Figure 6‑6. Testing of the light blue points is enabled according to a function parameter and thus is dependent upon the invocation of the 8-point search. If disabled, then only four points (i.e. dark blue points) are tested.



Figure 6‑6. 8-point search with iDist equal to one.

For values of *iDist* from two to eight, the search pattern is shown in Figure 6‑7 is used.



Figure 6‑7. 8-point search with iDist from two to eight.

For values of *iDist* greater than eight, the search pattern shown in Figure 6‑8 is used. This search pattern includes diagonal test points located at horizontal and vertical offsets of ±¼, ±½ and ±¾ of *iDist*, rounded to integer accuracy.



Figure 6‑8. 8-point search with iDist greater than eight.

Alternative square test patterns are also available when constant value bFirstSearchDiamond is set to false.

The iterative 8-point search is performed by iterating a variable *iDist* over powers of two from one to the highest power of two not exceeding the specified search range. For each value of *iDist*, a set of points is tested as described above. Thus, the iterative 8-point search is the union of the sets of points above over all iterated values of *iDist*. An early exit for the *iDist* loop is available using the FastMEAssumingSmootherMVEnabled configuration option. When enabled, the *iDist* loop terminates after a maximum of three iterations from the last iteration for which a new best motion vector was found. For the diamond search, the corner points (light blue in Figure 6‑6 are not tested) and for the enhanced diamond search, these points are tested as part of the iterative 8-point search.

For the enhanced diamond search, an iterative 8-point search (with corners not tested) is performed, with *iDist* iterating to a maximum of half the defined search range, around the (0, 0) position.

If *iDist* is equal to 1, then, due to the previous search at *iDist* equal to 0, there are only two untested points. These two points are tested.

Then, for the next stage of the first search of a diamond search, perform a raster search and for the enhanced diamond search, perform an adaptive raster search.

The raster search (enabled by constant value bEnableRasterSearch) is performed if the distance *iDist* corresponding to the current best motion vector is greater than five. When performing the raster search, *iDist* for the best motion vector is set to five, and the raster search tests blocks along a sparse grid within the defined search window, testing blocks at every five luma samples horizontally and vertically.

For the adaptive raster search, the search window is halved in size). If the *iDist* of the current best motion vector is less than or equal to five then the current best distance is set to six. Then a raster search is performed using the halved search window, also testing blocks located every five luma samples horizontally and vertically.

After the first search completes, a refinement search is performed. By default, a star refinement search is performed, although a raster refinement search is also available in the HM source code. The refinement search only occurs if the *iDist* associated with the best motion vector is nonzero. If so, an iterative 8-point search is performed, starting from *iDist* equal to one, and doubling *iDist* until the search range is exceeded. By default the diamond pattern of test points is used, however a square pattern can also be used.

**Selective search**

The selective search begins by setting the median predictor value as the search starting point and the best MV. Then, predictors PRED\_A, PRED\_B and PRED\_C are tested to see if they provide a better starting point, in which case the preferred predictor is selected. Then the zero motion vector is tested similarly.

For an initial search, an initial search window is set, centred at the selected best predictor and one quarter the width and height of the defined search range. Then, the initial search window is sparsely searched, such that every fourth integer pel horizontally and vertically is tested. At each tested point, two 8-point diamond searches are performed, at *iDist* equal to one and two and with the corner four points disabled.

If the L1 distance between the best motion vector and the chosen predictor exceeds a constant of eight, then a full search is performed over a search window relative to the predictor and sized according to the search range. Otherwise, if *iDist* resulting from the initial search is equal to one, a refinement search is performed. The refinement search performs an iterative 8-point diamond search, with corners always disabled and no early exit tests active.

###### Fast encoder modes

Table 6‑2 provides a summary of the approaches selected according to the fast encoder mode in use.

Table 6‑2. Fast encoder mode summary.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Fast mode 1** | **Fast mode 2** | **Fast mode 3** |
| Subsampled SAD for selective searches | Enabled | Disabled | Enabled |
| Subsampled SAD for searches other than selective searches | Enabled | Disabled | Enabled |
| Bi-predictive iterations reduction | Enabled | Enabled | Disabled |
| Reference picture list adaptive selection | Enabled | Enabled | Disabled |

**SAD for selective searches**

To speed up the SAD operation, a subsampling approach is provided where the number of rows tested is reduced.

When the selective search is used and RestrictMESampling is enabled, this approach operates as follows:

Variable iSubShift results in testing every *n*th row, such that *n* = 1 << iSubShift. The variable iSubShift is initialized according to the number of rows in the block under test, such that the spacing between tested rows increases with the block size. Then, the SAD for the block under test is calculated iteratively, with iSubShift decremented on each iteration, until the SAD of the current iteration does not improve upon the best SAD resulting from earlier iterations (with adjustments to compensate for iSubShift variation).

**SAD for searches other than selective searches**

Depending on the fast encoder mode, the SAD can be subsampled to either check every row or every second row. Table 6‑2 documents control of this option.

**Bi-predictive iterations reduction**

By default, the bi-predictive search performs four iterations. For fast modes 1 and 3, this is reduced to one iteration.

**Reference picture list adaptive selection**

By default, the reference picture list to be used for a given iteration of the bi-predictive search is alternatively selected with consecutive iterations, i.e., iterations 0, 2, …, use reference picture list 0 and iterations 1, 3, …, use reference picture list 1. When reference picture lists are adaptively selected, the list to be tested for the bi-predictive search is the opposing list to the best list resulting from the uni-predictive search.

##### Fractional-pel search strategy

Fractional refinement is performed in two steps, firstly testing half-pel positions and secondly testing quarter-pel positions. The eight half-pel positions surrounding the current best integer motion vector are tested, with the best (if any) being selected. Then, the eight quarter-pel positions surrounding the current best integer or half-pel motion vector are tested.

##### Bi-predictive search strategy

The objective of the bi-predictive motion search is to produce two motion vectors which result in minimum error between the original block (O) and the predicted block P with two predictions P0 and P1, such that P = P0 + P1. In the HM encoder, an iterative uni-predictive search is implemented, providing a reasonable compromise between search duration and minimising error. The bi-predictive search steps implemented in the HM encoder are as follows:

1. Search P1 which produces minimum error with (2O - P0), where O represents original block and P0 means predictor produced by the first motion vector. P0 is fixed in this step. To derive motion vector for P1, a uni-predictive motion search is performed with the reference samples set to (2O - P0).
2. Search P0 which produces minimum error with (2O − P1), where O represents original block and P1 means predictor produced by the second motion vector. P1 is the predictor obtained in step 1) and fixed in this step. To get P0, uni-predictive search is utilized after setting (2O − P1) as reference samples.
3. Repeat steps (1) and (2) until maximum number of iterations is reached. The maximum number of iterations is set to four, unless the fast search option is enabled (in which case steps (1) and (2) are performed once only).

Behaviour of the bi-predictive search is also influenced by the fast encoder mode selection, see section 6.5.2.2.1.1 for details.

#### Decision process on AMP mode evaluation procedure

To speed up the HM encoder for testing AMP modes, early exit conditions are present as are heuristics to reduce the set of tested modes. When any such early exit condition is met, the additional motion estimation search for AMP can be skipped. Conditions of mode test-set reduction are based on two values: the best partition mode (PartMode) before AMP modes are evaluated and the PartMode and prediction mode (PredMode) at the lower level in the CU quadtree, the so-called parent CU, which contains the current PU. The conditions and actions are specified in Table 6‑3.

Table 6‑3. Conditions and actions for fast AMP mode evaluation.

|  |  |
| --- | --- |
| **Conditions** | **Actions** |
| The best PartMode is SIZE\_2NxN | Try SIZE\_2NxnU and SIZE\_2NxnD |
| The best PartMode is SIZE\_Nx2N | Try SIZE\_nLx2N and SIZE\_nRx2N |
| The best PartMode is 2Nx2N &&  !merge mode && ! skip mode | Try all AMP modes |
| PartMode of parent CU is AMP mode | Try merge mode only for all AMP modes |
| PartMode of parent CU is PART\_2Nx2N && parent CU is not skipped | Try merge mode only for all AMP modes |
| PredMode of parent CU is intra && the best PartMode is SIZE\_2NxN | Try merge mode only for SIZE\_2NxnU and SIZE\_2NxnD |
| PredMode of parent CU is intra && the best PartMode is SIZE\_Nx2N | Try merge mode only for SIZE\_nLx2N and SIZE\_nRx2N |
| Size of current CU is 64x64 | No AMP modes are evaluated |

### Intra/Inter/PCM mode decision

A recursive search from the CTU down the CU hierarchy is performed. At a given hierarchy level, a loop over QPs according to the base QP provided to HM and a range implied from the delta QP parameter is performed.

Then, for the considered CU, the following mode decision process is conducted in the HM encoder for inter-prediction modes. A corresponding flowchart is also shown in Figure 6‑9.

1. Coding costs (*J*mode) for MODE\_INTER with PART\_2Nx2N is computed and *J*mode is set to minimum CU coding cost *J*.
2. Check if motion vector difference of MODE\_INTER with PART\_2Nx2N is equal to (0, 0) and MODE\_INTER with PART\_2Nx2N contains no non-zero transform coefficients (Early\_SKIP condition). If both are true, proceed to 17 with setting the best interim coding mode as MODE\_SKIP. Otherwise, proceed to 3.
3. Check if MODE\_INTER with PART\_2Nx2N contains no non-zero transform coefficients (CBF\_Fast condition). If the condition is true, proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_2Nx2N. Otherwise, proceed to 4.
4. *J*mode for MODE\_SKIP is evaluated and *J* is set equal to *J*mode if *J*mode < *J.*
5. Check if the CU is at the maximum depth and the current CU size is not 8×8. If the conditions are true, proceed to 6. Otherwise, proceed to 7.
6. *J*mode for MODE\_INTER with PART\_NxN is evaluated and *J* is set equal to *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_NxN contains no non-zero transform coefficients (CBF\_Fast condition). If the condition is true, proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_NxN. Otherwise, proceed to 7.
7. *J*mode for MODE\_INTER with PART\_Nx2N is evaluated and *J* is set equal to *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_Nx2N containsno non-zero transform coefficients (CBF\_Fast condition). If the condition is true*,*proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_Nx2N. Otherwise, proceed to 8.
8. *J*mode for MODE\_INTER with PART\_2NxN is evaluated and *J* is set equal to *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_2NxN containsno non-zero transform coefficients (CBF\_Fast condition). If the condition is true*,*proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_2NxN. Otherwise, proceed to 9.
9. Invoke a process to determine AMP mode evaluation procedure specified in 6.8.2.3. Output of this process is assigned to TestAMP\_Hor and TestAMP\_Ver. TestAMP\_Hor specifies whether horizontal AMP modes are tested with specific ME or tested with merge mode or not tested. TestAMP\_Ver specifies whether vertical AMP modes are tested with specific ME or tested with merge mode or not tested.
10. If TestAMP\_Hor indicates that horizontal AMP modes are tested, MODE\_INTER with PART\_2NxnU is evaluated with procedure suggested by TestAMP\_Hor and *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_2NxnU containsno non-zero transform coefficients (CBF\_Fast condition). If the condition is true*,*proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_2NxnU. Otherwise, MODE\_INTER with PART\_2NxnD is evaluated with procedure suggested by TestAMP\_Hor and *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_2NxnD contains no non-zero transform coefficients (CBF\_Fast condition). If the condition is true, proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_2NxnD. Otherwise, proceed to 11.
11. If TestAMP\_Ver indicates that vertical AMP modes are tested, MODE\_INTER with PART\_nLx2N is evaluated with procedure suggested by TestAMP\_Ver and *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*. After that, check if MODE\_INTER with PART\_nLx2N contains no non-zero transform coefficients (CBF\_Fast condition). If the condition is true, proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_nLx2N. Otherwise, MODE\_INTER with PART\_nRx2N is evaluated with procedure suggested by TestAMP\_Ver and *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*. After that, checkif MODE\_INTER with PART\_nRx2N contains no non-zero transform coefficients (CBF\_Fast condition). If the condition is true, proceed to 17 with setting the best interim coding mode as MODE\_INTER with PART\_nRx2N. Otherwise, proceed to 12.
12. MODE\_INTRA with PART\_2Nx2N is evaluated by invoking the process specified in 6.8.1, only when at least one or more non-zero transform coefficients can be found by using the best interim coding mode. *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*.
13. Check if the current CU depth is maximum, If the condition is true, proceed to 14. Otherwise, proceed to 15.
14. MODE\_INTRA with PART\_NxN is evaluated by invoking the process specified in 6.8.1, only when the current CU size is larger than minimum TU size. The resulting coding cost *J*mode is set to *J* if *J*mode < *J*.
15. Check if the current CU size is greater than or equal to the minimum PCM mode size specified by the log2\_min\_pcm\_coding\_block\_size\_minus3 value of SPS parameter. If the condition is true, proceed to 16. Otherwise, proceed to 17.
16. Check if any of the following conditions are true. If the condition is true, PCM mode is evaluated and *J* is set equal to the resulting coding cost *J*mode if *J*mode < *J*.

* Bit cost of *J* is greater than that of the PCM sample data of the input image block.
* *J* is greater than bit cost of the PCM sample data of the input image block multiplied by λmode.

1. Update bit cost *B*mode by adding bits for CU split flag and re-compute minimum coding cost *J*.
2. Check if the best interim coding mode is MODE\_SKIP (Early\_CU condition). If the condition is true, do not proceed to the recursive mode decision at next CU level. Otherwise, go to next CU level of recursive mode decision if the current CU depth is not maximum.



Figure 6‑9. The schematic of Intra/Inter/PCM mode decision.

For the computation of *J*mode except for PCM mode, residual signal is obtained by subtracting intra or inter prediction samples from source samples and is coded with transform and quantization with quadtree TU partitioning as specified in the section 6.9. Bits for side information (skip\_flag, merge\_flag, merge\_idx, pred\_type, pcm\_flag, inter\_pred\_flag, reference picture indices, motion vector(s), mvp\_idx, intra prediction mode signaling) and residual coded data are considered as *B*mode. *SSE*luma and *SSE*chroma are obtained by using local decoded samples, except for MODE\_SKIP case where prediction sample is used as local decoded samples.

For the computation of *J*mode for PCM mode, bits for side information (skip\_flag, pred\_type, pcm\_flag, pcm\_alignment\_zero\_bit) and PCM sample data are considered as *B*mode. *SSE*luma and *SSE*chroma are set to 0. (Note that in current test conditions, the PCM mode decision processes in (15) and (16) are skipped since the minimum PCM mode size is 128).

This CU level mode decision is recursively performed for each CU depth and final distribution of CU coding modes is determined at CTU level.

### Adaptive QP

This tool varies the quantization parameter for each coding unit to provide improved perceived image quality. QP variation is performed using the same technique originally implemented in the MPEG-2 TM5 which works according to the following rationale: lower QP values are used on smooth image areas while higher values on highly active blocks. The activity of each CU is measured by the variance of its luma samples. More precisely, given a CU with size 2N×2N, the luma variance of its four sub blocks with size N×N is computed first. Let *σ*2(*i*) denote the luma sample variance for sub block *i*. The CU activity (*act*CU) is then computed as:

,

In order to increase QP values on highly active image areas and decrease in smooth ones, the quantity *actCU* is then normalized with respect to the average activity measured over all coding units with size 2N×2N inside one picture. Let *actf* denote the average activity for all CUs with size 2N×2N belonging to picture *f*. The normalized CU activity *norm\_actCU* is then given by:

,

where *s* denotes the scaling factor associated to the QP adaptation parameter (QPA) and computed as:

.

The value for QPA is provided as input using the configuration option MaxQPAdaptationRange and has default value of 6. Finally, the coding unit QP is adjusted according to:

,

where QPbase denotes the QP value for the slice where the coding unit belongs to and ⎣⋅⎦ operator returns the largest integer smaller than or equal to the argument. The adaptive QP tool is enabled by the configuration option AdaptiveQP. The minimum CU size at which QP can be adapted according to its normalized activity is specified by the configuration option MaxCuDQPDepth whose value should always be less than the maximum CU depth.

## Derivation process for TU-level coding parameters

### Residual quadtree partitioning

The residual quadtree is a recursive representation of the partitioning of a coding unit into transform units.

The encoding process for intra-coded coding units can be summarized as follows.

– The luma intra prediction mode (or modes for intra\_split\_flag equal to 1) is determined using the residual coding with the largest applicable transform size.

– Given the determined luma intra prediction mode (or modes for intra\_split\_flag equal to 1), the transform tree and the corresponding luma transform coefficient levels are determined using an exhaustive subdivision process, taking into account the maximum allowed transform hierarchy depth and considering only the luma component.

– The chroma intra prediction mode and the corresponding chroma transform coefficient levels are determined given the determined transform tree.

The encoding process for inter-coded coding units can be summarized as follows.

– The transform tree and the corresponding luma and chroma transform coefficient levels are determined using an exhaustive subdivision process, taking into account the maximum allowed transform hierarchy depth and considering both the luma component and the chroma components.

### Rate-distortion optimized quantization

RDOQ is enabled by the encoder option --RDOQ and performs soft decision quantization for each transform coefficient by minimizing a rate-distortion Lagrangian cost function. The RDOQ implemented in HM consists of three steps: 1) selection of the optimal level for each coded coefficient, 2) decision whether all coefficients belonging to a given coefficient group (CG) of 4×4 transform coefficients should be set to zero and 3) selection of the last significant coefficient position. Over a given transform block RDOQ operates independently on each CG. Within each CG, RDOQ processes the transform coefficients according to the scanning order used by the entropy encoder, in particular starting from the last significant coefficient to processing towards the lowest frequency coefficient for the CG. This order generally results in processing coefficients of increasing magnitude as the scan progresses. Operation of the three aforementioned steps is further described as follows:

* **Step 1** (Selection of the best level for each coded coefficient): Over a CG each coefficient *ci* (for *i* = 0,…,15) is quantized to the following three level values: 0, *lfloor* and *lceil*. The value for *lfloor* is the coefficient value resulting from quantization with the selected QP. Conversely, *lceil* = *lfloor* + 1. For each level value *l*, the following Lagrangian cost function *J* is computed:



where *λ* denotes the Lagrangian multiplier computed as described in Section 6.3.4.1, *D*(*ci, l*) is the distortion measured when *ci* is quantized to *l* and *R*(*l*) is the (estimated) coding rate associated to *l*. The value for *l* (i.e. 0, *lfloor* or *lceil*) which minimizes *J* is selected as the best value. Then the process advances to the next coefficient in the CG in the scan order. In order to keep low computational complexity during this first step, the coding rate *R*(*l*) is estimated by tabularized values of entropy of the probabilities corresponding to states in the CABAC engine.

* **Step 2** (Decision whether all coefficients belonging to a given coefficient group should be set to zero): In this step RDOQ checks whether setting all levels for a given CG minimizes the Lagrangian cost function *J*. If this is the case, that particular CGs will be coded as all zero and the corresponding coefficient group flag set accordingly. The process is repeated for all CGs belonging to the current transform block. As for Step 1, also here the value for the coefficient group flag is estimated by the internal state values of the CABAC engine. Moreover, to further reduce encoder complexity, the distortion associated with all levels equal to zero is accumulated while all transform coefficients are visited in Step 1.
* **Step 3** (Selection of the last significant coefficient position): During this step RDOQ decides the (*x*, *y*) coordinates for the position of the last significant coefficient beyond which all levels are considered zero during entropy coding. The search for the optimal position starts from the position of the last significant coefficient obtained after Steps 1 and 2 and continues by pushing the last significant coefficient position along the scan order, i.e. towards the top left corner of the current transform block. For each tested position, the Lagrangian cost function *J* is evaluated. To minimize the complexity associated with this step, 1D arrays are used to store the Lagrangian costs associated with both the case when a coefficient level is set to zero and when the coefficient level is left as decided by Steps 1 and 2. These variables are updated according to best position found at each iteration.

The workflow associated with Steps 1 to 3 can summarized in the following pseudo code:

For *i* = 0 to all CGs belonging to the current transform block do

// Step 1

Set the Lagrangian cost associated with the coded transform coefficient *JCoded* = 0

For each transform coefficient *c*(*j*) with *j* = 0 to 15 do

Quantize *c*(*j*) to levels (0, *lfloor* or *lceil*) and computed for each level the associate Lagrangian cost *J*(*cj*, *l*)

Select the optimal level (*lopt*) which minimizes *J*(*cj*, *l*) and add it to *JCoded*

Endfor

// Step 2

Set all levels in CG(*i*) to zero, entropy code the coefficient group flag, measure distortion and compute *JZero*

If *JZero* < *JCoded* then set all levels in CG(*i*) to zero

Endfor

// Step 3

Select the optimal last significant coefficient position

RDOQ can be used also for those transform units where the transform skip is selected. In this case the processing is identical and this option can be enabled or disabled by the encoder option --RDOQTS.

### Quantization rounding for residual DPCM

Over blocks where both transform skip and residual DPCM (RDPCM) are applied, the quantization rounding offset is different from the one used in spatially transformed blocks. More precisely, the HM-RExt software applies a dead zone uniform quantizer with quantization step *Q* and rounding offset *α* as shown in Figure 6‑10. For blocks where spatial transformation is applied, *α* is equal to 1/3 and 1/6 for intra and inter coding modes, respectively. Conversely, over blocks where transform skip and RDPCM are used, the offset *α* is set equal to1/2.

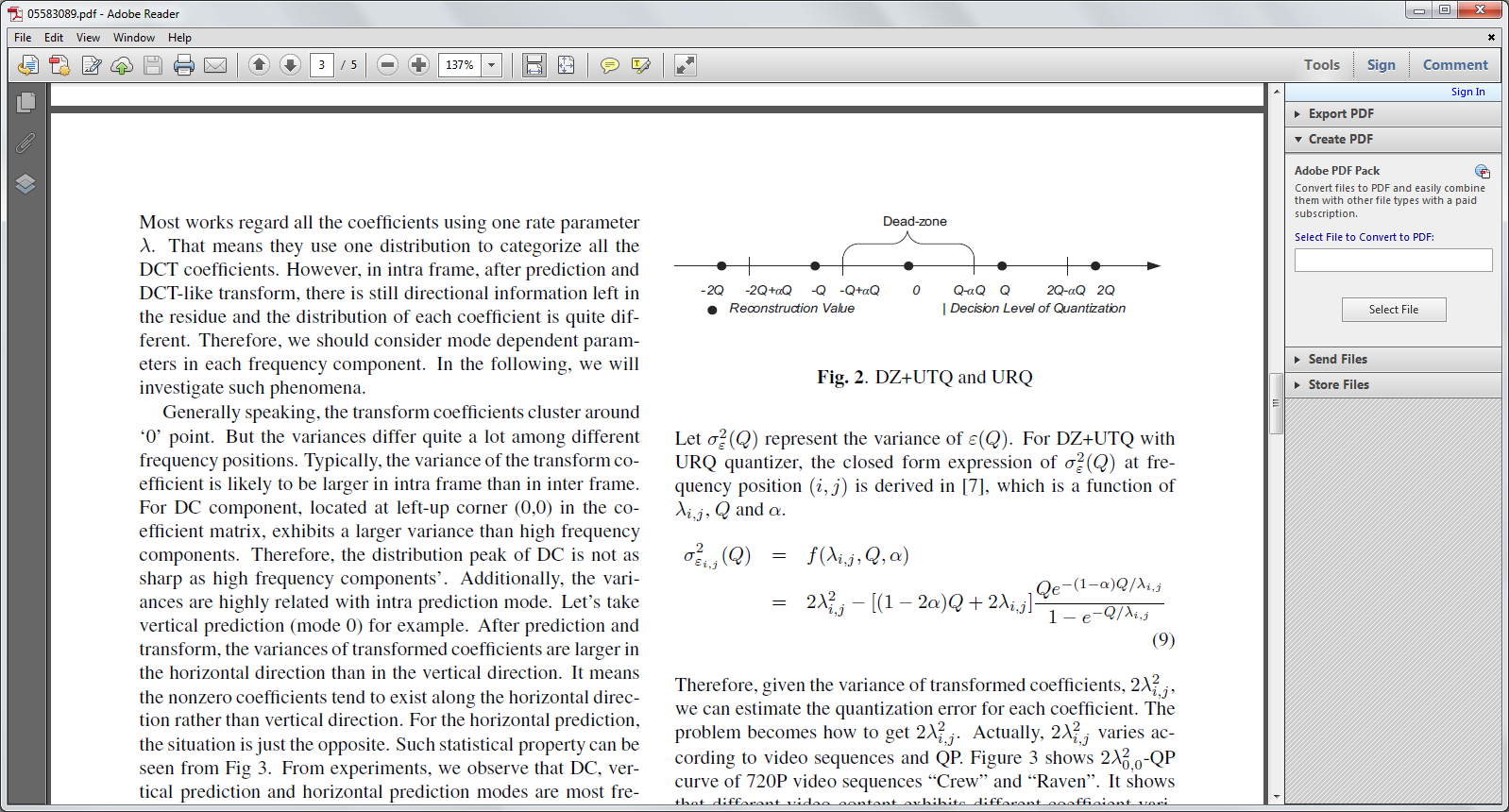


Figure 6‑10. Dead zone uniform quantizer with rounding offset.

### Cross-component prediction

Cross component prediction (CCP) predicts each chroma transform block by its co-located reconstructed luma transform block. CCP is available for 4:4:4 chroma format only and assumes a linear model for the correlation between the luma and chroma components. More precisely, let *rC*(*x*,*y*) be the residuals associated with the chroma component for pixel located at (*x*,*y*). Let also *r’L*(*x*,*y*) be the reconstructed residuals for the luma component at the same spatial location. . The CCP chroma residual, Δ*rC*(*x*,*y*), for the same spatial location is obtained as:



where *α* denotes the linear model parameter which is selected on a chroma transform block basis. By using the luma residuals, the implementation complexity is reduced and pipelining and parallel processing are facilitated as the prediction can be performed without waiting until reconstruction of luma residual signal. Ten different values for *α* are specified in the standard and the encoder selects the best value based on the following:







where cov(·) and var(·) denote the covariance and variance operation, respectively.

### Transform skip selection

The transform skip is allowed for large TUs by a RDO with the transform mode. The maximum size allowed is specified by TransformSkipLog2MaxSize. There is one fast mode decision specified by setting TransformSkipFast to 1, which skips the transform skip on luminance intra coding when the PartMode is not PART\_NxN and on chrominance intra coding when all corresponding luminance TUs are not coded in transform skip mode. Therefore, to enable the transform skip on large TUs when the PartMode is PART\_2Nx2N for intra coding, the TransformSkipFast option should be set to 0. Another fast method for transform skip is to disable the RDOQ when a TU chooses the transform skip mode, which is achieved by setting RDOQTS to 0.

### Sign data hiding

Sign data hiding (SDH) allows to omit the transmission of the sign of the last non zero coefficient (in reverse scan order) in each CG. SDH is enabled using the flag *sign\_data\_hiding\_enabled\_flag* in the PPS. In reverse scan order, the sign of the last non zero coefficient is omitted if the distance (in scan positions) between the first and last non zero coefficients is greater than or equal to 4. If this is the case the sign is instead encoded (or ‘hidden’) in the parity sum of the CG coefficients as follows:

* Even sum corresponds to plus ‘+’ and
* odd sum corresponds to minus ‘-’.

When encoding, if the parity sum of the CG coefficients matches the omitted sign (according to the aforementioned) convention, no additional processing is performed. Otherwise the encoder has to change the level of one nonzero coefficient in the CG so that the sign value inferred corresponds with the sign value omitted from the bitstream. The selection of which coefficient to adjust is outside the scope of the HEVC specification. The HM encoder selects the coefficient that results in minimizing the impact on the rate-distortion performance of the whole CG. This processing is performed in HM within the method xQuant of the object TComTrQuant (file TComTrQuant.cpp). In particular if RDOQ is not used, then the method signBitHidingHDQ is called to loop over all CGs and for each one select which coefficient needs to be changed (if any) to match the sign hiding convention. The selection is done by purely minimizing the reconstruction error which is passed to the method via the parameter deltaU. Conversely, if RDOQ is used, SDH is performed in the method xRateDistOptQuant. In this case the Lagrangian rate-distortion cost is minimized, with rate estimates from RDOQ reused. Note that these estimates include the cases of incrementing and decrementing a given coefficient relative to the value selected in the RDOQ process, and thus the estimates are also suitable for controlling the selection of coefficient to adjust for SDH.

## Inter-prediction residual quadtree derivation

For a CU, the RQT hierarchy is traversed.

Firstly, the alpha parameter for CCP for each chroma TB is estimated by computing the SSD between the two blocks and then quantising the result into legal alpha parameter values.

For alpha estimation, use either the residual between the input YUV file and the predicted image, or use the reconstructed residual (i.e. resulting from the inverse transformed and dequantized residual coefficients), according to ReconBasedCrossCPredictionEstimate.

Then, a search covering the cases of CCP enabled (only if available and nonzero alpha) and disabled, along with TS enabled (if available) and disabled is performed. Of the tested combinations of CCP and TS enablement, the best combination is selected. Note that CCP is tested using only the initial alpha parameter estimate.

## Quantization control

For an explanation of the usage of each control for quantization refer to the Software Manual [1]. Below are additional notes on the intention of these controls.

When encoding a sequence and targeting a particular rate, it is sometimes desirable not to use the rate control functionality of HM. Then, changing the quantization parameter directly is an alternative approach of influencing the rate of the coded sequence. Changing the quantization parameter once for a sequence provides a mechanism for rate targeting. In HM, the practice is to derive a frame-level quantization parameter (QP), used for encoding each picture, from a base QP. The base QP for pictures with POC up to *n* is set specified by the ‘-q’ command-line parameter and for the remaining pictures the frame-level QPs are calculated using base *qp + 1*. The picture at which this transition occurs is specified using the configuration option QPIncrementFrame.

The correspondence between lambda and QP may be changed as follows [9]:

To use a QP that corresponds to the lambda change for intra slices derived from the GOP size as in HM, configuration option IntraQPOffset is present that defines the QP for intra slices to be equal to the sum of the base QP and intraQPOffset.

A linear model to adjust the base QP (QPb) according to the hierarchy level of inter pictures is also present.

In summary, the procedure for deriving the frame-level QP is as follows [11]:

**Step1:** Get QPb

**Step2:** Calculate QP’1 = QPb + QPOffset3

**Step3:** Calculate QP’2 = QP’1 + QPOffset1

**Step4:** Calculate QP’3 = QP’2 + QPOffset2, where QPOffset2 = a × QP’2 + b

Where QPOffset3 is 1 if the POC of the current frame is greater than or equal to the POC at which the QP increment happens (i.e. the switching point) and QPOffset1 is derived by a linear model as described in [9].

When encoding HDR sequences, it is advantageous to adjust the frame-level QP locally based on luminance. Using the delta QP mechanism, an offset is applied at the CTU level, set according to the following configuration options: LumaLevelToDeltaQPMode, LumaLevelToDeltaQPMaxValWeight, LumaLevelToDeltaQPMappingDQP. Also, it is advantageous to adjust the chroma QP offset to improve the balance of bit allocation between luma and chroma components. See the section “HM encoding” of [10] for more details.

## Rate control

The HM encoder implements a single-pass rate control algorithm which provides a constant bit rate (CBR) mode. The algorithm is controlled using the RateControl configuration option and is based on a *R*-*λ* model, as presented in document JCTVC-K0103 [4]. This model assumes an exponential relationship between the coding rate and the Lagrangian multiplier *λ*:

, (11)

where *α1* and *β1* are constants dependent on the video source. The rate control process can be summarized into two main steps:

1. Bit allocation,
2. Quantization parameter derivation according to the used *R*-*λ* model.

Bit allocation is performed at the following levels of granularity: GOP, picture and CTU level. Sequence-level information is held in an instance of TEncRCSeq, GOP-level information is held in an instance of TEncRCGOP created for each GOP, and picture-level information is held in an instance of TEncRCPic. Initially, the rate controller computes the average bits per picture (*RPicAvg*) from the target bit rate (*Rtar*) and the video frame rate (*f*):

Then, bit allocation at the GOP level is performed using a smoothing window (SW) of 40 frames in length and taking into account the number of pictures already coded (*Ncoded*) and the bits spent on these pictures (*Rcoded*). For the current GOP the target bits (*TGOP*) are then given as:

where *NGOP* denotes the number of pictures in the current GOP. At the picture level bit allocation is performed taking into account *TGOP* and the bits spent over the pictures inside the current GOP and already coded (*CODEDGOP*). The number of bits allocated for each picture also considers a weight. The weight models the relative number of bits that should be allocated to one picture with respect to the others. As an example, in the random access coding configuration, when a hierarchical GOP structure is used, pictures belonging to higher levels of the hierarchy will receive a larger number of bits since they will serve as reference for subsequent coded pictures (i.e. at lower levels of the hierarchy). Conversely those located to the lower level of the hierarchy will receive fewer bits. The weights used in HM are selected differently depending on whether random access or low delay GOP type is used and depending on the bits per pixel (bpp) associated with the target bit rate (*Rtar*). Table 6‑4 and Table 6‑5 list the weights used for random access (8-picture GOP) and low delay GOP configurations, respectively. Random access with a 16-picture GOP is also supported. The target coding rate for current picture (*TCurrPic*) is then given as:

, (22)

From the allocated bit budget (*TCurrPic*) the rate controller subtracts the number of bits associated with the header information (e.g. slice header, SPS, PPS, etc.) which is estimated from the header bits spent in previously coded pictures at the same hierarchical level.

Table 6‑4. Values for *ωCurrPic* for random access GOP 8-picture configuration.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Coding**  **order** | **POC in the GOP structure** | **bpp > 0.2** | **0.2 ≥ bpp ≥ 0.1** | **0.1 ≥ bpp ≥ 0.05** | **Otherwise** |
| 1 | 8 | 15 | 20 | 25 | 30 |
| 2 | 4 | 5 | 6 | 7 | 8 |
| 3 | 2 | 4 | 4 | 4 | 4 |
| 4 | 1 | 1 | 1 | 1 | 1 |
| 5 | 3 | 1 | 1 | 1 | 1 |
| 6 | 6 | 4 | 4 | 4 | 4 |
| 7 | 5 | 1 | 1 | 1 | 1 |
| 8 | 7 | 1 | 1 | 1 | 1 |

Table 6‑5. Values for *ωCurrPic* for low delay GOP configuration.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Coding**  **order** | **POC in the GOP structure** | **bpp > 0.2** | **0.2 ≥ bpp ≥ 0.1** | **0.1 ≥ bpp ≥ 0.05** | **Otherwise** |
| 1 | 1 | 2 | 2 | 2 | 2 |
| 2 | 2 | 3 | 3 | 3 | 3 |
| 3 | 3 | 2 | 3 | 2 | 2 |
| 4 | 4 | 6 | 10 | 12 | 14 |

Finally, for each CTU the bit rate is allocated taking into account the number of bits allocated for the picture of which the CTU belongs to and the number of bits spent while encoding previous CTUs (*CODEDCTU*). As for the case of picture bit allocation, CTUs are weighted depending on their position within the picture. More precisely, the weight for the current CTU (*ωCurrCTU*) is set equal to the sum of absolute transform differences (SATD) for intra coded slices or to the number of bits estimated using the *R*-*λ* model in (1) for inter coded slices. Therefore the target bits allocated for the current CTU (*TCurrCTU*) is given by:

, (33)

where *SWCTU* = min(4, *non coded CTUs*) and denotes a smoothing window used to spread the allocated bits over the next *SWCTU* CTUs. As may be noted from the formula to compute *TCurrCTU*, the initial estimated bits for the current CTU are adjusted by the amount of bits overspent or underspent in the current picture up to the coding for this CTU.

Once the available bits have been allocated for the current CTU, the quantization parameter QP can be obtained according to the used *R*-*λ* model. More precisely, the Lagrangian multiplier is derived from *TCurrCTU* as follows:

where *N* denotes the number of pixels contained in the current CTU. Parameters *α* and *β* are initialized to average values of (3.2003 and -1.367, respectively) and then updated by least square regression once a coded picture is available, which considers the previous picture at the same temporal level. The updating strategy is based on the following RD model:

 (44)

With lambda being the slope of the line tangential to the RD curve:

 (55)

where  are obtained after encoding a picture and the two variables  are estimated with the above two equations. Then, the updated C and K variables are available for use as the new R-D parameters.

The parameter update considers only non-skipped CTUs of the picture, avoiding the very flat RD curve of skipped CTUs from influencing the parameter update [12]. Moreover, different values are used with respect to level of hierarchy of the GOP structure.

From *λ*, QP is finally obtained as:

(66)

where the ⎣⋅⎦ operator returns the largest integer smaller than or equal to the argument and ln(·) is the natural logarithm.

The HM rate control algorithm allows to manually set the initial QP value used for the first coded picture using option --InitialQP=*value*. If *value* is equal to zero, the algorithm automatically derives the value using the R-*λ* model otherwise HM uses the specified value. The remaining parameters associated with the rate control algorithm are listed in Table 6‑6 along with a brief description.

Table 6‑6. Configuration options for rate control algorithm.

|  |  |
| --- | --- |
| **Parameter name** | **Description** |
| TargetBitrate | Target bit rate for the whole sequence measured in bit per second (bps). |
| KeepHierarchicalBit | Determines how bit allocation is done across different pictures. Allowed values are: 0 = uniform, 1 fixed ratio according to the used GOP structure (i.e. random access or low delay) and 2 = adaptive with respect to the source content. |
| LCULevelRateControl | Switch between CTB-based or picture-based rate control. |
| RCLCUSeparateModel | Selects whether to use *α* and *β* parameters on a CTB or picture basis. |
| InitialQP | When specified this initial QP is used for the first (Intra) picture. If RCForceIntraQP is enabled, then this value is used for all pictures at temporal layer 0.  When this option has effect, λ is set as follows:  Where *GOPsize* is the size of the GOP in frames and *QPslice* is the value specified with the ‘InitialQP’ option. |
| RCForceIntraQP | Force QP value for intra-pictures to be equal to the value specified with InitialQP. |

The following subsections will describe the rate control workflow for bit allocation at different levels (e.g. GOP, picture and CTB) and update of the model parameters.

### Workflow for bit allocation and lambda estimation

#### Sequence-level operation

The function init for class TEncRateCtrl and the function create for class TEncRCSeq perform the sequence rate allocation. Accordingly, a parameter ‘*adaptiveBits*’ is defined. When KeepHierarchicalBit is equal to 2 and a GOP size of 4 is used in a low-delay configuration, *adaptiveBits* is set equal to 1. When KeepHierarchicalBit is equal to 2 and a GOP size of 8 is used in a non low-delay configuration, (i.e. random access), *adaptiveBits* is set equal to 2. Otherwise, *adaptiveBits* is set equal to 0.

A list of ratios *bitsRatio* is also set for the above two GOP structures. Initially the ratios of *bitsRatio* are set with the values listed in Table 6‑4 and Table 6‑5, then, depending on the value for parameter *adaptiveBits*, these values are updated on a GOP basis with the number of bits spent in the previous GOP.

#### GOP-level operation

The function create of class TEncRCGop performs the GOP-level rate allocation. Firstly, the target bits for the current GOP are estimated as follows:

where *picCount* denotes a smoothing window of up to 40 pictures. If the value of the parameter *adaptiveBits* (see Section 6.9.1.1) is greater than zero then the values for *bitsRatio* are adjusted as follows. Let *λL1* denote the Lagrangian multiplier for the picture at hierarchical level equal to 1 in the GOP structure and let *lambdaRatio*(*i*) for *i* = 0,…,*GOPsize* – 1 denote an array of Lagrangian multiplier ratios computed as listed in Table 6‑7. By inverting (1) to derive *λ* as a function of *R* it yields:

Given the computation of *lambdaRatio*(*i*) values, for each frame *i* in the GOP the associated rate *R*(*i*) can be obtained as:

where coefficients *A*(*i*) and *B*(*i*) are computed by the function xCalcEquaCoeff of class TEncGOP. Given the bit budget for the current GOP *TargetBitsGOP* and coefficients *A*(*i*) and *B*(*i*), the function xSolveEqua of class TEncGOP computes *λ\** by solving the following equation:

The equation is solved using the bisection method where the maximum number of iterations is set to 20. After *λ*\* is obtained, the function setAllBitRatio in class TEncGOP sets the weights *ωi* for all pictures in the GOP as:

where *PicSize* denotes the number of pixels in a picture. From the derived *ωi* and inter coded pictures, the rate allocation can be performed using formula (2) and subtracting the estimated rate for the headers. For intra-coded pictures, the rate (*TCurrPic*) is modified as follows: let *TotalCostIntra* be the picture SATD computed over a non-overlapping grid of 8×8 blocks where for each block the average value of the associated luminance pixels is subtracted to each sample prior to apply the Hadamard transform. Then let *αINTRA* and *βINTRA* be defined as:

The bits allocated for this intra picture (*TCurrPic*) are finally modified as:

Table 6‑7: Values for *lambdaRatio* used to derive the *bitsRatio* for a GOP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Random access GOP configuration | | Low delay GOP configuration | |
| *lambdaRatio(i)* | *λL1* < 90 | Otherwise | *λL1* < 120 | Otherwise |
| 0 | 1 | 1 | 1.3 *× lambdaRatio(1)* | 5 |
| 1 | 0.725·ln(*λL1*) + 0.7963 | 4 | 0.725 *×* ln(*λL1*) + 0.5793 | 4 |
| 2 | 1.3 *× lambdaRatio(1)* | 5 | 1.3 *× lambdaRatio(1)* | 5 |
| 3 | 3.25 *× lambdaRatio(1)* | 12.3 | 1 | 1 |
| 4 | 3.25 *× lambdaRatio(1)* | 12.3 | n.a. | n.a. |
| 5 | 1.3 *× lambdaRatio(1)* | 5 | n.a. | n.a. |
| 6 | 3.25 *× lambdaRatio(1)* | 12.3 | n.a. | n.a. |
| 7 | 3.25 *× lambdaRatio(1)* | 12.3 | n.a. | n.a. |

#### Picture-level operation

At picture level, two main operations take place:

* Estimation of the picture level Lagrangian multiplier *λ* and QP
* Bit allocation for each CTU with subsequent estimation of the associated Lagrangian multiplier *λCTU* and QP.

For the estimation of the picture level *λPic*, the function estimatePicLambda of class TEncRCPic performs this estimation differently for intra and inter coded slices. For inter coded slices, *λ* is given as:

Conversely, for intra-coded slices, the value *λ* of is given as:

The obtained *λPic* is then clipped using the values obtained in previously coded frames belonging to the same hierarchical level in the GOP structure. More precisely, let *λSameLevel* and *λLastPicture* denote the Lagrangian multiplier value for the last coded picture at the same hierachical level and the last coded picture, respectively. The current value of *λPic* is clipped by the following three ordered steps:

After the clipping, *λPic* is used to compute the quantity *ωCurrCTU* for each CTU. This quantity represents an estimate of the number of bits which will be allocated to that CTU using the R- *λ* model:

where *αCTU* and *βCTU* denote the *α* and *β* parameters derived over previously coded CTUs at the same position and hierarchical level of *CurrCTU*. After the computation of each *ωCurrCTU*, their value is normalized between [0, 1]. Finally the value of the picture level QP (*QPPIC*) is computed using (6) and clipped to avoid large quality fluctuations among frames.

At CTU level, the function getLCUTargetBpp of class TEncRCPic computes the target bits for each CTU (*TCurrCTU*). For inter coded slices, *TCurrCTU* is computed as specified in (3). For intra coded slices, *TCurrCTU* is computed as follows.

A variable *remainingCostIntra* is initialized to the same value of *TotalCostIntra* in Section 6.9.1.2. A CTU cost estimate *MAD* is derived by summing up the SATD values for all 8×8 blocks belonging to the current CTU. Also, for each CTU at position *idx* in raster scan order inside each picture, the variable *TargetBitsLeft*(*idx*) is defined as:

If *remainingCostIntra* > 0.1 the variable *weightedBitsLeft* is derived as follows:

Where *bitrateWindow* denotes a window to spread the bit budget across the next (up to ) 4 CTUs in raster scan order.Then, the target bits allocated for the current CTU (*TCurrCTU*) is given as:

If instead *remainingCostIntra* ≤ 0.1, then *TCurrCTU* is derived as follows:

The function getLCUTargetBpp returns *TCurrCTU* expressed as bits per pixel (bpp) which is computed as:

Where *CTUpixels* includes adjustment for reduces pixel count of CTUs along right edge and bottom row.

The function getLCUEstLambda in TEncRCPic computes the Lagrangian multiplier for the current CTU as:

,

Where, as above, *αCTU* and *βCTU* denote the *α* and *β* parameters derived over previously coded CTUs at the same spatial position and hierarchical level of *CurrCTU*. As for the picture level case, also at the CTU level is clipped to avoid large quality and coding mode selection swings between CTUs in the same neighbourhood. More precisely, a search for is performed by searching back from previous CTUs until a valid value (greater than zero) is found. If is found, is updated as follows:

If is available then then is constrained as follows:

Otherwise a generic constraint is applied:

The function getLCUEstQP of TEncRCPic computes the quantization parameter for the current CTU (*QPCTU*) using (6) and . The value obtained is then clipped as follows. Let *QPneighbour* be the QP value obtained by searching back from previously coded CTUs until a valid value (i.e., greater than zero) is found. Using also *QPPIC*, the value for *QPCTU* is derived by applying the following ordered steps:

### Workflow for parameters update

After one CTU or picture has finished being encoded, the state of the rate controller needs to be updated so that the model parameters can adjust their value to the characteristics of the video content. In particular, the following parameters are modified:

* Bits spent and frames encoded
* *αPic* and *βPic* parameters as well as the estimate for the bits spent in the slice header
* *αCTU* and *βCTU*

The following subsections will described how the aforementioned updates are computed.

#### Update of parameters after one picture is encoded

The function updateAfterPic of class TEncRCSeq updates the global counters which store the bits spent so far as well as the total coded frames. The same kind of update is also performed in the function updateAfterPicture of class TEncRCGOP. Conversely, the function updateAfterPicture of class TEncRCPic updates parameters *αPic* and *βPic* using the number of bits spent to encode the picture. More precisely, the variables *ln*bpp and *λdiff* are set as follows:

Where *RSpent* denotes the bits spent to encode the current picture in units of bits per pixels. If the current picture is intra coded then parameters *αPic* and *βPic* are updated as follows:

If instead the current picture is inter coded, the update operation firstly computes the Lagrangian multiplier *λcalc* from the actual bits spent (*RSpent*) still in units of bits per pixels:

If *λcalc* < 0.01 or *λPic* < 0.01 or *RSpent*/ < 0.001 then *αPic* and *βPic* are updated as follows:

where *δα* and *δβ* are the update steps for the related parameters whose value is set based on the initial bpp allocated to the picture (see function create in TEncRCSeq).

Otherwise if none of the above three conditions are true, then *αPic* and *βPic* are updated as:

After computation, *αPic* and *βPic* are clipped in the range [0.05, 500] and [-3.0, -0.1], respectively. More details about the derivation of the update formulae are provided in Appendix 1 of [6].

Along with the update of *αPic* and *βPic*, the function updateAfterPic stores the updated value in the memory structure m\_picPara of class TEncRCSeq. This structure contains *αPic* and *βPic* for all pictures at different hierarchical levels in the GOP. Finally, the object TEncRCPic associated with the current picture is then copied to the memory structure m\_listRCPictures which is C++ list modelling a circular buffer with maximum capacity equal to 32. The information stored in this buffer is then used to estimate the bits spent in coding the slice header and clip the values for *λPic* and *QPPic* (see Section 6.9.1.3).

Finally, if the hierarchical level of the picture being updated is one, the value for the lagrangian multiplier *λL1* (see Section 6.9.1.2) is updated using a weighting of 0.5 of the previous *λL1* value computed and a weighting of 0.5 of the *λcalc*.

#### Update of parameters after one CTU is encoded

The function updateAfterCTU in class TEncRCPic is called to update *αCTU* and *βCTU*. The update is performed as follows. Firstly, the bits per pixel *bpp* is computed:

where *RCTU* denotes the coding bits spent on the current CTU. As for the picture case, λcal is then derived using *bpp* and *αCTU* and *βCTU*.

If < 0.01 or < 0.01 or bpp < 0.0001 then *αCTU* and *βCTU* are adjusted, as follows:

Where parameters *δα* and *δβ* have the same value as in Section 6.9.2.1. Otherwise, the adjustment occurs as in the picture case described above. Once *αCTU* and *βCTU* are updated the function updateAfterCTU stores their values in the memory structure m\_LCUPara of class TEncRCSeq. This structure stores the parameters for all CTUs in a frame at different temporal hierarchy level so that the rate controller can update the statistics to the video content.

### Target bits saturation for rate control

When RCCpbSaturation is enabled, a ‘target bits saturation’ method is used to avoid that the coded picture buffer (CPB) overflows or underflows.

The hypothetical reference decoder (HRD) defined in Annex C of [2] specifies a coded picture buffer characterized by three parameters (*R*, *B*, *F*) where *R* denotes the transmission bit rate, *B* is the CPB size, and *F* is the CPB fullness. The model assumed by the HRD is the so-called leaky bucket model. Figure 6‑11 shows an example of CPB with the HRD parameters. Then, for frame *i*, *bi* is the amount of bits for the frame at time *ti*, with frame rate *f*.

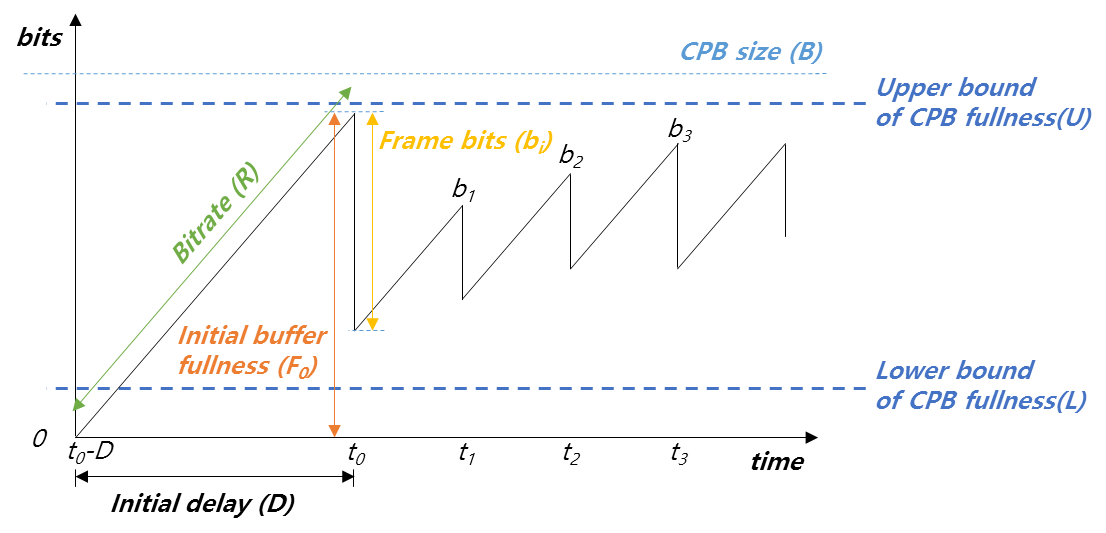


Figure 6‑11. Example CPB behavior.

As seen in Figure 6‑11, an upper bound and lower bound of CPB fullness are defined. The upper bound is set at 90% and the lower bound is set adaptively.

To prevent overflow, i.e. to make sure that the buffer fullness will stay below the upper bound level when the picture at time instant *i* is decoded, the target bits allocated for frame *i*, *b*(*i*), is adjusted as follows:

Let *Be*(*i*) be the estimated buffer fullness at time *i*, given as:

where *F*(*i*) denotes the actual buffer fullness at time *i*. If the following holds true:

then the target bit *b*(*i*), is adjusted as:

otherwise *b*(*i*), is left with the same value as allocated by the process described in Section 6.8.

To prevent underflow, i.e. to make sure that the buffer fullness will stay above the lower bound level when the picture at time instant *i* is decoded, *b*(*i*) is adjusted as follows. The lower bound L(*i*) is determined by:

where *TGOP* is the target bit for the current group of picture (GOP), *CODEDGOP* is coded bits spent for the current GOP, *wj* is weighted parameter for *j*-th picture, and *N* is number of pictures in GOP.

The lower bound check is defined by:

If the above relation holds, *b*(*i*), is set to *max*(*200*, *F*(*i*)-*L*(*i*)). It should be noted that for the underflow control the estimated buffer fullness is set to the actual buffer fullness. This is to guarantee that the buffer does not underflow at any time within the interval whereby picture *i* gets decoded. For more details on the CPB control refer to [7] and [8].

## Derivation process for slice-level coding parameters

### Sample Adaptive Offset (SAO) parameters

#### Search the SAO type with minimum rate-distortion cost

In the HM encoder, the following process is performed to determine the SAO parameters for each CTB in the slice or tile:

* Loop over the three colour components in a CTB, performing the following steps:
  + Collect the statistical information forall SAO type as follows
    - Set sao\_type\_idx = 0.
    - Classified pixels into categories according to sao\_type\_idx.
    - Calculate the sum of differences between original signal and reconstructed signal in each category.
      * Calculate number of pixels in each category.
      * Calculate offsets using step 1.1.2.1 and step 1.1.2.2.
      * Calculate RD-cost.
    - Set sao\_type\_idx = sao\_type\_idx+1; if sao\_type\_idx <= 5, run step 1.1.2; otherwise, end.
  + Determine the SAO parameters with lowest rate-distortion (RD) cost among the following three items.
    - If left CTB is available, calculate the RD cost by reusing the SAO parameters of left CTB.
    - If upper CTB is available, calculate the RD cost by reusing SAO parameters of upper CTB.
    - Five SAO types with minimum RD-cost in step 1.1.
  + Update pixels in DPB according to selected SAO type by adding offset.

#### Slice level on/off Control

A hierarchical coding of pictures is used for both low delay and random access configurations which allows the encoder to enable or disable SAO for picture with higher QP according to the percentage of CTBs to use SAO from the previous picture with lower QP. If previous picture with lower QP had more than 25% of CTBs using SAO type from 1-5, SAO will be enabled for the current picture, otherwise SAO will be disabled for the current picture.

### Adaptive QP selection

When this tool is used, the quantization parameter (QP) for each slice is changed based on the distribution of quantized coefficients in previous pictures. More specifically, for the current slice, the QP used is given as the one which minimizes the following cost measure:

,

where *q* denotes the quantization step associated to QP and *cl,i* denotes the *i*-th coefficient which is quantized to the level *l*. As stated above, the optimal QP derivation is computed using data from the previously coded picture to avoid two-pass encoding. The optimal quantization step *q* is then translated into the corresponding QP and set for the slice being encoded.

### Adaptive search range for motion estimation

The HM encoder provides an adaptive search range (ASR) algorithm which varies the extent of the region used to search for themotion vectors during motion estimation. The variation of the search range depends on the temporal distance between the current and the reference picture. This tool is enabled with the --ASR option and computes the new search range when the encoding of each slice starts.

More precisely, the search range for the current picture (*SRcurr*) is modified as follows:

,

where ⎣·⎦ denotes the floor operation, *SRori* denotes the original search range set from command line (under the common test conditions [3], adaptive search range is enabled, with a range from 96 to 384), *POCcurr* and *POCref* are the picture order count for the current and reference picture, respectively, gopSize is the size of the group of picture and *θ* is a rounding offset equal to gopSize/2. The value of *SRcurr* is then clipped in the range [*SRMIN*, *SRori*] where *SRMIN* is the minimum search range specified by command line option MinSearchWindow (default value 8).

### Weighted prediction control

Weighted prediction statistics are calculated once per picture, regardless of the decomposition of the picture into slices.

Firstly, weighted prediction parameters are selected according to the method described in section 6.10.4.1.

The following methods are provided for controlling the application of the selected weighted prediction parameters (in some cases the initial selection is overridden):

WP\_PER\_PICTURE\_WITH\_SIMPLE\_DC\_COMBINED\_COMPONENT =0,

WP\_PER\_PICTURE\_WITH\_SIMPLE\_DC\_PER\_COMPONENT =1,

WP\_PER\_PICTURE\_WITH\_HISTOGRAM\_AND\_PER\_COMPONENT =2,

WP\_PER\_PICTURE\_WITH\_HISTOGRAM\_AND\_PER\_COMPONENT\_AND\_CLIPPING =3,

WP\_PER\_PICTURE\_WITH\_HISTOGRAM\_AND\_PER\_COMPONENT\_AND\_CLIPPING\_AND\_EXTENSION=4

These methods are selected using WeightedPredMethod,-wpM and operate as described in sections 6.10.4.2.

#### Weighted prediction parameter selection

A search is performed iterating over each colour component of each reference picture present in each reference picture list. DC and AC values, RefPicDC, RefPicAC, OrigPicDC and OrigPicAC are obtained for the considered reference picture and the original picture, respectively. The weight is derived from the ratio OrigPicAC / RefPicAC. Then, the offset is derived from the difference OrigPicDC − RefPicDC.

#### Weighted prediction enablement

**Simple DC combined component method**

For each reference picture in each reference picture list, two values SADWP and SADnoWP are computed for the luma component between the original picture data and the considered reference picture, with the estimated weighted prediction parameters applied and no clipping in case such application results in values outside the range afforded by the sample bit depth. The SADWP is computed using the weighted prediction parameters associated with the considered reference picture and SADnoWP is computed using a default weight *iDenom*. If ratio SADWP / SADnoWP exceeds a fixed threshold then WP is disabled for all colour components of the considered reference picture.

**Simple DC per component method**

As with the simple DC combined component method, however now SADWP and SADPnoWP are calculated for each colour component. For the chroma components, if either component is determined to be using weighted prediction, then both chroma components will be configured to use weighted prediction (for a component that would otherwise be disabled, the default weight and zero offset are used).

**Histogram per component method**

In addition to the DC per component method, histograms Horig and Href are computed. Weights are searched in a range of [−10, 10] relative to the starting weight (inherited from the previous search for the considered reference picture). Then, for each colour component, offsets are search in a range of [-10, 10] relative to the starting offset, also inherited from the previous search for the considered reference picture. Href is scaled to produce Href\_scaled according to the candidate weight and offset and the distortion computed (as SAD between the Href\_scaled and Horig), with the candidate weight and offset offering minimum distortion being selected (this method can select new WP parameters, in addition to enabling the parameters from the earlier search).

**Histogram per component with clipping method**

As per the histogram per component method, however Href\_scaled is clipped such that sample values lie within the range afforded by the bit depth.

**Histogram per component with clipping and extension method**

As per the histogram per component with clipping method, but also search the case of using the default weight and an offset of zero.

## Pre-encoding GOP-based temporal filter

HM includes a temporal filter that is applied prior to the encoding operation, i.e., directly after reading input pictures, when the TemporalFilter configuration option is enabled. The following steps describe this process in more detail:

Step 1: Picture reading.

Step 2: Picture filtering.

The filter is applied only to pictures low in the coding hierarchy. In particular, for the random access configuration, pictures with POC % 8 == 0 are filtered and for the low delay configuration, pictures with POC % 4 == 0 are filtered. The filter is not applied in all intra configuration.

The overall filter strength, , is set according to the equation below for random access.

where is the number of pictures read.

For the low delay configuration, .

Step 3: The (up to two) pictures temporally adjacent to the current picture read. Normally two pictures are read, however for the first picture or close to the last picture, only the available pictures are read. Pictures subsequent to the current picture are only read when the TemporalFilterFutureReference configuration option is set. This option is typically disabled for low-delay configurations.

Step 4: Motion is estimated of the temporally adjacent pictures relative to the current picture per 8x8 sample block.

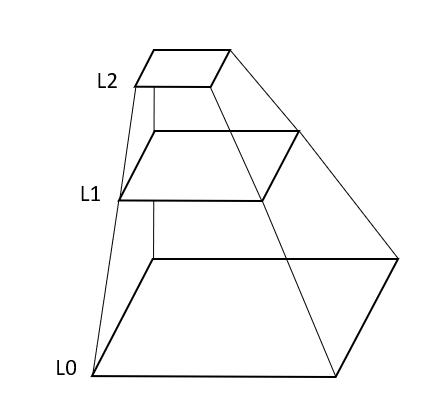


Figure 6‑12. The different layers of the hierarchical motion estimation. L0 is the original resolution. L1 is a subsampled version of L0. L2 is a subsampled version of L1.

A hierarchical motion estimation scheme is used with layers L0, L1 and L2, as illustrated in Figure 6‑12. Subsampled pictures are generated by averaging each 2x2 block for all read pictures and the current picture, i.e., L1 in Figure 6‑12. L2 is derived from L1 using the same subsampling method.

First, motion estimation is performed for each 16x16 block in L2. The squared difference is calculated for each selected motion vector and the motion vector corresponding to the smallest difference is selected. The selected motion vector is then used as initial value when estimating the motion in L1. Then the same is done for estimating motion in L0. As a final step, subpixel motion is estimated for each 8x8 block by using an interpolation filter on L0.

The following 6-tap interpolation filter is used[[1]](#footnote-2):

0: 0, 0, 64, 0, 0, 0

1: 1, -3, 64, 4, -2, 0

2: 1, -6, 62, 9, -3, 1

3: 2, -8, 60, 14, -5, 1

4: 2, -9, 57, 19, -7, 2

5: 3, -10, 53, 24, -8, 2

6: 3, -11, 50, 29, -9, 2

7: 3, -11, 44, 35, -10, 3

8: 1, -7, 38, 38, -7, 1

9: 3, -10, 35, 44, -11, 3

10: 2, -9, 29, 50, -11, 3

11: 2, -8, 24, 53, -10, 3

12: 2, -7, 19, 57, -9, 2

13: 1, -5, 14, 60, -8, 2

14: 1, -3, 9, 62, -6, 1

15: 0, -2, 4, 64, -3, 1

Step 5: Motion compensation is applied on the pictures before and after the current picture according to the best matching motion for each block. I.e., so that the sample coordinates of the original picture in each block have the best matching coordinates in the referenced pictures.

Step 6: The samples of the processed one by one for the luma and chroma channels as described in the following steps:

Step 6.1: The new sample value, , is calculated using the following formula.

Where is the sample value of the original sample, is the intensity of the corresponding sample of motion compensated picture and is the weight of motion compensated picture when the number of available motion compensated pictures is .

In the luma channel, the weights, , is defined as follows:

Where

For all other cases of , and :

For the chroma channels, the weights, , is defined as follows:

Where and

The strength of the temporal filter can be adjusted for different frames in each GOP by using one or moreTemporalFilterStrengthFrame\* configuration options, where ‘\*’ indicates a particular frame in the GOP for which the strength is being set.

Step 6.2: The filter is applied for the current sample and resulting sample value is stored to produce a filtered current picture.

Step 9: The filtered current picture is encoded.

## Encoder-only supplemental motion vector estimation for point cloud coding content

*Note: This section describes functionality anticipated for a future release of HM.*

HM includes supplementing motion estimation starting points using external information, to assist coding of point cloud content. The motion estimation is performed at the encoder side based on the supplementary information provided by a video-based point cloud compression (V-PCC) engine. Source pictures generated by the video-based point cloud compression engine can be accompanied with additional information that is generated during projection and atlas composition stage in the V-PCC system. Samples in the selected picture are allocated based on 3d co-ordinates and thus the 2d location can be inferred from the information that is present in coded point cloud bitstream.

This functionality is enabled with the UsePccMotionEstimation configuration option. When enabled, the following three files are provided to HM to provide motion estimation search hints:

* Occupancy map (specified using the OccupancyMapFile configuration option)
* Block-to-patch file (specified using the BlockToPatchFile configuration option)
* Patch info file (specified using the PatchInfoFile configuration option)

Pseudocode for this algorithm is as follows:

|  |
| --- |
| Algorithm. Find matched patches in reference frames |
| **while** frame *k* in GoP frames **do**  **for each** patch in patches list of reference frame *[refIdx]*  **for each** patch in patches list of frame *[k]*  estimate cost in RDO based on prediction  **if** (patchIdx[k] = patchIdx[refIdx])  do motion refinement  **end if**  **if** (dist < bestDist)  update pcMvFieldNeighbours  **end if**  **end for**  **end for**  **end while** |

The centre of the region of interest of a candidate inter-predicted PU is tested against the occupancy map at a pixel granularity. If occupied then a patch index is obtained from the block-to-patch file for the 16x16 block corresponding to the pixel. Then, a patch info object is obtained for the determined patch index for the current frame.

Then, match patches defined for the reference picture against the projection index of the current candidate block.

For matching patches, if the current patch 3D co-ordinates are within the range of the reference patch 3D co-ordinates then find the one with lowest difference in supplied distortion between the current patch vs the reference patch.

Set the startMv according to the best identified patch and invoke xTZSearchHelp(). The relationship between 2D motion vectors of the pictures vs 3D vectors of the patches is shown in Table 6‑8.

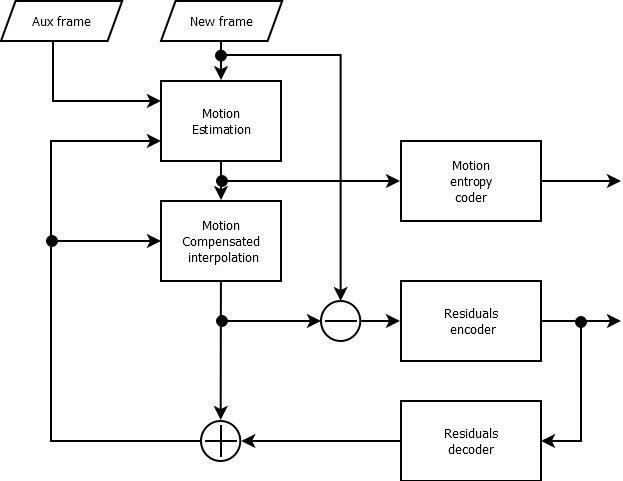
 ****

Table 6‑8. 3D motion vector derivation process based in 3D information

### Supplemental motion vector estimation file formats

Occupancy map

The occupancy map is a binary file mapping occupancy of each pixel (i.e. luma sample) for pictures of a test sequence. Occupancy of even-numbered POC pictures is present in the file. Occupancy for each pixel of a picture is stored using an ‘int’ in the occupancy file, as a raster scan of the picture. Each pixel is either occupied (zero occupancy value) or unoccupied (nonzero occupancy value).

Block-to-patch file

The block-to-patch file is a binary file mapping each 16x16 block of even-numbered POC pictures of the test sequence to a patch index, stored using ‘int’ data type. The file contains a raster scan list of patch indices for the 16x16 blocks of the applicable pictures.

Patch info file

The patch info file contains a list of patches for each even-numbered POC picture in the test sequence. The following structure is used for each applicable picture:

* Number of patches (size\_t) *n*. For each patch eight ‘ints’ are read, as follows:
  + Projection index (range 0-255)
  + 2D patch data: (u0, v0, sizeU0, and sizeV0)
  + 3D patch data: (d1, u1, and v1)

# References

1. F. Bossen, D. Flynn, K. Sharman, K. Sühring, “HM Software Manual”, docs/software-manual.pdf, included in the HM16.21 software release package.
2. ITU-T and ISO/IEC, “High efficiency video coding”, [Rec. ITU-T H.265](http://www.itu.int/rec/T-REC-H.265) | [ISO/IEC 23008-2](http://www.iso.org/iso/home/search.htm?qt=23008-2&published=on&active_tab=standards&sort_by=rel) (in force edition).
3. K. Sharman and K. Sühring, “Common Test Conditions for HM”, [JCTVC-AF1100](http://phenix.it-sudparis.eu/jct/doc_end_user/documents/32_Ljubljana/wg11/JCTVC-AF1100-v1.zip), April 2017.
4. B. Li, H. Li, L. Li and J. Zhang, “Rate control by R-lambda model for HEVC”, [JCTVC-K0103](http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=6473), October 2012.
5. R. Sjoberg, Y. Chen, A. Fujibayashi, M. M. Hannuksela, J. Samuelsson, T. K. Tan, Y.-K. Wang, and S. Wenger, “Overview of HEVC High-Level Syntax and Reference Picture Management”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1858–1870, December 2012.
6. B. Li, H. Li, L. Li and J. Zhang, “λ domain rate control algorithm for High Efficiency Video Coding”, *IEEE Transactions on Image Processing*, vol. 23, no. 9, pp. 296‒300, September 2014.
7. Y.-J. Ahn, X. Wu, W. Lim, D. Sim, “Target bits saturation to avoid CPB overflow and underflow under the constraint of HRD”, [JCTVC-U0132](http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=10136), June 2015.
8. Y.-J. Ahn, X. Wu, D. Sim, H. Ryu, “Improvement of coding efficiency for rate control under the constraint of HRD”, [JCTVC-V0078](http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=10272), October 2015.
9. K. Andersson, P. Wennersten, J.Samuelsson, J. Ström, P. Hermansson, M. Pettersson, ” AHG 3 Recommended settings for HM”, [JCTVC-X0038](http://phenix.int-evry.fr/jct/doc_end_user/documents/24_Geneva/wg11/JCTVC-X0038-v1.zip), May 2016.
10. E. François, J. Sole, J. Ström, P. Yin, “Common Test Conditions for HDR/WCG video coding experiments”, [JCTVC-Z1020](http://phenix.int-evry.fr/jct/doc_end_user/documents/26_Geneva/wg11/JCTVC-Z1020-v3.zip), January 2017.
11. P. Hanhart, Y. He, Y. Ye, X. Ma, H. Chen, H. Yang, M. Sychev, “On internal QP increase for bitrate matching”, [JCTVC-AB0043](http://phenix.int-evry.fr/jct/doc_end_user/documents/28_Torino/wg11/JCTVC-AB0043-v1.zip), July 2017.
12. Z. Liu, Y. Li, Z. Chen, X. Li, S. Liu, “Improvements for HEVC rate control”, [JCTVC-AH0024](http://phenix.it-sudparis.eu/jct/doc_end_user/documents/34_Marrakech/wg11/JCTVC-AH0024-v1.zip), January 2019.
13. P. Wennersten, J. Östrand, R. Sjöberg, “Encoder-only GOP-based temporal filter”, [JCTVC-AI0023](http://phenix.it-sudparis.eu/jct/doc_end_user/documents/35_Geneva/wg11/JCTVC-AI0023-v2.zip), March 2019.
14. V. Zakharchenko, J. Chen, “Encoder-only Supplemental Motion Vector Estimation for Point Cloud Coding content”, [JCTVC-AJ0028](http://phenix.int-evry.fr/jct/doc_end_user/current_document.php?id=10947), July 2019.
15. K. Sharman, A Tourapis, “On GOP-16 structures of the CTC”, [JCTVC-AK0030](http://phenix.int-evry.fr/jct/doc_end_user/documents/37_Geneva/wg11/JCTVC-AK0030-v1.zip), October 2019.

1. This interpolation filter is obtained from the Versatile Video Coding Test Model (VTM). [↑](#footnote-ref-2)