

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**ISO/IEC JTC 1/SC 29/WG 11 N18673**

**Document type: Approved WG 11 document**

**Title:**

**Status: Approved**

**Date of document: 2019-09-06**

**Source: 3DG**

**Expected action:**

**No. of pages: 62**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: mpeg.chiariglione.org**

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 N18673**

**July 2019, Gothenburg, SE**

|  |  |
| --- | --- |
| **Source** | **3DG** |
| **Title** | **G-PCC codec description v4** |

# G-PCC codec description v4

# Abstract

ISO/IEC MPEG (JTC 1/SC 29/WG 11) is studying the potential need for standardization of point cloud coding technology with a compression capability that significantly exceeds that of the current approaches and will target to create the standard. The group is working together on this exploration activity in a collaborative effort known as the 3 Dimensional Graphics Team (3DG) to evaluate compression technology designs proposed by their experts in this area.

This document provides a detailed description of the point cloud compression G-PCC (Geometry based Point Cloud Compression). It describes the coding features that are under coordinated test model (TMC13) study by 3DG as potential point cloud coding technology. G-PCC addresses the compression of point clouds in both Category 1 (static point clouds) and Category 3 (dynamically acquired point clouds).

Ed. Notes

v1:

* m42238: Neighbour-dependent entropy coding of occupancy patterns
* m42239: Inference of a mode using point location direct coding
* m42689: Sibling neighbour-dependent entropy coding
* m43591: Look ahead cube for efficient neighbours information retrieval
* m43592: Binarization of occupancy information
* m43600: Intra mode for geometry coding
* m43649: alternative entropy codecs
* m43665: Adaptive predictor selection for attributes coding
* m43780: Binarization of transform coefficients
* m43781: Efficient implementation of the Lifting Scheme
* m44750: A new binary entropy coder with update for geometry coding

v2:

* m44752: falsely occupied neighbours
* m44753: adjacent child neighbours
* m44899: a simplified version of the adaptive prediction scheme
* m44940: binary-tree based LoD generation
* m45811: An overview of OBUF and neighbour usage for geometry coding
* m45867: tile and slice partition
* m42538: recolouring

v3:

* m44486, m46209: fixed-point RAHT
* m46148: further reduction of neighbour configurations
* m46149: an improvement of advanced neighbours
* m46150: the reduction of states related to advanced neighbours in OBUF
* m47398: geometry slice header reduction in slice partition
* m46107: Reference structure modification on attribute predicting transform
* m44990: Fixed-point implementation of lifting and predicting transform
* m46188: regular sampling based LoD generation (remove binary LoD)
* m47399: slice based QP delta
* m47401, m47507: QP table
* m46108: attribute residual coding
* m45019: adaptive quantization scheme for RAHT

v4:

* m48892: slice partitioning order, limit the number of points per slice
* m49121: constrain slice partitions to trisoup node size
* m47834: add support for per-layer luma/chroma qp offsets
* m47827: Bypass coding of bypass bins
* m47352: Spatial scalability support
* m48918: LUT-based quantization
* m49407: Distance-weighted color transfer
* m47378: Upsampled transform domain prediction in RAHT

# Overview

Figure 1 provides an overview of the G-PCC encoder and decoder. The modules shown are logical, and do not necessarily correspond one-to-one to implemented code in the TMC13 software.

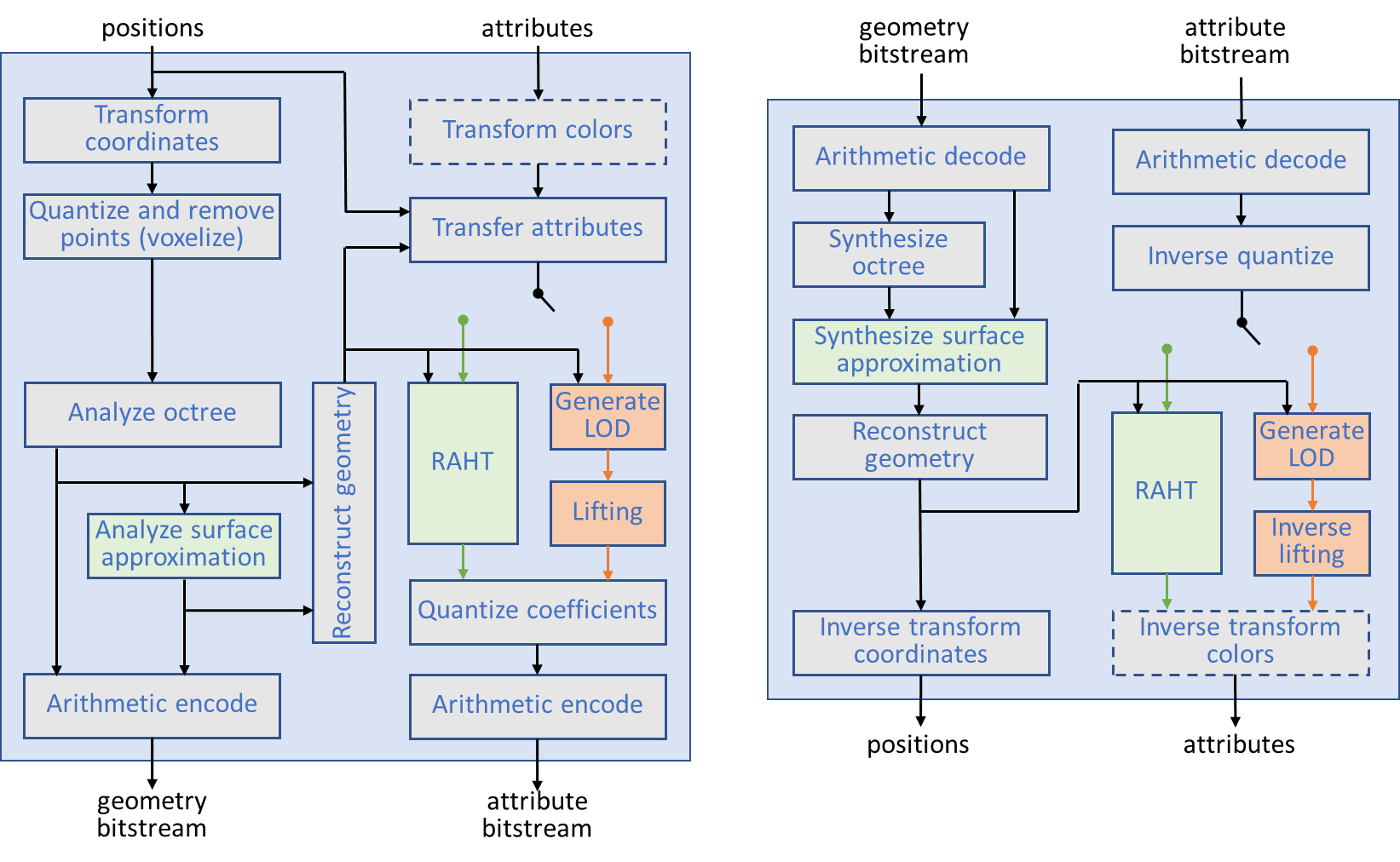


Figure : Overview of the G-PCC encoder (left) and decoder (right).

In both the encoder and decoder, point cloud positions are coded first. Attribute coding depends on the decoded geometry.

In Figure 1, the green modules are options typically used for Category 1 data. Orange modules are options typically used for Category 3 data. All the other modules are common between Categories 1 and 3.

For Category 3 data, the compressed geometry is typically represented as an octree from the root all the way down to a leaf level of individual voxels. For Category 1 data, the compressed geometry is typically represented by a pruned octree (i.e., an octree from the root down to a leaf level of blocks larger than voxels) plus a model that approximates the surface within each leaf of the pruned octree. In this way, both Category 1 and 3 data share the octree coding mechanism, while Category 1 data may in addition approximate the voxels within each leaf with a surface model. The surface model used is a triangulation comprising 1-10 triangles per block, resulting in a triangle soup. The Category 1 geometry codec is therefore known as the Trisoup geometry codec, while the Category 3 geometry codec is known as the Octree geometry codec.

There are 3 attribute coding methods in G-PCC: Region Adaptive Hierarchical Transform (RAHT) coding, interpolation-based hierarchical nearest-neighbour prediction (Predicting Transform), and interpolation-based hierarchical nearest-neighbour prediction with an update/lifting step (Lifting Transform). RAHT and Lifting are typically used for Category 1 data, while Predicting is typically used for Category 3 data. However, either method may be used for any data, and, just like with the geometry codecs in G-PCC, the user has the option to choose which of the 3 attribute codecs they would like to use.

In the remainder of this document, Section 3 describes the algorithmic details of the geometry and attribute coding methods in G-PCC. Relevant references for this document are provided in Section 4.

# Codec Descriptions

This section is organized as follows. Section 3.1 describes the pre- and post-processing of point clouds that may be common to Category 1 and Category 3 data in G-PCC. Section 3.2 describes the details of the Octree method for geometry encoding/decoding, while Section 3.3 describes the Trisoup geometry encoding/decoding. Section 3.4 describes the entropy coding method for geometry. Section 3.5 details the attributes transfer (recolouring) module that is used to transfer attributes to point cloud geometry that has been compressed and then reconstructed (decompressed) at the encoder, prior to attribute encoding. Section 3.6 then describes the Predicting method for attribute coding, Section 3.7 describes the Lifting method, and Section 3.8 describes the Region Adaptive Hierarchical Transform (RAHT). Section 3.10 describes the entropy coding for attribute and Section 3.11 describes tools for functionality.

## Pre- and post-processing

A point cloud is a collection of points with positions , , where is the number of points in the point cloud, and optional attributes , , where is the number of attributes for each point. The geometry of the point cloud comprises the point positions only. The attributes of the point cloud comprise the point attributes only. In this Test Model, the only attributes supported are a colour triple and/or a reflectance (or else no attributes). The geometry and the attributes of a point cloud are often expressed in application-specific spaces. The Test Model provides pre-processing and post-processing to convert between these application-specific spaces and finite-resolution internal spaces, where the point clouds are compressed.

### Coordinate transform and inverse

Original application-specific point positions are generally represented by floating point numbers and need not have any structure, lying in an original (or world) coordinate system denoted , .

Internal (or frame) coordinates , are obtained from original coordinates by the coordinate transformation

where . The parameters and are such that the point positions lie in a bounding cube for some non-negative integer parameter .

Point positions in the internal coordinate system that have been compressed and decompressed are denoted , where is the number of points in the decoded point cloud. may not be the same as .

Decoded point positions in the original coordinate system are obtained from decoded point positions in the internal coordinate system by the coordinate transformation

This can alternatively be expressed by the following homogeneous transformation from internal to original coordinates:

If the Trisoup geometry codec is used, is specified by the triSoupIntToOrigScale parameter, while is [0, 0, 0], and is specified by the triSoupDepth parameter. Components of points outside the bounding cube are clipped to the range if necessary.

If the Octree geometry codec is used, is specified by the positionQuantizationScale parameter, while is determined by and is determined by

such that is the smallest bounding cube with side an integer power of two that contains the point positions in internal coordinates. See Appendix A for definitions of Ceil, Log2, and Max.

### Colour transform and inverse

Attribute quantization of colour components is agnostic to the colour space of the components, since the components are processed independently. However, the TM supports conversion from RGB to YCbCr and back again (ITU Rec.709), if desired.

### Point quantization and duplicate point removal (voxelization)

Point positions are represented internally as non-negative -bit integers before being compressed. To obtain these integers, the point positions in the internal coordinate system are rounded. Let be a point position in the internal coordinate system. Then its representation as a non-negative -bit integer is

where is the function that rounds the components of a vector to the nearest integer. See Appendix A for details.

After such quantization, there may be multiple points with the same position, called duplicate points. The duplicate points removal process is optional. If enabled, it removes points with the same quantized coordinates. In order to detect duplicates, the STL set data structure is leveraged.

Multiple points with the same quantized position and different attributes will be merged in a single point. The attributes associated with the single point will be computed by the attributes transfer module described in Section 3.5

The process of position quantization, duplicate point removal, and assignment of attributes to the remaining points is called voxelization. In other words, voxelization is the process of grouping points together into voxels. The set of voxels are the unit cubes for integer values of , , and between 0 and . Specifically, the locations of all the points within a voxel are quantized to the voxel centre, and the attributes of all the points within the voxel are combined (e.g., averaged) and assigned to the voxel. A voxel is said to be occupied if it contains any point of the point cloud.

## Octree geometry encoding/decoding

If the Octree geometry codec is used, then the geometry encoding proceeds as follows. First, a cubical axis-aligned bounding box B is defined by the two extreme points and .

An octree structure is then built by recursively subdividing B. At each stage, a cube is subdivided into 8 sub-cubes. An 8-bit code, named an occupancy code, is then generated by associating a 1-bit value with each sub-cube in order to indicate whether it contains points (i.e., full and has value 1) or not (i.e., empty and has value 0). Only full sub-cubes with a size greater than 1 (i.e., non-voxels) are further subdivided. Since points may be duplicated, multiple points may be mapped to the same sub-cube of size 1 (i.e., the same voxel). In order to handle such a situation, the number of points for each sub-cube of dimension 1 is also arithmetically encoded. The same arithmetic encoder is used to encode all the information put into the bitstream. Currently the implementation from [1] is used.

The decoding process starts by reading from the bitstream the dimensions of the bounding box B. The same octree structure is then built by subdividing B according to the occupancy codes. Each time a sub-cube of dimension 1 is reached, the number of points for that sub-cube is arithmetically decoded and points located at the origin of the sub-cube are generated.

Note: In order to guarantee encoder/decoder synchronization, the point order defined by the decoding process is used during the level of detail generation process.

### Direct coding mode (DCM) [7]

The octree representation, or more generally any tree representation, is efficient at representing points with a spatial correlation because trees tend to factorize the higher order bits of the point coordinates. For an octree, each level of depth refines the coordinates of points within a sub-volume by one bit for each component at a cost of eight bits per refinement. Further compression is obtained by entropy coding the split information, i.e. pattern, associated with each tree node. This further compression is possible because the pattern distribution is not uniform, non-uniformity being another consequence of the correlation.

On the other hand, isolated points P cannot be better coded than directly coding their coordinates without compression, simply because by definition, there are no other points within the volume to correlate with. To do otherwise risks a worst-case penalty of five (=8-3) bits per refinement without taking into account entropy coding or assuming uniform distribution. Directly coding point coordinates in a volume/sub-volume is called Direct Coding Mode (or DCM hereafter).

Furthermore, isolated points “pollute” the distribution of patterns, inducing many patterns with only one occupied child, thus changing the balance of the distribution and penalizing the coding of other patterns.

It would be highly beneficial to get rid of isolated points in octree/tree coding in order to obtain better compression performance in volumes where point correlation exists. Also, complexity would be greatly reduced because a DCM is by essence much simpler than a recursive split of a tree.

A combination of octree coding and the Direct Coding Mode, see Figure 2: combining tree coding and Direct Coding Mode, is a straightforward attempt.

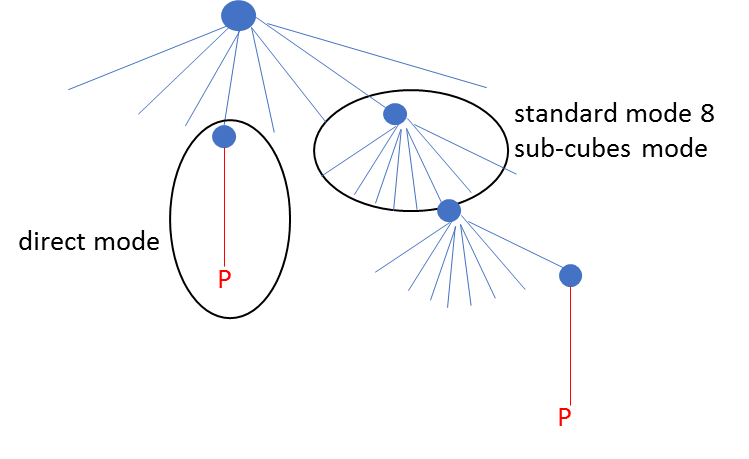


Figure 2: combining tree coding and Direct Coding Mode

Instead of signaling the usage of the DCM for all nodes of the tree is inferred from information coming from the node neighbourhood, leading to what we call Inferred Direct Coding Mode (or IDCM), see Figure 3.

A new eligibility (for DCM) condition is introduced and depends on information coming from the parent node itself or the neighbours of the parent node; this is the inference. If the node is not eligible, then tree coding is applied. If the node is eligible, then:

1. a binary flag is coded to signal if the DCM is applied (flag=1) or not (flag=0) to the node
2. if the flag is equal to 1, then points belonging to the associated volume are directly coded using the DCM. Otherwise (the flag is equal to 0), the tree coding process continues for the current node.

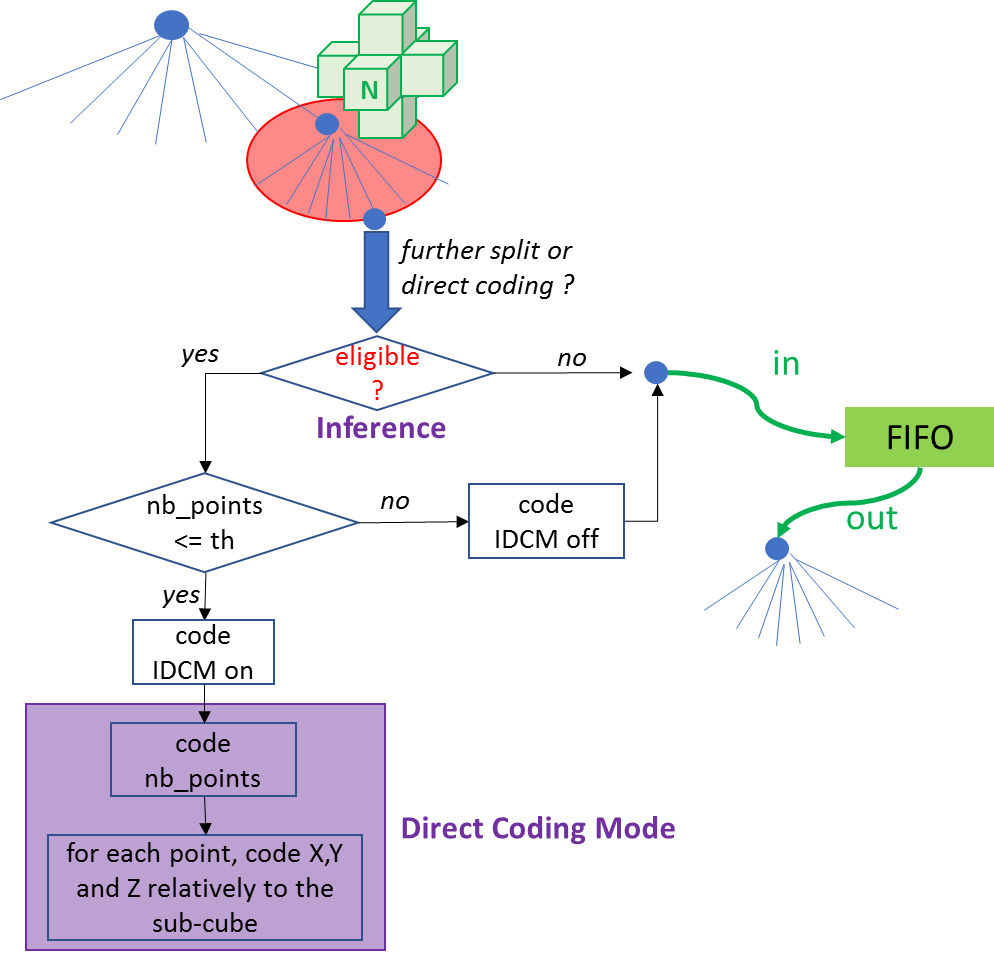


Figure 3: overview of IDCM

If the node is eligible for DCM, then a flag is coded to signal if the DCM is applied or not. This flag may be determined by an encoder based upon the number of points belonging to the volume attached to the node. If this number is less than or equal to a threshold th, then DCM is activated; otherwise it is not. TMC13 software has used the value th=2, i.e. up to two points can be directly coded in a volume. The value th is implicit, but could be a coded parameter at sequence/picture level.

If a DCM is applied, the coding of points is performed as follows

1. the number of points (necessarily at most th points) is coded using a truncated unary binarizer followed by a binary entropy coder. With th=2, there is only on flag signalling if the number of points is either 1 or 2. This flag is entropy coded using a binary arithmetic coder with a dedicated context
2. positions X, Y and Z are coded independently for each point, and relatively to the volume associated with the node. For example, if the volume is a cube of size 2^D, then D bits are needed for each coordinate of each point. These bits are direct pushed into the bitstream (bypass coding).

The criterion for eligibility can take two flavours

1. parent-based-eligibility. There is only one occupied child (=the current node) at parent-node level, AND the grand-parent node has at most two occupied children (= the parent node + possibly one other node).
2. 6N eligibility. There is only one occupied child (=the current node) at parent-node level, AND there is no occupied neighbour N (among the six neighbours sharing a face with the current cube associated with the current node, see Figure 3: overview of the proposed IDCM).

If eligibility condition is not fulfilled, the node is not eligible and the process continues to octree coding as in TMC13.

Concerning 6N eligibility, he breadth first scan order of the octree as performed in TMC13 ensures that the six neighbours N are available when determining the eligibility of the current node.

### Neighbour-Dependent Entropy Context (NEIGHB) [8]

Neighbour-Dependent Entropy Context (NEIGHB) selects the configuration depending on the six neighbours N of the parent node and these 6 neighbours, a neighbour configuration number (NC) is deduced to code occupancy pattern. This number NC is an integer between 0 and 63. The value 0 means that there is no occupied neighbour, and the value 63 means that all neighbours are occupied.

The decision process is to choose directly a distribution, among 64 distributions, from the neighbouring configuration number NC, but with a special handling of the case NC=0 which is further split into two sub-cases:

* If the parent node has only one occupied child node (Number of Occupied child nodes =: NO =1), then the position is directly coded by using 3 bits to code the occupied child node position in XYZ inside the volume associated with the parent node
* Otherwise the 0-th distribution corresponding to NC=0 is used.

In order for the decoder to know if NO=1 or not, an additional flag stating whether or not NO=1 must be coded when NC=0. This flag is also entropy coded using a binary arithmetic coder with a dedicated context.

#### Configuration and geometrical invariance

By construction of the octree, a current cube (in blue on Figure 4) associated with a current node is surrounded by six cubes of the same depth sharing a face with it. As depicted on the figure, weights (1, 2, 4, etc.) are associated with each of the six cubes and a neighbouring configuration NC is determined by summing the weights of occupied cubes among the six cubes. Figure 4 on the right, depicts the example for NC=15.

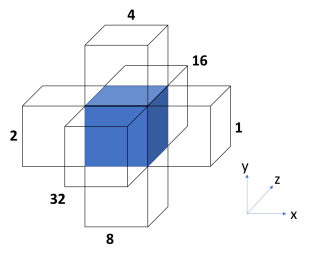
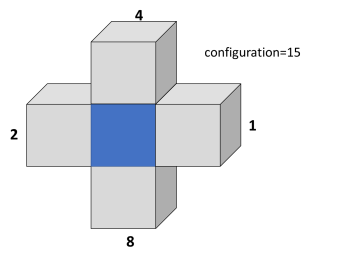
 

Figure 4: neighbour configuration NC (left) and example for NC=15 (right)

Using a breadth-first scanning order ensures that the occupancy of the six cubes neighbouring the current cube is known before (de)coding the 8-bit occupancy pattern of the current node. Therefore, the set Ɗj={b0...bj-1,NC} of states can be used in OBUF (Optimal Binarization with Update On-the-fly, or OBUF) to code the occupancy bit bj. For example, the size of Ɗ7 is 128\*64= 8192 states. This is marginally practical for HW implementation and, more importantly, this leads to the dilution of occupancy statistics into too many states to obtain optimal compression performance.

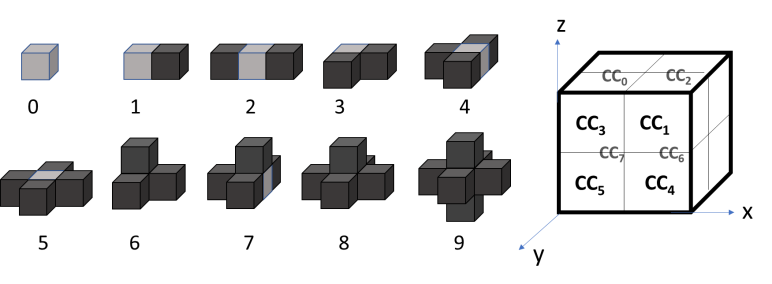


Figure 5: the ten invariant neighbour configurations NC10 (left) and

the scanning order of current child cubes Ci (right)

To solve these issues, the 64 neighbouring configurations NC are reduced to 10 invariant configurations NC10 by using geometry invariance. Assuming local geometry correlation of the point cloud as invariant under 3D isometries (for example by assuming anisotropy of the 3D space), then the neighbouring configuration can be transformed using 90° rotations and symmetries to match uniquely one of the ten configurations shown on Figure 5. The integer NC in [0,63] is thus mapped onto NC10 in [0,9].

As consistency between neighbouring configuration and occupancy pattern must be preserved, the pattern undergoes the same geometrical transform. Next, in the transformed space for both configuration and pattern, the scanning of the child cubes of the current cube is performed in the order shown in Figure 5 (right).

#### Configuration-driven OBUF State reduction

The size of the sets Ɗj of states can be lowered further by using a state reduction process. Using NC10 instead of NC, the size of Ɗ7 has become 128\*10=1280 states. However, the future introduction of additional intra (but also potentially inter) prediction tools will unavoidably increase its cardinality. Therefore, the number of states must be further reduced already at this stage in order to anticipate new tools. Further reduction is obtained using anisotropy and screening.

When the neighbourhood is empty (NC10=0), the points belonging to the current cube are isolated. In this case, one can use anisotropy, i.e. there is no privileged direction for occupancy. Consequently, when coding bj, the order of the preceding bits b0 to bj-1 is of no importance. Because we are dealing with binary data, the non-ordered set {b0,…,bj-1} is totally characterized by the sum b0+…+bj-1. Consequently, as shown on Figure 6 on the left branch of the decision tree, the set of states is reduced from {b0...bj-1, NC=0}to {b0+…+bj-1, NC10=0}when the neighbouring NC10 configuration is zero.

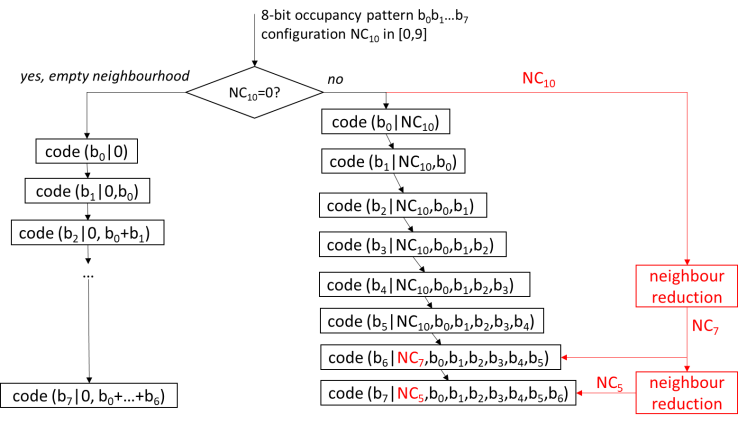


Figure 6: reduced dependency state (bj, N10) driving the entropy coder OBUF

When the neighbourhood is not empty, a state reduction is possible by using screening, a term used in physics when mobile charge carriers damp an electric field, like electrons around a nucleus. To continue with this analogy, we will simply replace the nucleus by a neighbouring volume and electrons by already coded current child volumes as depicted on Figure 7. Occupancy bits of child volumes CC0 to CC3 (blue small cubes) have already been coded, and the occupancy of these child volumes “screens” the occupancy of the neighbouring volume (green cube) located above the current volume. Therefore, when coding the occupancy of the child volume CC4 (red small cube), one can neglect the effect of this neighbouring volume, thus leading to the reduction of configurations NC10 = 6, 7 and 8 to configurations 3, 4, and 5 respectively. Practically, one replaces NC10 by NC7 that takes only seven different values.

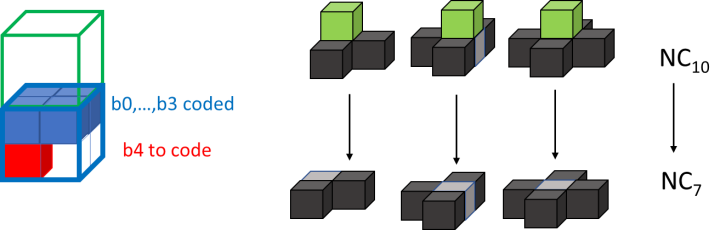


Figure 7: application of screening to reduce the number of configurations from 10 to 7

Figure 8 shows another usage of the screening technique for the last child cube CC7 for which the front and right neighbouring volumes are screened by the seven already coded child cubes CC0 to CC6. Consequently, for coding b7, NC7 can be replaced by NC5 that takes only five different values. Tests have shown that a good trade-off between compression performance and size of the coder mappings is to use NC10 for bits b0 to b5, NC7 for bit b6 and NC5 for bit b7, as shown on Figure 5.

At this stage the sizes of the eight sets Ɗj of states, for j=1,…,7, are 10, 20, 39, 76, 149, 294, 391 and 520 respectively. These are reasonable sizes that can be implemented.

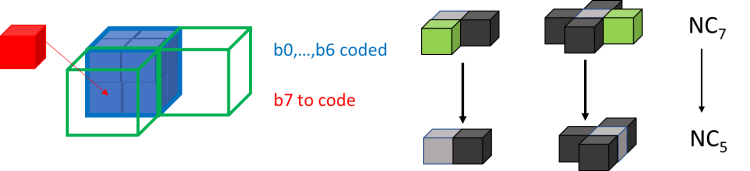


Figure 8: further application of screening to reduce the number of configurations from 7 to 5

#### On further reduction of neighbour configurations (nine neighbour configurations) [27]

Firstly, a minor change in the scan order of the child nodes of the current node must be introduced by swapping bits b5 and b6. This swap is performed to obtain a better screening of b6 from the right.

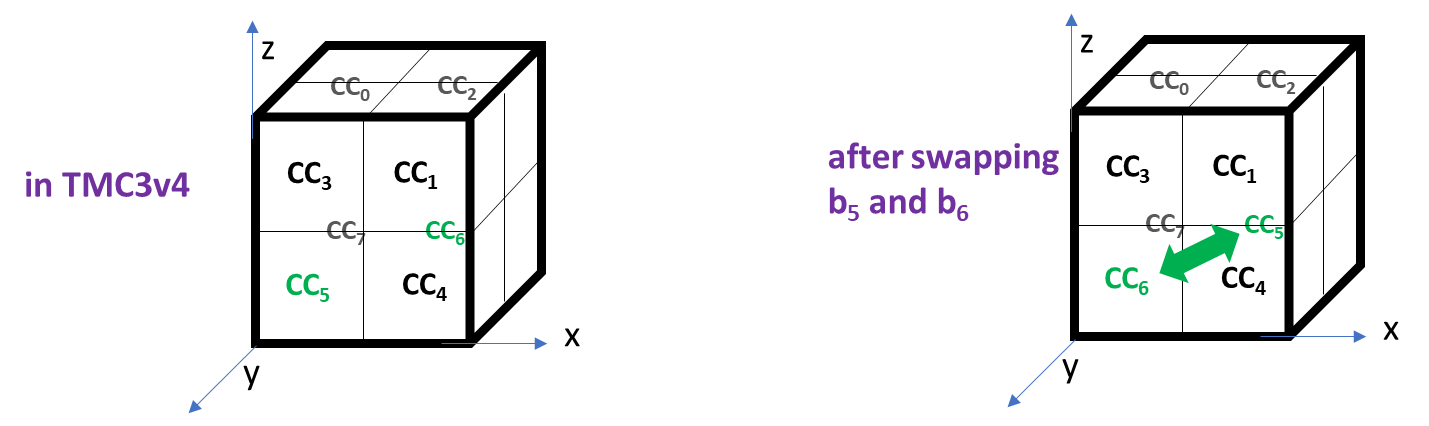


Figure 9: swapping b5 and b6

Secondly, NC10 is reduced to nine neighbour configurations by regrouping NC10=1 and NC10=2 together. Also, a more aggressive reduction is performed down to five configurations for bits b4 and b5, to three configurations for bit b6 and to two configurations for bit b7 as explicated in the figure below.

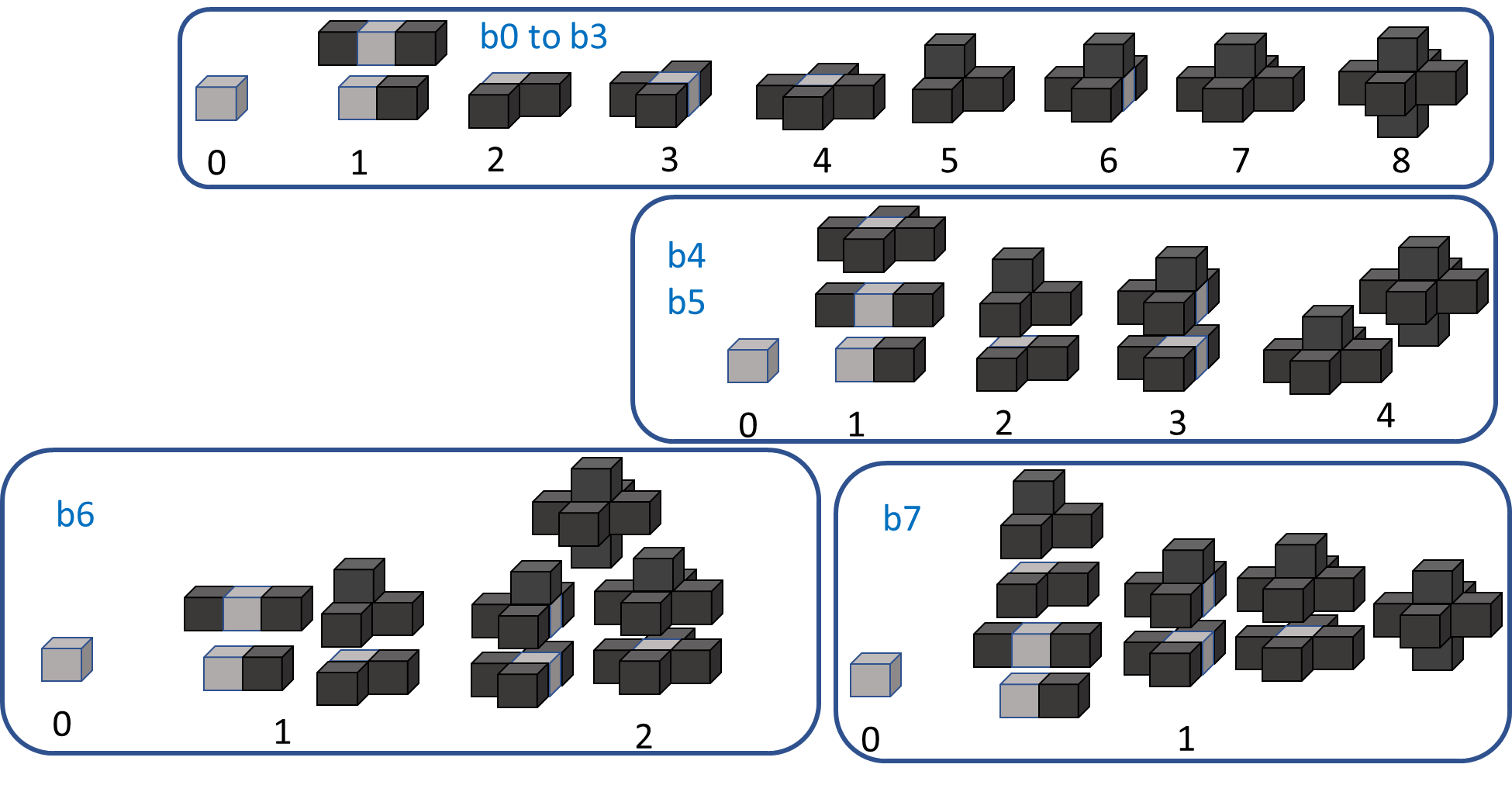


Figure 10: nine neighbour configurations with more aggressive reduction from bit b4

This leads to a significant decrease of the number of states, see table below. The max number of states associated with an occupancy bit has decreased from 520 to 136; the total number of states for all bits has decreased from 1499 to 604.

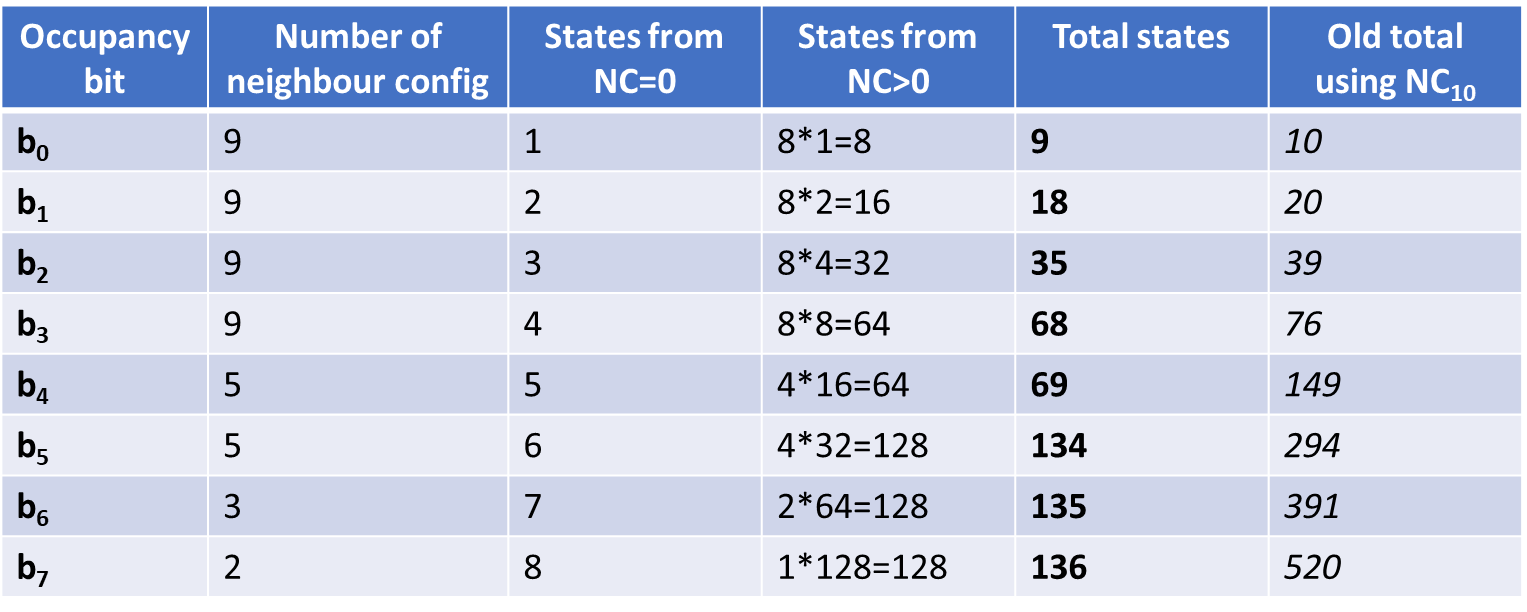


Figure 11: number of states using the proposed nine neighbour configurations

#### On using child nodes of already-coded neighbouring nodes [20][21]

Among the six neighbours sharing a face with a current node, some of them are already coded. Consequently, if they are occupied, their occupancy information is already coded in the bitstream and the occupancy of their child nodes is known by the decoder when processing the decoding of the current node. Therefore, the knowledge of the occupancy of the occupied already-coded neighbours’ child nodes can be used to better code the occupancy information of the current node.

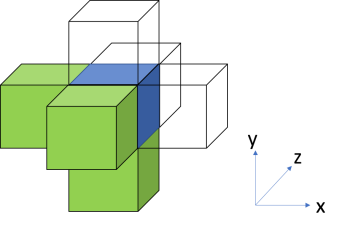
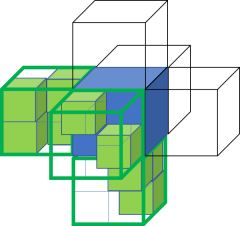
 

Figure 12: already coded neighbours for a breadth-first scan (left) and children of already-coded occupied neighbours (right)

For example, as shown on Figure 12, when nodes are scanned in breadth-first octree scanning order, in increasing order along the three XYZ axis, there are systematically three neighbouring nodes, sharing a face with the current node, that are already coded. These three nodes are those with lower X, Y and Z coordinates than the current node (Figure 12, left). The child nodes (see Figure 12, right) of the occupied nodes among these three nodes will be used to

* improve the determination of the neighbouring configuration NC10, and
* augment the set Ɗj of states used by OBUF to code the occupancy bits bj of the current node

##### Falsely occupied neighbour

The determination of the neighbouring configuration NC or NC10 as described in section 3.2.2.1 is modified as depicted on Figure 13. Let us consider an occupied already-coded neighbour (red cube on the left of the current node in blue). Instead of systematically taking the neighbour as occupied in the computation of the neighbouring configuration, the occupancy status of this neighbour is determined depending on its child nodes distribution.

If at least one child node is immediately adjacent to the current node, i.e. one of the child’s faces is shared with a face of the current node, then the neighbour is determined as “truly occupied”, thus not changing its status. However, if no child node of the neighbour touches the current node, the neighbour is said to be “falsely occupied”, and its status is set to “non-occupied” in the computation of the neighbouring configuration.

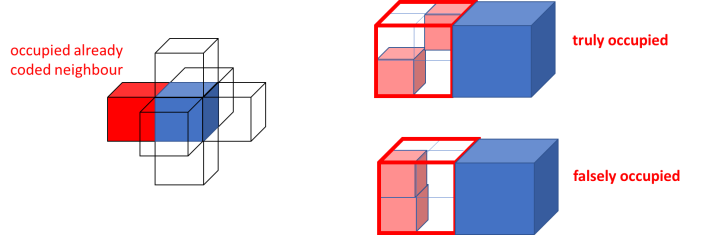


Figure 13: truly and falsely occupied neighbours

##### Occupied child nodes adjacent to a current sub-node

For a given sub-node of a current node, let NT (Number Touching) be the number of all occupied child nodes touching (or directly adjacent to) the sub-node from all already-coded occupied neighbouring nodes of the node. The number NT is computed, before the geometrical transform that reduces the neighbour configurations from 64 to 10. Figure 14 illustrates all possible configurations for a sub-node position adjacent to three already coded neighbour nodes. Depending on the distribution of the touching neighbour child nodes, the NT value is comprised between 0 and 3. As shown in Figure 15, the maximum value for NT is 1 as the sub-node touches only one already-coded neighbour node. One also understands that the top-right-rear sub-node of any node necessarily has NT=0 as this sub-node does not touch any already-coded neighbour.

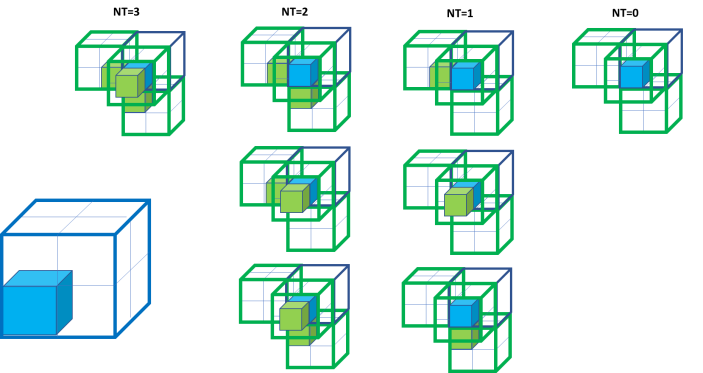


Figure 14: example of values of NT depending on neighbour’s child nodes distribution

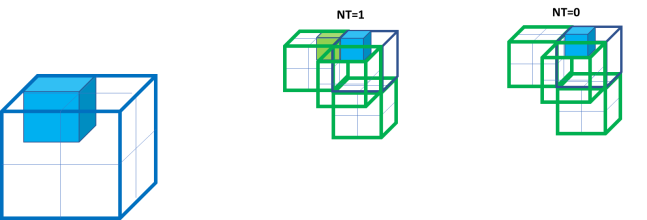


Figure 15: another example of values of NT depending on neighbour’s child nodes distribution

After applying the geometrical transform to reduce the neighbouring configuration, each of the child node CCj (as depicted in Figure 5) inherits a value NT[j] indicating the number of neighbour’s occupied child nodes touching the child node. This value can be used to augment the sets Ɗj of states. It has been observed that the case NT[j]=3 is marginal and does not provide extra information to OBUF compared to NT[j]=2. Therefore, in order to minimize the size of the set of states, the value NT[j] is capped to the value 2 to obtain the new value C[j]. The set of states is then augmented as follows

The sets Ɗj of states are practically small enough as reductions based on anisotropy and screening do still apply. By construction, not all combinations of the neighbouring NC10 configuration and C[j] are possible. For example, when NC10=0 (no occupied neighbours), C[j] value is always zero. In another example, when NC10=1, then C[j] is at most one, and necessarily zero for the four child nodes CCj (j=0,3,5,7) on the left on the Figure 5. All those natural reductions lead to sets of states not bigger than a thousand states in average.

##### On an improvement of advanced neighbours[28]

This method replaces the count C[i] of occupied child nodes of occupied neighbours that are adjacent to the current child node to code (occupancy bit bi) by the following information (see figure below)

* the number of occupied child nodes of occupied neighbours, i.e. essentially the same as C[i]
* the number of missed (=non-occupied) child nodes of occupied neighbours

that are adjacent to the current child node.

Then two quantities are deduced

* C[i] = min(2,occupied)
* M[i] = min(1,missed)

then combined into CM[i] = M[i] + 2\*C[i] that can take six valued in [0,5]. A reduction is performed, for i>4, as follows

* Nadv[i] = CM[i] if i<=4
* Nadv[i] = LUT\_red[CM[i]] if i>4 where LUT\_red = {0,0,1,2,3,3}.

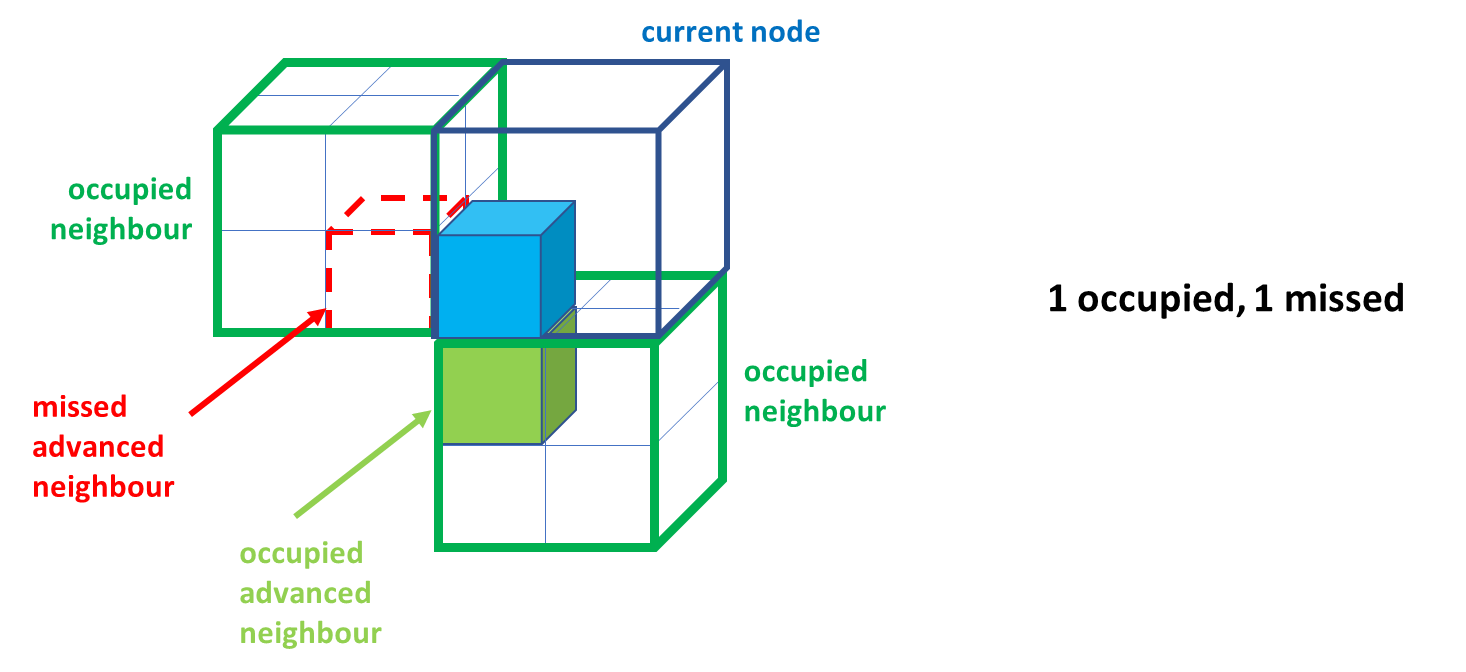


Figure 16: definition of occupied/missed advanced neighbours

##### Incompatibilities between NC9 and new advanced neighbour states Nadv[i][29]

For example, considering the empty configuration NC9 = 0, obviously C[i] must be 0 because there can’t be any occupied advanced neighbour. However, due to “falsely occupied” (see section 3.2.2.4.1), there may be missed advanced neighbours.

Generally speaking, due to “falsely occupied” neighbours, M[i] may be 0 or 1 for any configuration. In the general case, the max value for Nadv[i] depends on

* the position of the child node, i.e. the index i
* the value of the configuration NC

Figures below depict the max possible values for occupied/missed advanced neighbours depending on the index I of the occupancy bit bi and on the neighbour configuration NC.

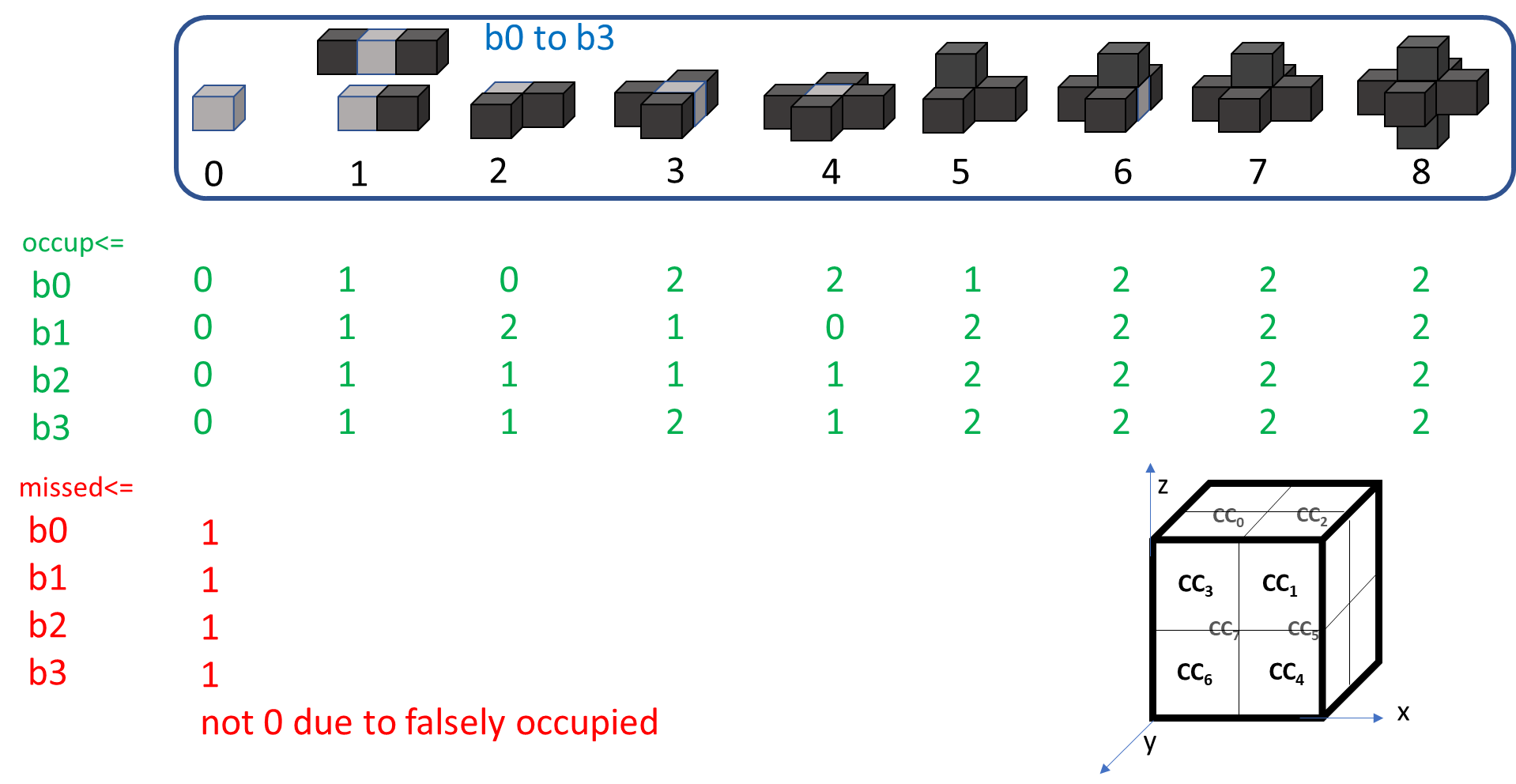


Figure 17: max possible values for occupied/missed advanced neighbour for bits b0 to b3

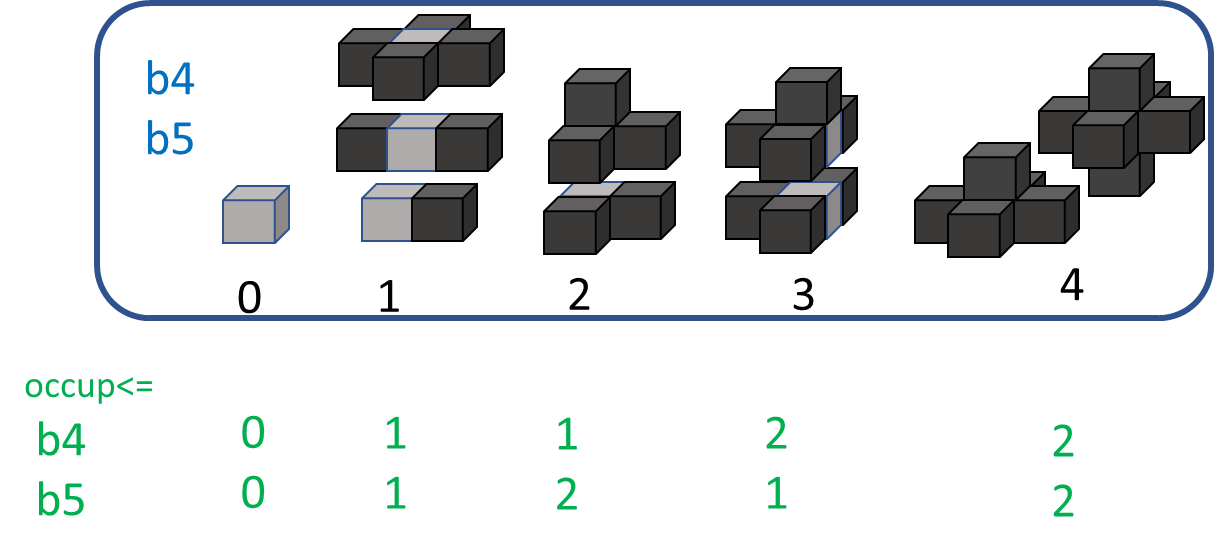


Figure 18: max possible values for occupied/missed advanced neighbour for bits b4 and b5

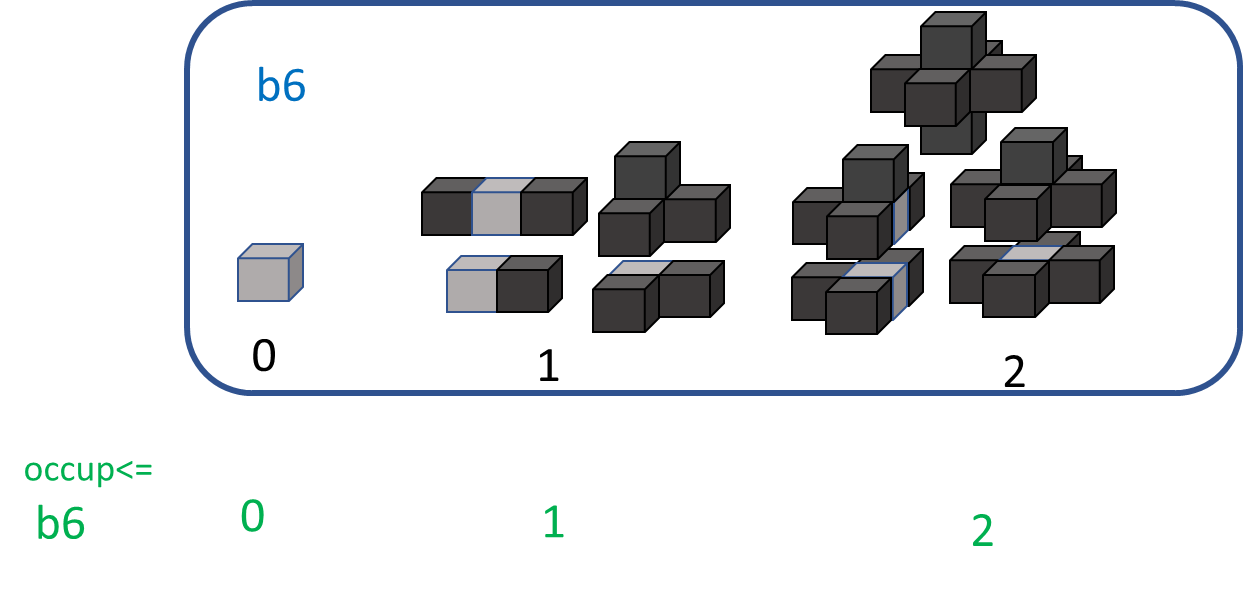


Figure 19: max possible values for occupied/missed advanced neighbour for bit b6

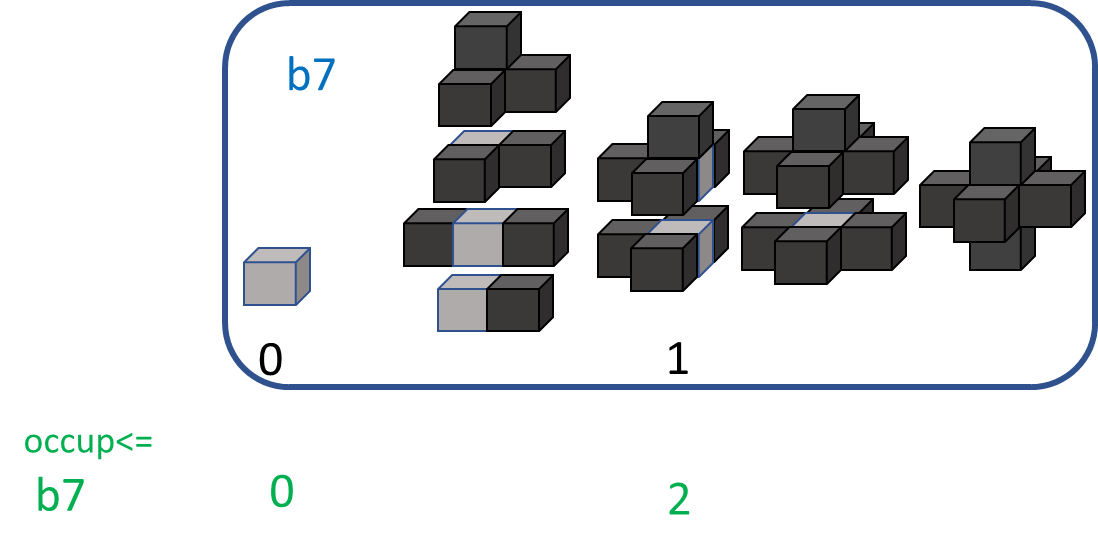


Figure 20: max possible values for occupied/missed advanced neighbour for bit b7

The max value for Nadv[i] is given by

* max Nadv[i] = max missed[i] + 2\* max occup[i]

and, in case i>4, then the LUT\_red = {0,0,1,2,3,3} is applied to this max. This leads to the below in term of number of states.

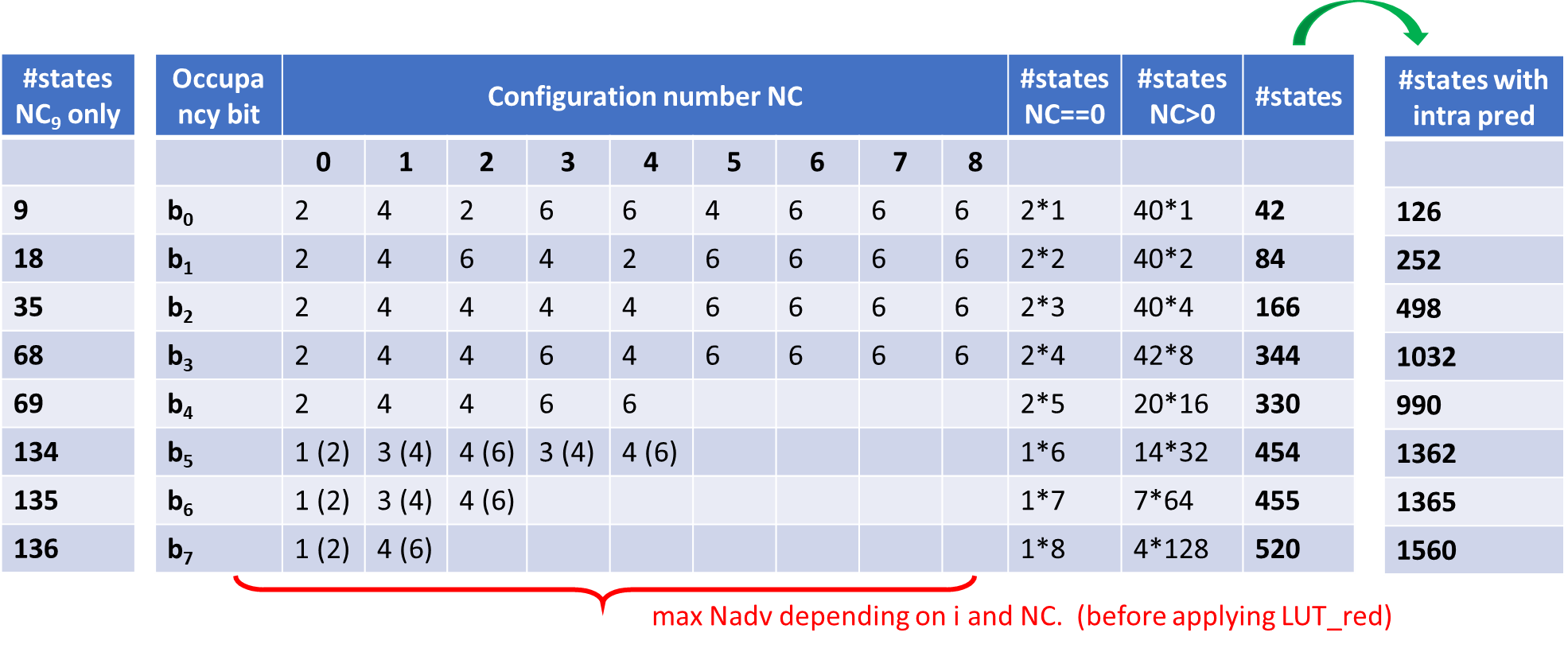


Figure 21: number of states when considering incompatibilities between NC and advanced neighbours

#### Look ahead table [9]

The current octree-based geometry approach exploits an octree-based subdivision of the 3D space in order to efficiently encode regions containing points. At each level of subdivision of the octree, cubes of the same size are subdivided and an occupancy code for each one is encoded.

* + For subdivision level 0, it has single cube of (2C,2C,2C) without any neighbors.
  + For subdivision level 1, it may have up to 8 cubes of dimension (2C-1,2C-1,2C-1) each
  + …
  + For subdivision level L, it may have up to 8L cubes of dimension (2C-L,2C-L,2C-L) each

At each level L, TMC13 defines a set of non-overlapping look-ahead cubes of dimension (2H-C+L,2H-C+L,2H-C+L) each, as described in Figure 22: Look-ahead cubes. Note that the look-ahead cube can fit 23xH cubes of size (2C-L,2C-L,2C-L).

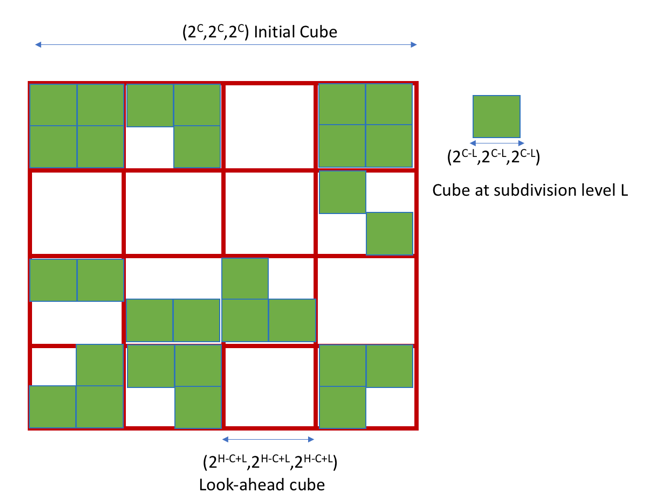


Figure 22: Look-ahead cubes

At each level L, TMC13 encodes the cubes contained in each look-ahead cube without referencing cubes in other look-ahead cubes. This later constraint makes it possible to use a look-up table with a pre-defined (and limited) size to store neighborhood information of the all the cubes within each look-ahead cube. Such a look-up-table-based approach offers the advantage of avoiding the linear search required in [8], at the cost of slightly higher memory usage (i.e., space to store the LUT) and slightly lower compression efficiency.

TMC13 proceeds as follows:

* During the look-ahead phase, the cubes of dimension (2C-L,2C-L,2C-L) in the current look-ahead cube are extracted from the FIFO and a look-up table that describes for each (2C-L,2C-L,2C-L) region of the current look-ahead cube whether it is occupied or empty is filled.
* Once, the look-up table was filled, the encode phase for the extracted cubes begins. Here, the occupancy information for the 6 neighbors is obtained by fetching the information directly from the look up table.
* For cubes on the boundary of the look-ahead cube, the neighbors located outside are assumed to be empty.
* Efficient implementation could be achieved by
  + Storing the occupancy information of each group of 8 neighboring (2C-L,2C-L,2C-L) regions on one byte
  + Store the occupancy bytes in a Z-order to maximize memory cache hits

#### Sibling dependent coding [10]

In TMC13, 10 coding tables are used in arithmetic coding. The switching between the coding tables is controlled by the six neighbours of the current parent node, shown in Figure 23.

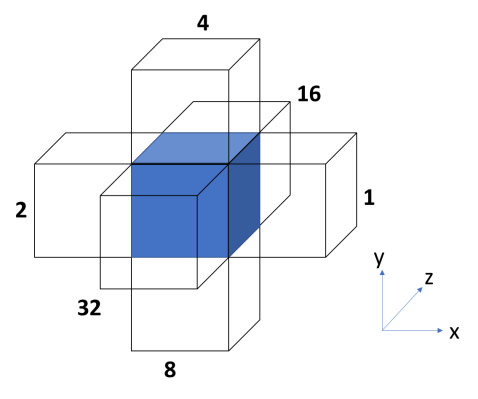


Figure 23: Six neighbours used to decide the switching of the coding table

During encoding/decoding, the neighbour information for the child node is obtained by searching the encoded/decoded nodes. Information of the encoded/decoded node is also updated by the newly encoded/decoded one. In this way, the six-neighbour information of each child node could be fully obtained by the searching and updating. The same process exists in each level of the octree (except for the last one).

Among the 6 neighbours, three of them are easy to be obtained without any searching. They are shown in Figure 24.

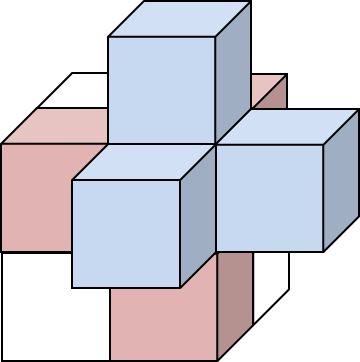
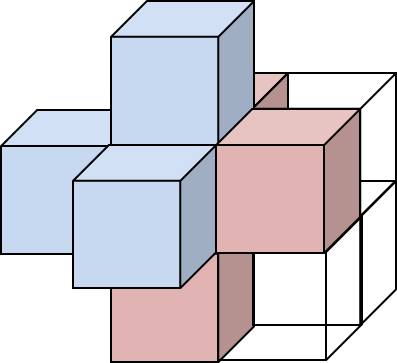
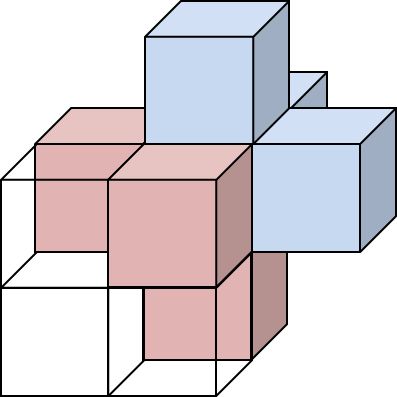


Figure 24: Three of the six neighbours are in the same node in the grand parent point of view. Pink nodes are in the same 2x2x2 node, while the other 3 (blue) are outside of the node.

The occupancy of three children that outside of volume of the current parent node is more difficult to be checked. The searching is needed for these three outside children. The other half of the neighbour information could be obtained by checking the occupancy code of the parent node. The efficiency could be improved if the searching process is skipped, since checking current occupancy code would be simpler than searching in the coded node.

The neighbour information retrieval method only considers the three sibling neighbours in case of the parameter neighbour\_context\_restriction\_flag equal to 1 in TMC13 software. For all the child locations, local neighbours are shown in Figure 25.

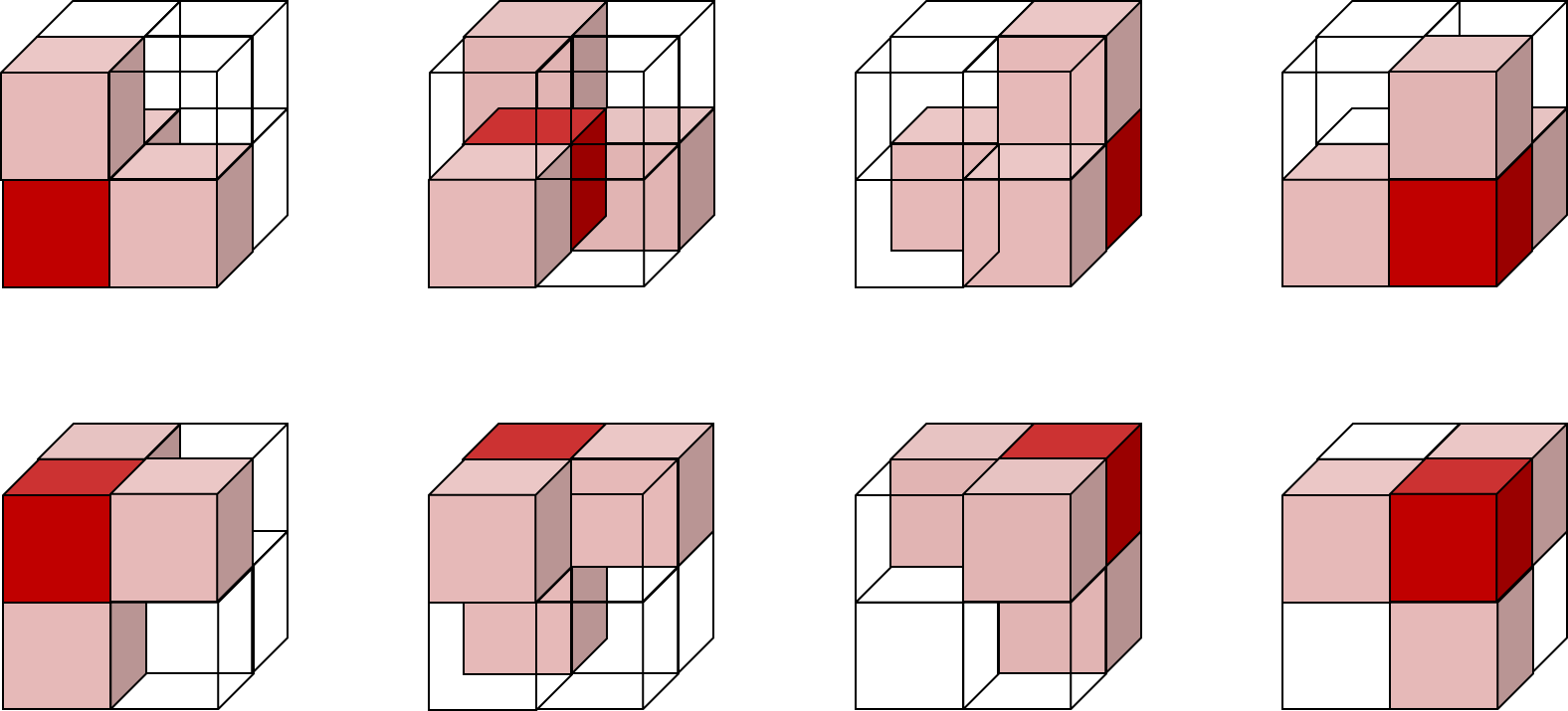


Figure 25: Possible neighbours (pink) in eight cases

In the reduction, the three non-sibling neighbours are considered non-exist. The 64 neighbour configurations are reduced to 6:

* 0 occupied neighbor.
* 1 occupied neighbor, with target node they are horizontal to the x-y plane.
* 1 occupied neighbor, with target node they are vertical to the x-y plane.
* 2 occupied neighbors, with target node they are horizontal to the x-y plane.
* 2 occupied neighbors, with target node they are vertical to the x-y plane.
* 3 occupied neighbors.

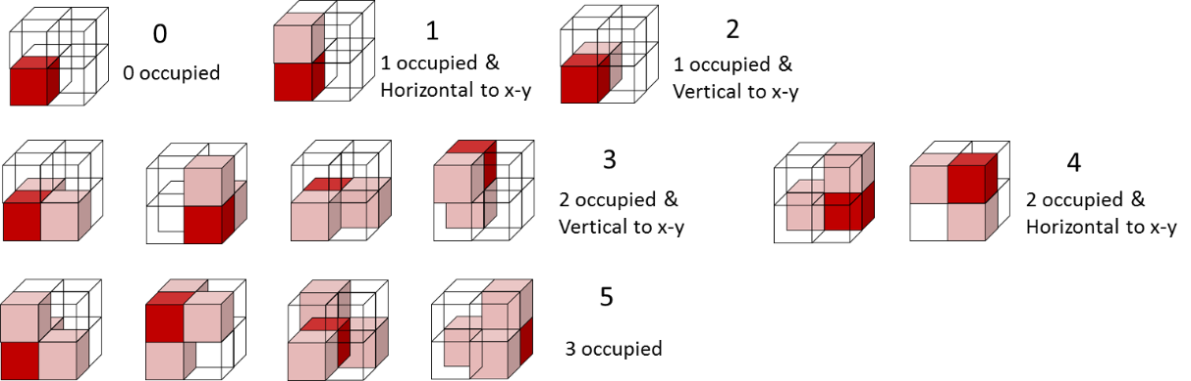


Figure 26: 6 configurations used in selecting coding tables

### Intra prediction [11]

Using the six neighbours of the same depth and sharing a face with a current node does not provide all possible information about the local geometry. Ideally, one would like to use at least the 26 neighbours that share a face, an edge or a vertex with the current node. Obviously, the number of possible patterns of occupancy for the 26 neighbours is by far too high to be directly used in the sets Ɗj of states, even trying complex and tricky direct state reductions. In this section, it is proposed to reduce the 26-neighbour pattern to a ternary information that predicts the value of the occupancy bits bj. This process will be called intra prediction. The practical feasibility of using as many as 26 neighbours has been made possible by a fast and efficient search of neighbours introduced in [23].

#### The occupancy score from the 26 neighbours

Firstly, before applying the geometrical transform that reduces the neighbour configurations from NC to NC10, an occupancy score scorem is computed for each of the eight sub-nodes SNm (m=0,…,7) of a current node by using a weighted sum over the 26 neighbours

where m is the sub-node index, k is a neighbour index, wk,m is the contribution (weight) from neighbour k to sub-node m, and δk is the occupancy status (0 for non-occupied, 1 for occupied) of the neighbour k, as depicted on Figure 27.

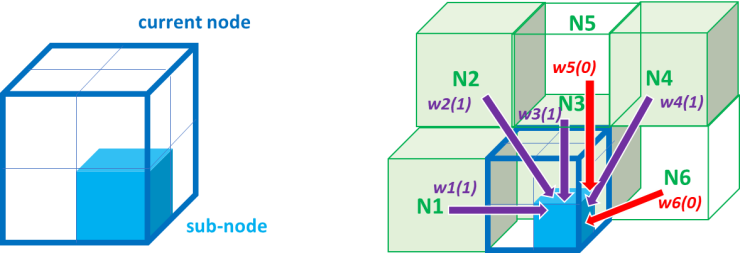


Figure 27: weights between neighbours and sub-nodes of a current node

Using the usual anisotropy argument, the weights are considered as a function W of the Euclidian distance dk,m, between the neighbour k and the sub-node m, and the occupancy status δk.

This function W is found empirically. The distance dk,m can take only eight different values and one gets

where W1 and W2 are two LUTs having eight entries (ordered from the shortest to the longest distance) each and are as follows

W0 = {-1, -6, 12, 20, 14, 28, 22, 12},

W1 = {27, 39, 20, 8, 18, 4, 11, 18}.

The LUTs have been constructed such that a higher score scorem indicates a higher probability of the sub-node SNm to be occupied, and such that the transition between low and high probability of occupancy is the sharpest possible as a function of the score.

#### Score-driven entropy coding

The score can take too many different values to be usable as is in the sets of states. Furthermore, it has been observed that the probability of a sub-node to be occupied depends not only on the score but also on the number No of occupied neighbours among the 26 neighbours.

The score is transformed into a ternary information Predm belonging to the set {“predicted non-occupied”, “predicted occupied”, “not predicted”} of three prediction states by using two thresholds th0(No) and th1(No) that depend on the number No of occupied neighbours. If the score scorem is lower than th0(No), then Predm is set to “predicted non-occupied”; if the score scorem is higher than th1(No), then Predm is set to “predicted occupied”; otherwise the score is between the two thresholds and Predm is set to “not predicted”.

After applying the geometrical transform to reduce the neighbouring configuration from NC to NC10, each of the child node CCj inherits a prediction value Pred[j] and the set of states becomes:

This leads to sizes of the eight sets Ɗj of states multiplied by a factor three because the intra prediction is taken independent on other neighbouring techniques. Basically, this means that the set of states is three copies of the sub-set without prediction, i.e.

The two thresholds are determined empirically for the five cases of occupied neighbours No≤9, No=10, No=11, No=12 and No≥13, and practically obtained from the following two LUTs:

TH0 = {62, 60, 61, 59, 59},

TH1 = {67, 66, 65, 66, 64}.

## Trisoup geometry encoding/decoding

Trisoup codec is a geometry coding option that represents the object surface as a series of triangle mesh. It is applicable for a dense surface point cloud. The decoder generates point cloud from the mesh surface in the specified voxel granularity so that it assures the density of the reconstructed point cloud.

If the Trisoup geometry codec is used, then the parameter *trisoup\_node\_size* defines the size of the triangle nodes in unit of voxel. The octree encoding and decoding stop at leaf level , in which case the leaf nodes of the octree represent cubes of width , or blocks, and the octree is said to be pruned. In the latter case, Inferred Direct Coding Mode is not allowed.

### Determining vertices

If , then the blocks are 2 x 2 x 2 or larger, and it is necessary to represent the collection of voxels within the block by some model. Geometry is represented within each block as a surface that intersects each edge of the block at most once. Since there are 12 edges of a block, there can be at most 12 such intersections within a block. Each such intersection is called a vertex. A vertex along an edge is detected if and only if there is at least one occupied voxel adjacent to the edge among all blocks that share the edge. The position of a detected vertex along an edge is the average position along the edge of all such voxels adjacent to the edge among all blocks that share the edge.

### Entropy encoding of vertices

Vertices, nominally being intersections of a surface with edges of a block, are shared across neighbouring blocks, not only guaranteeing continuity across blocks of the reconstructed surface, but also reducing the number of bits required to code the collection of vertices. The set of vertices is coded in two steps. In a first step, the set of all the unique edges (or segments) of occupied blocks is computed, and a bit vector (or segment indicator) determines which segments contain a vertex and which do not. In a second step, for each segment that contains a vertex, the position of the vertex along the segment is uniformly scalar quantized to a small number of levels, typically equal to the block width if the geometric spatial resolution is desired to approximate the voxel resolution, but it could be any number of levels. The segment indicators and the vertex positions are entropy coded by an arithmetic coder. The geometry bitstream becomes a compound bitstream comprising octree, segment indicator, and vertex position bitstreams.

### Triangle reconstruction

The vertices on the edges of a block determine a surface through the block. The surface is a non-planar polygon, triangulated as follows. Let , be the coordinates of the vertices on the edges of the block, in any order. Compute the centroid

the mean-removed coordinates

and the (scaled) variances

Find the minimum . If achieves the minimum, then project each vertex onto the x axis (the “dominant” axis) as , and onto the (y,z) plane as , where is the center of the block. Otherwise, if achieves the minimum, then project each vertex onto the y axis as , and onto the (x,z) plane as Otherwise, project each vertex onto the z axis as , and onto the (x,y) plane as

Compute the arctangent , and sort the angles in increasing order, breaking ties in order of increasing . For this order of the vertices, form triangles according to Table 1.

Table 1: Triangles formed from vertices ordered 1, …, n.

|  |  |
| --- | --- |
|  | triangles |
| 3 | (1,2,3) |
| 4 | (1,2,3), (3,4,1) |
| 5 | (1,2,3), (3,4,5), (5,1,3) |
| 6 | (1,2,3), (3,4,5), (5,6,1), (1,3,5) |
| 7 | (1,2,3), (3,4,5), (5,6,7), (7,1,3), (3,5,7) |
| 8 | (1,2,3), (3,4,5), (5,6,7), (7,8,1), (1,3,5), (5,7,1) |
| 9 | (1,2,3), (3,4,5), (5,6,7), (7,8,9), (9,1,3), (3,5,7), (7,9,3) |
| 10 | (1,2,3), (3,4,5), (5,6,7), (7,8,9), (9,10,1), (1,3,5), (5,7,9), (9,1,5) |
| 11 | (1,2,3), (3,4,5), (5,6,7), (7,8,9), (9,10,11), (11,1,3), (3,5,7), (7,9,11), (11,3,7) |
| 12 | (1,2,3), (3,4,5), (5,6,7), (7,8,9), (9,10,11), (11,12,1), (1,3,5), (5,7,9), (9,11,1), (1,5,9) |

### Triangle rasterization

To derive a decoded geometry point cloud from the trisoup in the specified voxel resolution, it is checked if each voxel in the bounding box intersects with the triangles.

More precisely, following steps are conducted.

* To prepare 6 unit vectors (±1, 0 ,0), (0, ±1, 0), (0, 0, ±1) around each triangles.
* To check if the unit vector and the triangle intersect, and if yes, the intersection is calculated and output as decoded voxel.

The intersection check is independent among the vectors, thus the point generation process can be done in parallel.

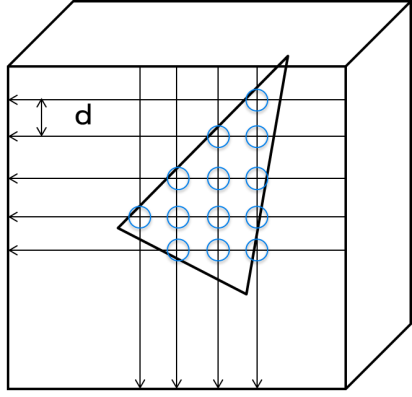


Figure 28: intersection check on voxel grid

## Geometry Entropy encoding

### Binary coding of occupancy code

TMC13 supports two binary coding mode of occupancy code. In this section, both methods are described in order.

#### Bitwise based binary coding of occupancy code [12]

TMC13 has supported a binarization of the entropy coder that codes the occupancy information of the octree. Therefore, a cascade of binary coders is used together with dependency reduction when using ten neighbour configurations in order to obtain a “reasonable” number of entropy coders, with no compression loss.

An improvement of this method has been introduced to obtain a tunable number of binary coders at the price of a slight compression loss with a “more reasonable” number of coders. This method is compatible with intra schemes as described in section 3.2.3 for instance.

The bitwise based binary coder is depicted in yellow and orange as additional blocks in the flowchart of (encoder) and (decoder).

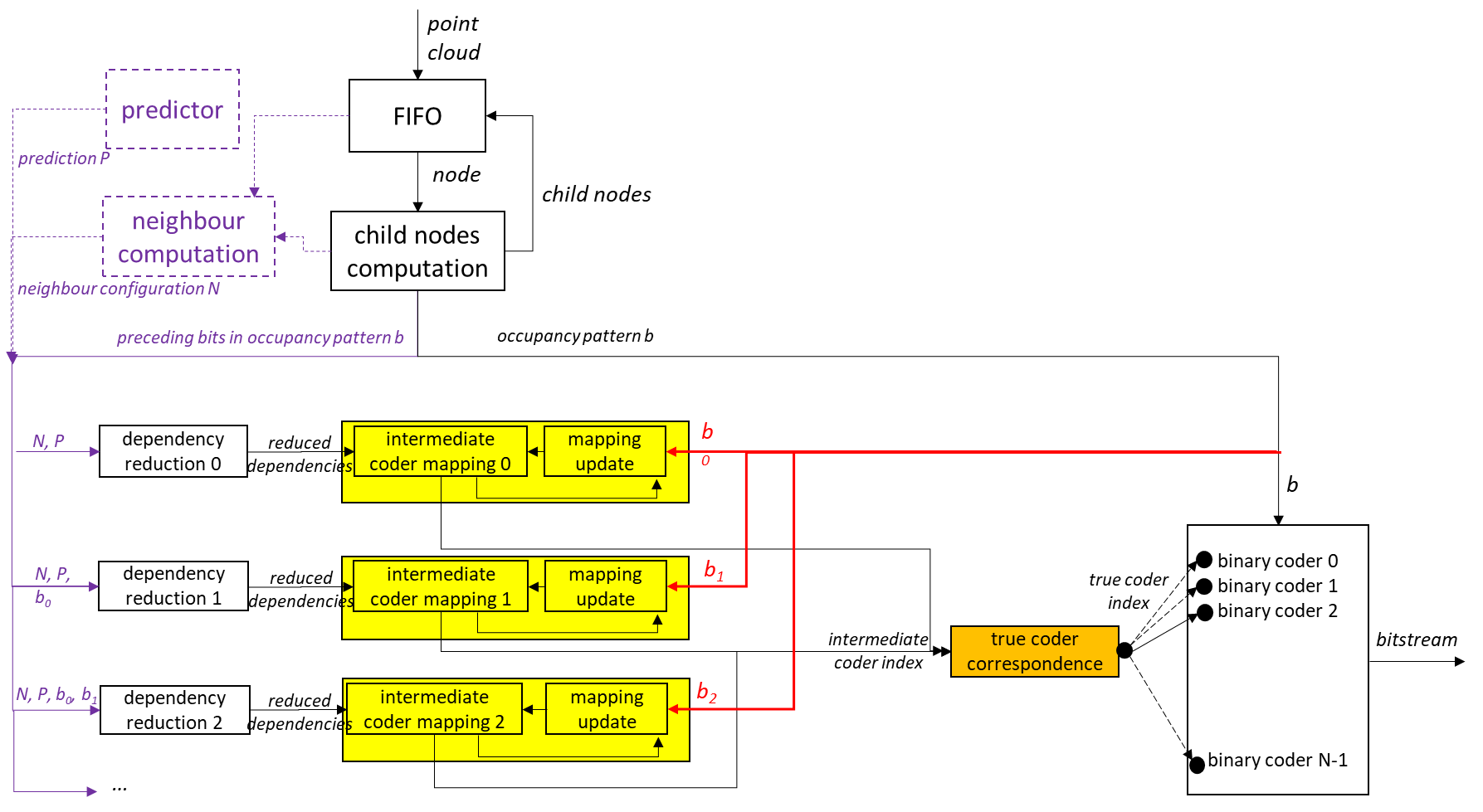


Figure 29: flowchart of the proposed encoder for the new binarization scheme

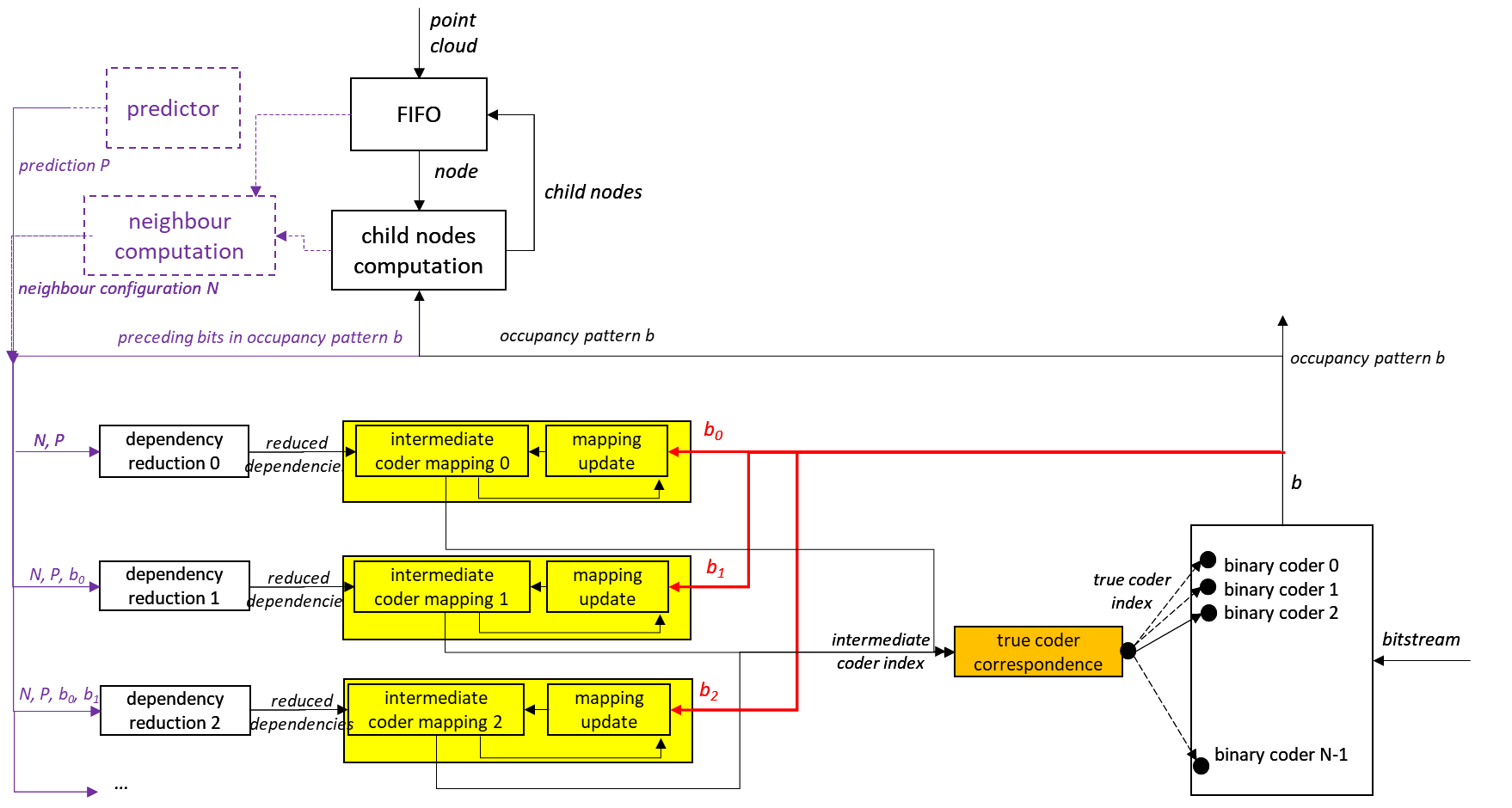


Figure 30: flowchart of the proposed encoder for the new binarization scheme

##### Binarization

A current cube has eight child cubes that may or may not be occupied by at least one point of the point cloud. To each child cube CCi is attached an occupancy bit bi representative of the occupancy state of the child CCi being occupied (bi=1) or non-occupied (bi=0). The concatenation of the eight bits bi forms an eight-bit integer b between 1 and 255; the value 0 being forbidden because at least one child is occupied by construction of the octree.

In order to profit from local geometry correlation, the bits bi should not be coded independently, and an ideal binarization can be obtained based on the well-known conditional entropy formula:

The first bit b0 is coded directly by a binary coder, the second bit b1 is coded by a binary coder depending on the value of b0, etc. The last bit b7 is coded depending on the 128 possible values of b0…b6. A practical implementation to solve these binary coders dependencies is desirable. A straightforward answer may be to introduce contexts as done for CABAC in video coding. For example, one may have 128 contexts to code b7 and the context is chosen based on the value of b0…b6. This would lead to 1+2+4+…+128=255 contexts for the eight bits bj to code. This is still manageable, however when introducing several prediction tools, the number of contexts may easily increase drastically to more than a few thousands, and HW implementations become difficult or impractical.

In the next section, a novel binarization scheme is described which reduces the number of needed contexts and allows for the use and development of additional occupancy prediction tools that will drive the binary entropy coders.

##### Optimal Binarization with Update On-the-fly

The binarization process has a fixed (small) number N of binary coders that are ideally arithmetic coders with an evolving internal probability, like CABAC or Dirac [15] coders. A bit bj representing the occupancy of a child node is coded using a binary coder Ci chosen among the coders C1 to Cn. The choice of the coder index i is performed depending on a dependency state D as depicted on Figure 31.

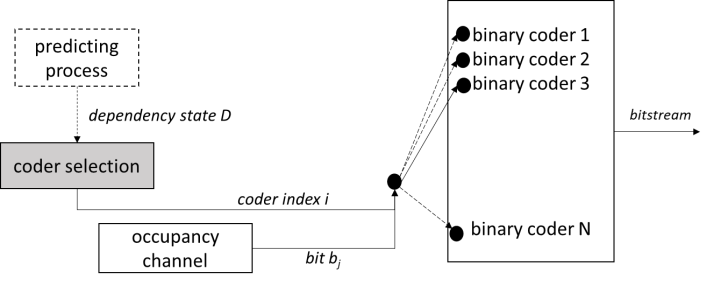


Figure 31: on choosing a binary coder to code an occupancy bit bj depending on a dependency state D

In general, the state D is an element of a set Ɗj, indexed by j from the child node CCj, of states. For example, the sets Ɗj can contain {b0...bj-1, NC, P1,…,PK} where

* NC is a neighbouring configuration that can take 10 different values, and
* the Pk’sare predictors for the occupancy of the cube associated with the bit bj. A predictor can typically take three values, namely “predicted occupied”, “predicted non-occupied” or “not predicted”.

Consequently, the state D can take 10.2j.3K different values, and the size of the set Ɗj of states can easily reach thousands of elements or even more. Example of construction of NC and Pk are explicated later in this paper.

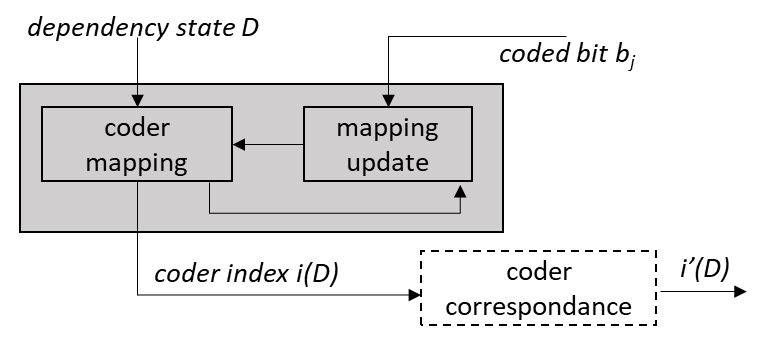


Figure 32: update process of the coder index i(D)

The coder selection consists of two processes. Firstly, a coder mapping provides a coder index i(D) obtained from the dependency state D, see Figure 32. Practically the mapping is a LUT having as many entries as elements in the set Ɗj of states. The LUT obviously depends on the index j of the child cube, thus one has eight LUTs that map D to i(D). Secondly, once the bit bj is (de)coded, the D-th entry of the mapping is updated from i(D) to a new value iupdate.

The mapping is based on a simple model of a channel with memory of L=10 symbols. A fixed theoretical probability pi (not to be confused with the internal evolving probability of the coder). of coding the symbol 1 is associated with each binary coder Ci. Consequently, in theory, the probability of getting bj=1 for the dependency state D is pi(D). After coding the bit bj, this theoretical probability is modified depending on the value of bj leading to a new probability pnew obtained by the relation, assuming a memory of L symbols:

The coder index is then updated to point to the coder whose probability piupdate is the closest to pnew, i.e.

Practically, the values iupdate obtained from the couple i and bj are precomputed into two LUTs, independent on j and D, such as to get a very compact update step by:

The fixed theoretical probabilities pi are determined such as to cover the interval [0,1] in some optimal way as follows. Let ε>0 be a positive real number. The set of coders {Ci} is said to be a set of ε-coders if

1. the fixed probabilities pi are an increasing sequence relative to the index i and cover the interval (0,1)
2. coding the symbols of a binary channel B, with associated probability pB in [pi-1, pi+1] of the symbols to be 1, using the coder Ci leads to an extra per-symbol entropy of at most ε relative to the coding with an optimal coder with associated probability pB.

In other words, one has an ordered set of coders, and coding a symbol with a marginally sub-optimal coder (Ci±1 instead of Ci) leads to at most ε extra bit per symbol. Therefore, an inaccuracy Δ in the coder index relative to the optimal coder index leads to an extra entropy bounded by Δε.

A set of ε-coders is said to be optimal if its number of coders is minimum among all possible sets of ε-coders. Covering the interval (0,1) is understood loosely as p1 being arbitrarily close to 0 and pN arbitrarily close to 1. Practically, one chooses ε such as to obtain a desired number N(ε) of coders. The probabilities pi can be determined using the following algorithm:

1. start with p1 arbitrary small
2. determine iteratively pi+1 from pi, until pi+1 is arbitrary close to 1, as follows
   1. for a probability p, define an entropy error E relatively to pi by
   2. take pi+1 as the lowest probability that guarantees the error to be bounded by ε, i.e. .

The value ε = 1.0870e-04 provides an optimal set of N(ε)=256 ε-coders.

It has been observed that the update step requires a fine granularity in term of probabilities pi, but the actual number of coders can be reduced without impacting noticeably the compression performance. Therefore, a coder correspondence (see Figure 23) has been added to map the coder index i(D) in [1,256] to an actual coder index i’(D) in [1,32]. The correspondence is simply a division by 8, and only 32 binary coders are used while maintaining an update working on 256 values. The binary coders Ci are initialized to their fixed theoretical probabilities pi before starting coding the octree.

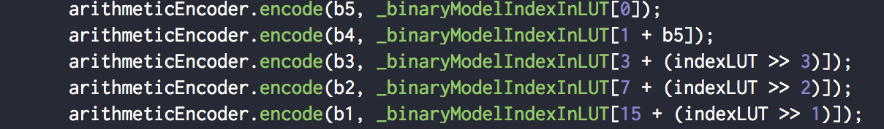
#### Bytewise based binary coding for occupancy code [13]

In order to efficiently encode the non-binary occupancy values with a binary arithmetic encoder TMC13 introduce:

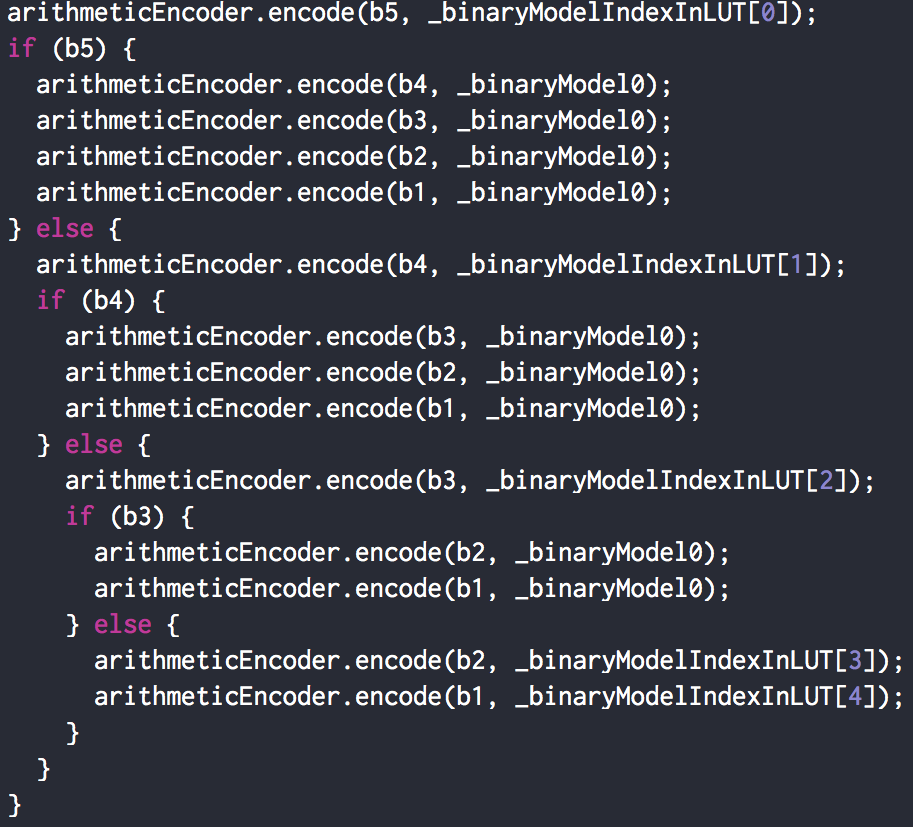
* an adaptive look up table (A-LUT), which keeps track of the N (e.g., 32) most frequent occupancy symbols,
* a cache which keeps track of the last different observed M (e.g., 16) occupancy symbols.

The algorithm proceeds as follows:

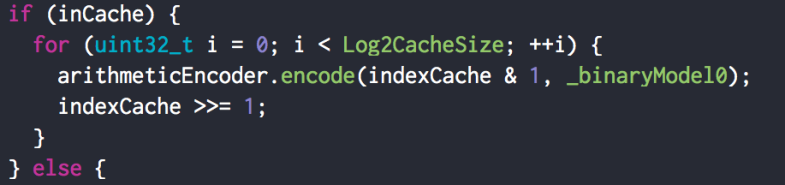
* + The A-LUT is initialized with N symbols provided by the user or computed offline based on the statistics of a similar class of point clouds.
  + The cache is initialized with M symbols provided by the user or computed offline based on the statistics of a similar class of point clouds.
  + Every time a symbol S is encoded the following steps are applied
    1. A binary information indicating whether S is the A-LUT or not is encoded.
    2. If S is in the A-LUT, the index of S in the A-LUT is encoded by using a binary arithmetic encoder
       - Let (b1, b2, b3, b4, b5) be the five bits of the binary representation of the index of S in the A-LUT. Let b1 be the less significant bit and b5 the most significant bit.
       - We propose two approaches to encode the index by using either 31 or 5 adaptive binary arithmetic contexts as shown in the pseudo-codes below (Note: \_binaryModel0 is a static binary arithmetic context, and \_binaryModelIndexInLUT[] is an array of adaptive binary arithmetic contexts)
         * 31 Contexts



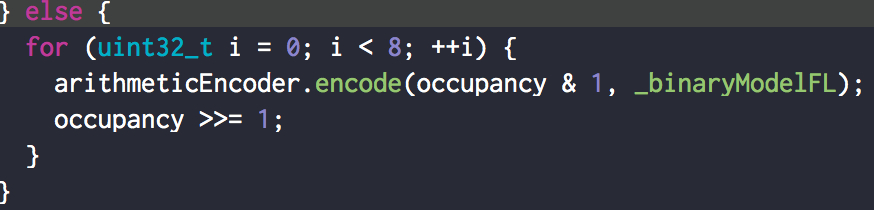
* + - * + 5 contexts



* + 1. If S is not the A-LUT, then
       - A binary information indicating whether S is in the cache or not is encoded.
       - If S is in the cache, then the binary representation of its index is encoded by using a binary arithmetic encoder
         * In the current implementation, the binary representation of the index is encoded by using a single static binary context as described in the pseudo-code below



* + - * Otherwise, if S is not in the cache, then the binary representation of **S** is encoded by using a binary arithmetic encoder
        + In the current implementation, the binary representation of S is encoded by using a single adaptive binary context as described in the pseudo code below



* + - * The symbol S is added to the cache and the oldest symbol in the cache is evicted.
    1. The number of occurrences of the symbol S in A-LUT is incremented by one.
    2. The list of the N most frequent symbols in the A-LUT is re-computed periodically (Note: the update period increases exponentially).
    3. At the start of each level of the octree subdivision, the occurrences of all symbols are reset to zero. The occurrences of the N most frequent symbols are set to 1.
    4. When the occurrence of a symbol reaches a user-defined maximum number (e.g., 1024), the occurrences of the all the symbols are divided by 2 to keep the occurrences within a user-defined range.

The approach described above makes it possible to reduce the number of adaptive binary contexts to 340 or 80, depending on the number of binary contexts used to encode the index in the adaptive LUT (i.e., 31 or 5).

This number could be further reduced by applying the adaptive neighborhood context selection described below. The idea is to reduce the number of neighborhood contexts from 10 to a lower number NC (e.g., 6), by assigning a separate context to the (NC-1) most probable neighborhood configurations, and making the neighborhood contexts corresponding to the least probably neighborhood configurations share the same context. The algorithm proceeds as follows:

* Before starting the encode, initialize the occurrences of the 10 neighborhood configurations:
  + Set all occurrences to 0
  + Set the occurrences based on offline/online statistics or based on user-provided information
* At the beginning of each subdivision level of the octree
  + Determine the (NC-1) most probable neighborhood configurations based on the statistics collected during the encoding of previous subdivision level
  + Compute a look-up table NLUT, which maps the indexes of the (NC-1) most probable neighborhood configurations to the numbers 0, 1, …, (NC-2) and maps the indexes of the remaining configurations to NC-1
  + Initialize the occurrences of the 10 neighborhood configurations to 0
* During the encoding
  + increment the occurrence of a neighborhood configuration by one each time such a configuration is encountered
  + use the look-up table NLUT[] to determine the context to use to encode the current occupancy values based on the neighborhood configuration index

If NC is set to 6, the variation described above makes it possible to use 48 adaptive binary contexts.

A high throughput version of the scheme described above could be achieved by encoding/decoding the occupancy symbols as a set of separate/independent streams, which could be processed in parallel. The idea is to encode/decode each P (e.g., 4, 8, or 16) consecutive occupancy symbols by using independent, which write to P independent binary sub-streams. The final bistream is obtained by concatenating the P sub-bitstreams. A header information describing the offset to each sub-stream is also included.

* + The occupancy symbols are accumulated in a fifo
  + If the fifo has at least P symbols
    - P symbols are extracted from fifo and encoded as described above by using P separate arithmetic encoders and with independent arithmetic contexts
    - The P symbols are pushed to the LUT or to the cache only after all symbols were arithmetically encoded.
  + If we reach the end of the encode process and less than P symbols are in the fifo, the fifo is padded with the last observed symbol until it has a size of P, then the P symbols are encoded as described above.

### Dirac / SMPTE VC-2 [14]

The Dirac video codec [15] was developed by the BBC with intra profiles being standardised by the SMPTE as VC2[16]. One of the principal objectives of the Dirac project was to produce a royalty free video codec.

For entropy coding, Dirac|VC-2 defines a context adaptive binary arithmetic codec using 32bit arithmetic and 16bit probabilities. Probabilities are updated on the basis that the current probability is an estimate for the number of symbols coded.

### Bypass coding of bypass bins [45]

G-PCC introduces an alternative to using an arithmetic entropy codec to code incompressible bypass symbols, using instead an approach to so-called bypass coding that bypasses the arithmetic codec completely without incurring additional signaling overheads.

It involves partitioning the symbol stream into two sub-streams: an arithmetically coded sub-stream and a bypass sub-stream. A data unit is constructed as per Figure 33 as the concatenation of the existing data unit header and (forward) AEC sub-stream with the byte reversed bypass sub-stream. Since the length of the entire data unit is known, using this construction it is not necessary to signal the length of either sub-stream.

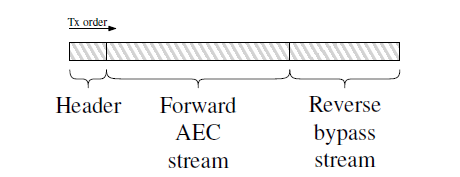


Figure : Construction of a data unit payload

From the point of view of a decoder, two read pointers are maintained, one for the payload and forward AEC sub-stream, the other for the reversed sub-stream. At the start of parsing a data unit, the read pointers are initialized to the first and last bytes of the data unit. For each byte read from each sub-stream the associated read pointer is respectively incremented or decremented.

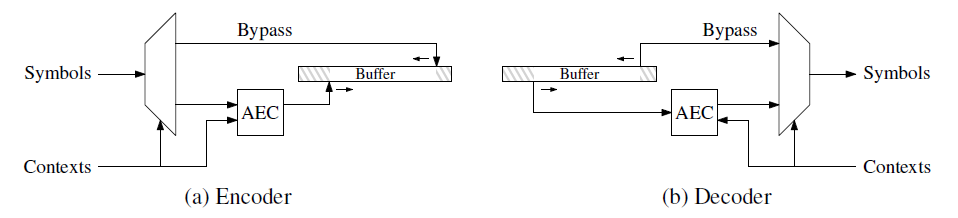


Figure : Conceptual semantics

## Attributes transfer (recolouring)

Given the input point cloud positions/attributes and the reconstructed positions , the objective of the attributes transfer procedure is to determine the attribute values that minimize the attribute distortions.

The implemented approach proceeds as follows:

* Letand be the input and the reconstructed positions, respectively.
* Let and be the number of points in the original and the reconstructed point clouds, respectively.

If duplicated point are merged, then , otherwise .

* For each point in the reconstructed point cloud, let be its nearest neighbour in the original point cloud and the attribute value associated with .
* For each point in the reconstructed point cloud, let be the set of points in the original point cloud that share as their nearest neighbour in the reconstructed point cloud, is the number of elements in , and is one of the elements of Note that could be empty or could have one or multiple elements.
* If is empty, then the attribute value is associated with the point .
* If is not empty, then we proceed as follows:
* The attribute value associated with the point is obtained by Eq.1.

Note: currently --searchRange = 0 shall be used for coding in TMC13.

### Distance-weighted color transfer [48]

The improved color transfer algorithm is introduced in the following. For each point of the target :

1. Find the (1 < ) nearest neighbors in source to and create a set of points denoted by .
2. Find the set of source points that belongs to their set of nearest neighbors. Denote this set of points by .
3. Compute the distance-weighted average of points in and by:

where denotes the Euclidian distance between the points and , and denotes the color of point .

1. Compute the average (or the weighted average with the number of points of each set as the weights) of and and transfer it to .

## Attribute coding (Predicting Transform)

### Level of detail generation

The level of detail (LOD) generation process (see Figure 35) re-organizes the points into a set of refinement levels , according to a set of Euclidean distances specified by the user. Note that the distances need to satisfy the following two conditions:

* , and
* .

The re-ordering process is deterministic and operates on the quantized positions ordered according to the octree decoding process. It is applied at both the encoder and the decoder side. It proceeds as follows:

* First, all the points are marked as non-visited and the set of visited points, denoted as , is set as empty.
* The algorithm proceeds iteratively. At each iteration , the refinement level is generated as follows:
  + The algorithm iterates over all the points.
  + If the current point has been visited, then it is ignored.
  + Otherwise, the minimum distance D of the current point to the set is computed.
  + If D is strictly lower than , then the current point is ignored,
  + If D is higher or equal than , then the current point is marked as visited and added to both and .
  + This process is repeated until all the points are traversed.
* The level of detail l, , is obtained by taking the union of the refinement levels .
* This process is repeated until all the LODs are generated or until all the vertices have been visited.

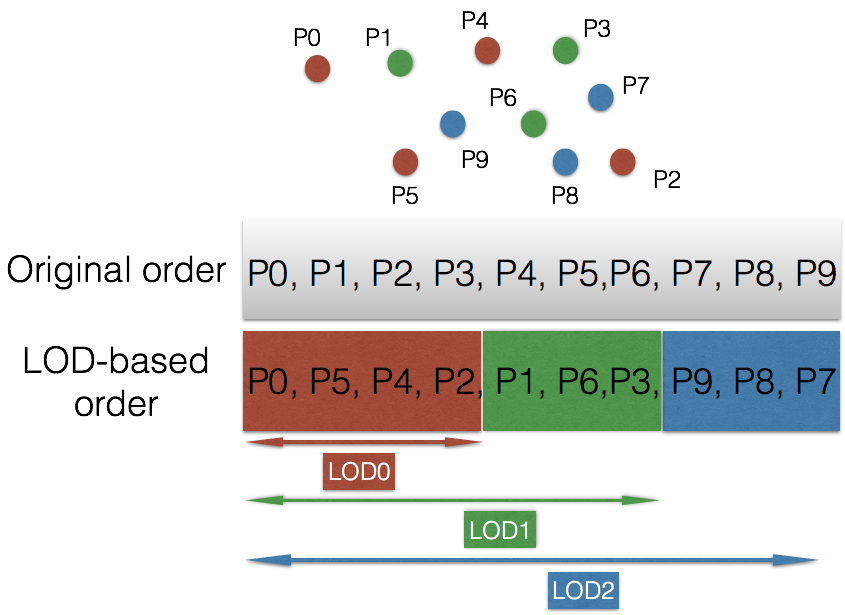


Figure 35: Level of detail generation process

### Scalable complexity implementation of LOD generation [19]

In order to provide a scalable complexity implementation of the lifting scheme, G-PCC introduces to:

* Use a bottom-up approach to build the LODs instead of the top-down technique
* Use an approximate nearest neighbor search instead of an exact nearest neighbor search to accelerate LOD and predictor creation.

Let be the set of positions associated with the point cloud points and let be the Morton codes associated with . Let and be the two user-defined parameters specifying the initial sampling distance and the distance ratio between LODs, respectively. Note that .

First the points are sorted according to their associated Morton codes in an ascending order. Let be the array of point indexes ordered according to this process. The algorithm proceeds iteratively. At each iteration , the points belonging to the LOD are extracted and their predictors are build starting from until all the points are assigned to an LOD. More precisely, the algorithm proceeds as follows:

* The sampling distance is initialized with
* For each iteration, where Let be the set of indexes of the points belonging to -th LOD and the set of points belonging to LODs higher than . and are computed as follows.
  + First, and are initialized
    - . Otherwise,
  + The point indexes stored in the array are traversed in order. Each time an index is selected and its distance to the most recent SR1 points added to is computed. SR1 is a user-defined parameter that controls the accuracy of the nearest neighbor search. For instance, SR1 could be chosen as 8 or 16 or 64. The smaller the value of SR1 the lower the computational complexity and the accuracy of the nearest neighbor search. The parameter SR1 is included in the bitstream. If any of the SR1 distances is lower than , then is appended to the array Otherwise, is appended to the array .
* This process is iterated until all the indexes in are traversed. At this stage, and are computed and will be used in the next steps to build the predictors associated with the points of . Let \ (where \ is the difference operator) be the set of points that need to be added to LOD(k-1) to get LOD(k). For each point in , we would like to find the -nearest neighbors ( is user-defined parameters that controls the maximum number of neighbors used for prediction) of in and compute the prediction weights associated with . The algorithm proceeds as follows.
  + Initialize a counter
  + For each point in
    - Let be the Morton code associated with and let be the Morton code associated with j-th element of the array
    - While (, incrementing the counter j by one (
    - Compute the distances of to the points associated with the indexes of that are in the range [j-SR2, j+SR2] of the array and keep track of the -nearest neighbors and their associated squared distances . SR2 is a user-defined parameter that controls the accuracy of the nearest neighbor search. Possible values for SR2 are 8, 16, 32, and 64. The smaller the value of SR2 the lower the computational complexity and the accuracy of the nearest neighbor search. The parameter SR2 is included in the bitstream. The computation of the prediction weights used for attribute prediction remains unchanged compared to [4].
      * + If the distance between the current point and the last processed point is lower than a threshold, use the neighbors of the last point as an initial guess and search around them.
        + The previous idea could be generalized to n=1,2,3,4… last points
        + Exclude points with a distance higher that a user-defined threshold.

### Regular sampling based LoD generation [33]

A low-complexity LOD generation procedure, which better captures the initial point distribution and can enable a more efficient prediction for non-smooth attribute signals defined over irregularly sampled point clouds was introduced. It has a linear complexity and does not requires a second reordering of the points. Instead, it directly leverages the Morton-based ordering of the points that is used to determine prediction neighbors.

Let be the set of ordered indexes and the associated LOD that represents the entire point cloud. Instead of defining a set of sampling distances, it defines a set of sampling rates denoted , where is an integer describing the sampling rate for the LOD (e.g., ). The ordered array of indexes associated with LOD , denoted as , is computed by subsampling , while keeping one index out of every indexes. The sampling rate could be further updated with an LOD in order to better adapt to the point cloud distribution. More precisely, the encoder may explicitly encode in the bitstream for a predefined group of points (e.g., each consecutive H=1024 points) different values or updates to be applied to the latest available value. can be automatically determined based on the characteristics of the signal and/or the point cloud distribution, previous statistics, or could be fixed. Different subsampling rates may be defined per attributes (e.g., color, reflectance) and per channel (e.g., Y and U/V)

### Simplified prediction structure in case of LoD equal one [22]

Let be the set of positions associated with the point cloud points and let be the Morton codes associated with . First, the points are sorted according to their associated Morton codes in an ascending order. Let be the array of point indexes ordered according to this process. The encoder/decoder compresses/decompresses respectively the points according to the order defined by . At each iteration , a point is selected. The distances of to the (e.g., =64) previous points are analyzed and the (e.g., =3) nearest-neighbors of are selected to be used for prediction in the same manner as in the current version of G-PCC.

### Intra LoD prediction on attribute predicting transform [31]

An EnableReferringSameLoD flag is introduced to control the reference structure for predicting transform in order to keep tradeoff of coding efficiency and parallel processing. If EnableReferringSameLoD flag is set to 1, 3D point in the same LoD could be referred. Additionally, the function to allow to switch this tool on for certain LoD was also introduced. This tool is called by intra LoD prediction.

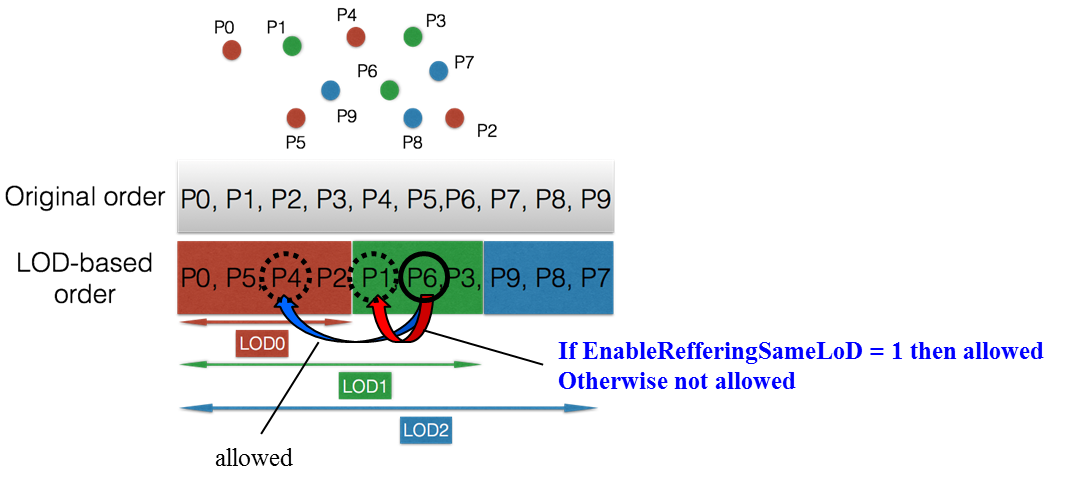


Figure 36: reference structure for intra LoD prediction

### Interpolation-based prediction

The attributes associated with the point cloud are encoded/decoded in the order defined by the LOD generation process. At each step, only the already encoded/decoded points are considered for prediction. More precisely, the attribute values are predicted by using a linear interpolation process based on the distances of the nearest neighbours of point i. Let be the set of the k-nearest neighbours of the current point i, and let be their decoded/reconstructed attribute values and their distances to the current point. The predicted attribute value is given by:

.

The number of nearest neighbours, k, is a parameter that is determined by the encoder for each point, and arithmetically encoded.

### Adaptive predictor selection [17]

In current TMC13 attributes coding, LoD (Level of Detail) of each 3D points is generated based on the distance of each points, then the attributes value of 3D points in each LoD is encoded by applying prediction in LoD-based order (Figure 35). For example, the attributes value of P2 is predicted by calculating the distance based weighted average value of P0, P5 and P4 which were encoded or decoded prior to P2.

In this method, multiple predictor candidates are created based on the result of neighbor point search in generating LoD. For example, when the attributes value of P2 is encoded by using prediction, a distance based weighted average value of P0, P5 and P4 is set to predictor index equal to 0. Then, the value of nearest neighbor point P4 is set to predictor index equal to 1. Moreover, the value of next nearest neighbor point P5 and P0 are set to predictor index equal to 2 and 3 respectively (Table 2). After creating predictor candidates, best predictor is selected by applying a rate-distortion optimization procedure and then, selected predictor index is arithmetically encoded.

Table 2: Sample of predictor candidate for attributes coding

|  |  |
| --- | --- |
| Predictor index | Predicted value |
| 0 | average |
| 1 | P4 (1st nearest point) |
| 2 | P5 (2nd nearest point) |
| 3 | P0 (3rd nearest point) |

The maximum number of predictor candidate (MaxNumCand) is defined and it is encoded into attributes header. In the current implementation, MaxNumCand is set to equal to 5 (= numberOfNearestNeighborsInPrediction + 1) and it is used in encoding and decoding predictor index with truncated unary binarization.

This method also includes the condition in which the variability of its neighborhood is computed to check how different the neighbor values are and if the variability is higher than a threshold, predictor selection is conducted.

### Quantization and inverse quantization of attribute prediction residuals

Let be the input attribute values and the predicted attribute values computed as described in the previous section. The attribute prediction residuals are given by:

.

The quantization and inverse quantization procedures of the attribute prediction residuals are described in Figure 37 and Figure 38, respectively.

|  |
| --- |
| int PCCQuantization(int value, int quantStep) {  if (!quantStep) {  return value;  }  return sign(value) \* ((abs(value) + quantStep / 3) / qs);  } |

Figure 37: Attribute prediction residuals quantization procedure

|  |
| --- |
| int PCCInverseQuantization(int value, int quantStep) {  return qs == 0 ? value : (value \* qs);  } |

Figure 38: Attribute prediction residuals inverse quantization procedure

## Attribute coding (Lifting Transform)

The Lifting Transform builds on top of the Predicting Transform described in Section 3.6. Figure 39 and Figure 40 describe the direct/forward and inverse transforms in the proposed lifting scheme, respectively. The two main differences between the prediction scheme described in Section 3.6 and the lifting scheme that will be described in the current section, are the following:

1. Introduction of an update operator
2. Use of an adaptive quantization strategy.



Figure 39: Direct/forward transform in the lifting scheme

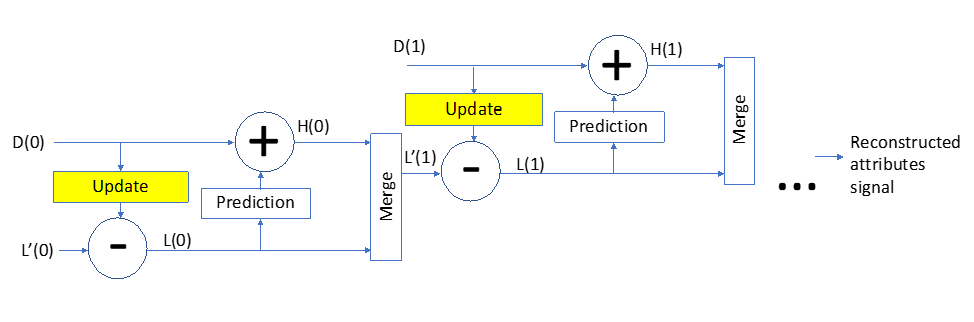


Figure 40: Inverse transform in the lifting scheme

### Update operator

The LOD-based prediction strategy described in Section 3.6 makes points in lower LODs more influential since they are used more often for prediction. Let be the influence weight associated with a point P. Then is computed by applying the following recursive procedure:

* Set for all points
* Traverse the points according to the inverse of the order defined by the LOD structure
* For every point , update the weights of its neighbors as follows:

.

The update operator uses the prediction residuals to update the attribute values of LOD(j). More precisely, let be the set of points such that . The update operation for P is defined as follows:

### Adaptive quantization

The influence weights computed during the transform process are leveraged in order to guide the quantization process. More precisely, the coefficients associated with a point P are multiplied by a factor of . An inverse scaling process by the same factor is applied after inverse quantization on the decoder side. Please note that the scaling factors are completely determined by the reconstructed geometry and they do not need to be encoded in the bitstream.

### Fixed-point implementation [32]

A fixed-point version of the Lifting and the Prediction schemes was introduced as follows:

* Floating-point arithmetic was replaced with fixed-point arithmetic
* An approximated version of the square-root function used
* (LUT-based division approximation was used to avoid expensive divisions needed to compute the prediction weights, the update weights and the quantized/unquantized coefficients.) 🡪 not yet
* The update operation was updated as follows in order to reduce the bit depth needed for fixed-point arithmetic:

## Attribute coding (RAHT)

### Transform coding

The voxel colours , are transform coded, analogously to a colour image, by a spatial transform, quantizer, and entropy coder.

### Spatial transform

The colours are spatially transformed with RAHT [5][6] to obtain transformed colours , . Appendix B provides details of how to obtain the transformed colours , , from the voxel colours , given a list of associated voxel locations , as side information.

### Quantization

The transformed coordinates are quantized by a uniform scalar quantizer with stepsize quantizationStepLuma to obtain the quantized transform coordinates , . The same stepsize is used for all colour components. The quantizationStepLuma is communicated to the colour decoder through the bitstream header.

### Fixed-point RAHT implementation [40][41]

An alternative implementation of the RAHT was introduced. In particular, this method provided a fixed-point implementation of RAHT, which is described in more detail in the attached paper [39]. It has no floating-point operations, and the resulting compression performance is essentially identical to the original floating-point-based transform.

In summary, the transform used in RAHT is as illustrated in Figure 41(a), and its respective inverse transform as in Figure 41(b), where and , and *w*0 is the notation of the weight of the input coefficient *Fl*+1,2*n* while *w*1 is the same for *Fl*+1,2*n*+1.

This method simplify the transform to be like in Figure 41(c) (and its inverse in Figure 41(d)), where  is a fixed-point approximation of *b*2. The coefficients, however, are no longer *Fl*,*n* and *Gl*,*n* but scaled versions of the original coefficients. For that, one needs to adjust the quantizer step. If the quantizer step is , the overall steps for DC and AC coefficients are:

, ,

where *wDC* is the weight of the DC coefficient, or the number of all occupied voxels in the cloud. In order to only use fixed-point values, it needs to approximate  in fixed point and approximate the square root operations using only fixed-point arithmetic.

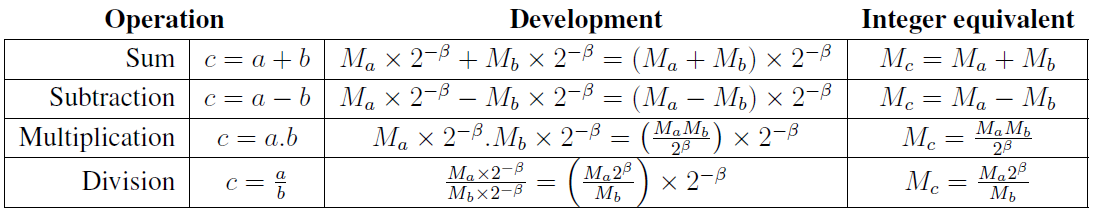


Figure 41: RAHT forward and inverse transforms

It can represent a natural number *a* with fixed-point and  bits of precision as:

*a* = Ma 2-,

where Ma is an integer value. Let us assume two other numbers, *b* = Mb 2- and *c* = Mc 2-. The four basic operations can be performed as:



where the multiplication and division by powers of two can be performed by a binary shift in integer arithmetic.

Another operation required by RAHT is the square root operation. The square root is also computed with integer arithmetic using two basic steps: an initial guess and two iterations of the Newton-Raphson algorithm for refinement:

1. Initial guess
   1. The integer input number, represented using 64 bits, is segmented into 8 bins with 8 bits each, according to Figure 1.

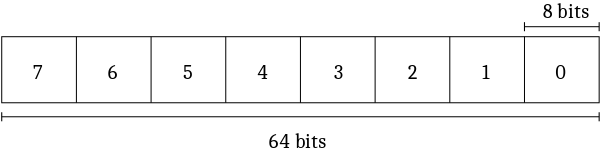


Figure 42: 64-bit segmentation

* 1. The number is right-shifted 8 by 8 bits until only the first bin has non-zero bits.
  2. At this point, we can guarantee that the resulting integer is smaller than 256.
  3. This value is used as an index to a previously computed table that stores integer approximate square roots values of numbers from 0 to 255.
  4. To compensate for the right-shifts, the value returned by the table is left-shifted half the number of bits of the right -shift.

1. Newton-Raphson refinement.

static uint32\_t \_\_tableSqrt[256] = {  
 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4,  
 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6,  
 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,  
 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9,  
 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10,  
 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11,  
 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,  
 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,  
 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,  
 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14,  
 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,  
 14, 14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,  
 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16,  
 16, 16, 16, 16, 16, 16, 16, 16, 16};  
  
uint32\_t sqrtFixedpoint(uint64\_t P)  
{  
 uint32\_t A;  
  
 // Initial guess  
 if( P >= ((int64\_t) 0x1<<32) )  
 {  
 if( P >= ((int64\_t) 0x1<<48) )  
 {  
 if( P >= ((int64\_t) 0x1<<56) )  
 A = (\_\_tableSqrt[P>>56]<<28) - 1;  
 else  
 A = \_\_tableSqrt[P>>48]<<24;  
 }  
 else  
 {  
 if( P >= ((int64\_t) 0x1<<40) )  
 A = \_\_tableSqrt[P>>40]<<20;  
 else  
 A = \_\_tableSqrt[P>>32]<<16;  
 }  
 }  
 else  
 {  
 if( P >= ((int64\_t) 0x1<<16) )  
 {  
 if( P >= ((int64\_t) 0x1<<24) )  
 A = \_\_tableSqrt[P>>24]<<12;  
 else  
 A = \_\_tableSqrt[P>>16]<< 8;  
 }  
 else  
 {  
 if( P >= ((int64\_t) 0x1<< 8) )  
 A = \_\_tableSqrt[P>> 8]<< 4;  
 else  
 return \_\_tableSqrt[P];  
 }  
 }  
  
 A = (A+P/A)>>1;  
 return (A+P/A+1)>>1;  
}

### Upsampled transform domain prediction in RAHT [49]

The transform domain prediction is introduced to improve coding efficiency on RAHT. It is formed of two parts.

Firstly, the RAHT tree traversal is changed to be descent based from the previous ascent approach; ie a tree of attribute and weight sums is constructed and then RAHT is performed from the root of the tree to the leaves for both the encoder and the decoder. The transform is also performed in units of 2×2×2 blocks, rather than breadth-first method. Within the block, the encoder transform order is from leaves to the root. The inverse transformed coefficients of one 2×2×2 block become the inherited DC coefficients of the next level.

Secondly, for each 2×2×2 block, a predicted block is produced by upconverting the previous transform level.

The prediction is transformed and subtracted from the transformed attributes at the encoder.

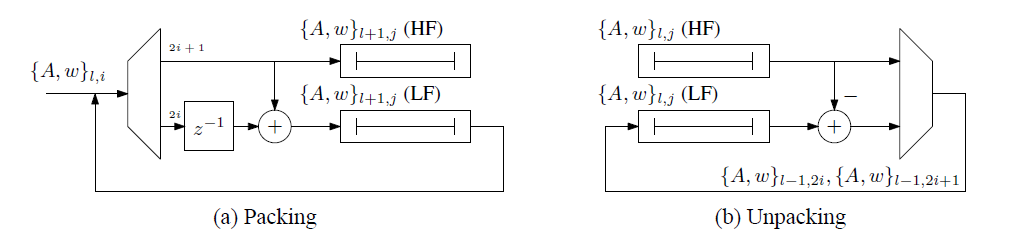


Figure : Construction of the transform tree levels

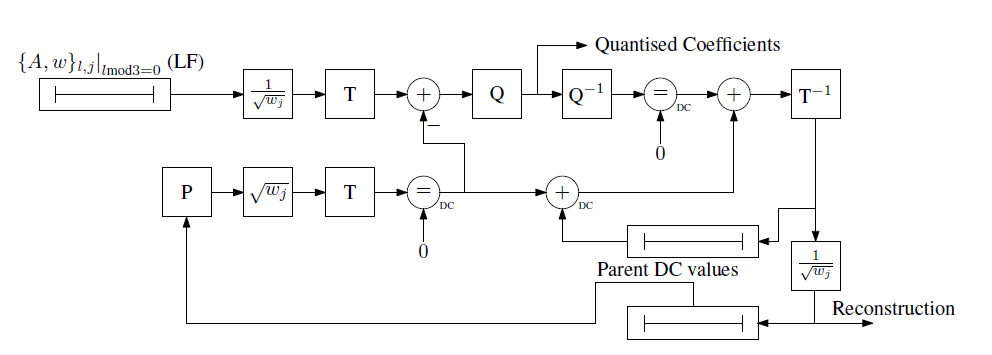


Figure : Transform and prediction process overview

Figure 46 shows the principle on the upsampled transform domain prediction and the detail processes on prediction, residual calculation, decoding and up-sampling are shown in Figure 47, Figure 48, Figure 49 and Figure 50 respectively.

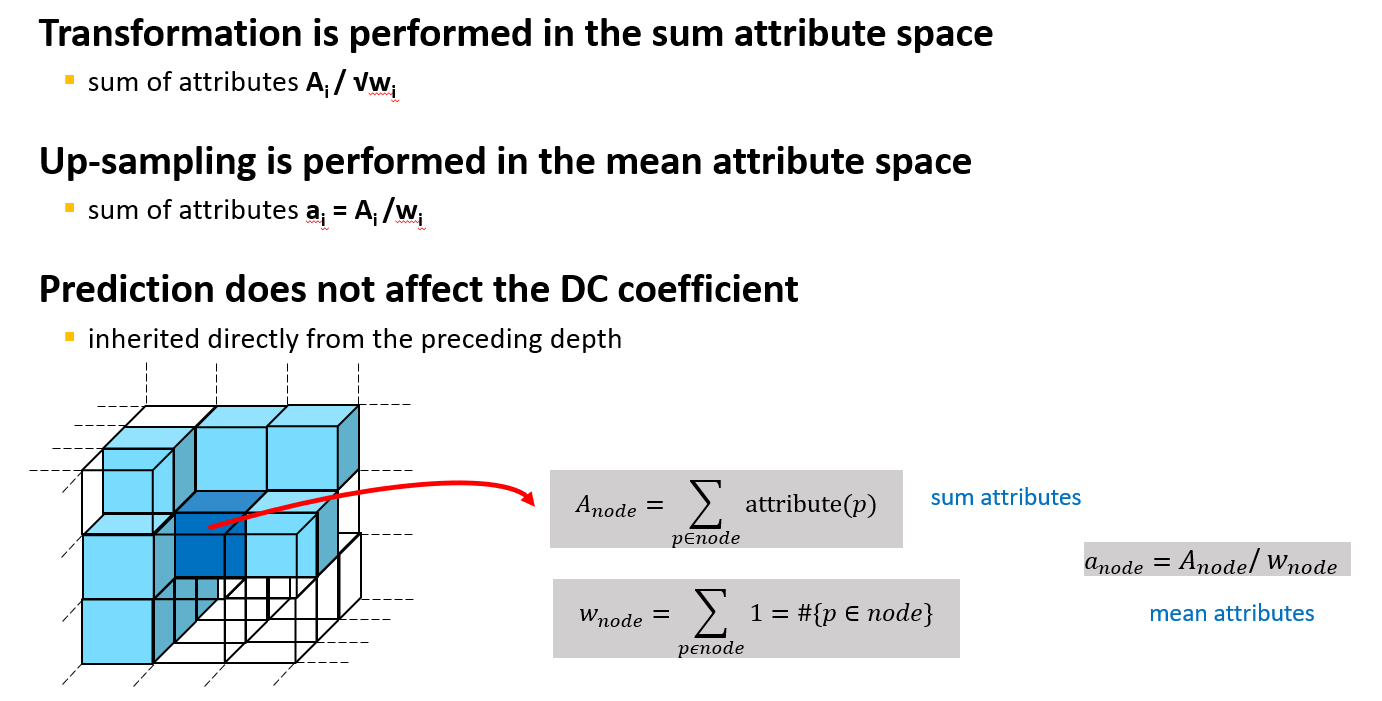


Figure : Principle on prediction

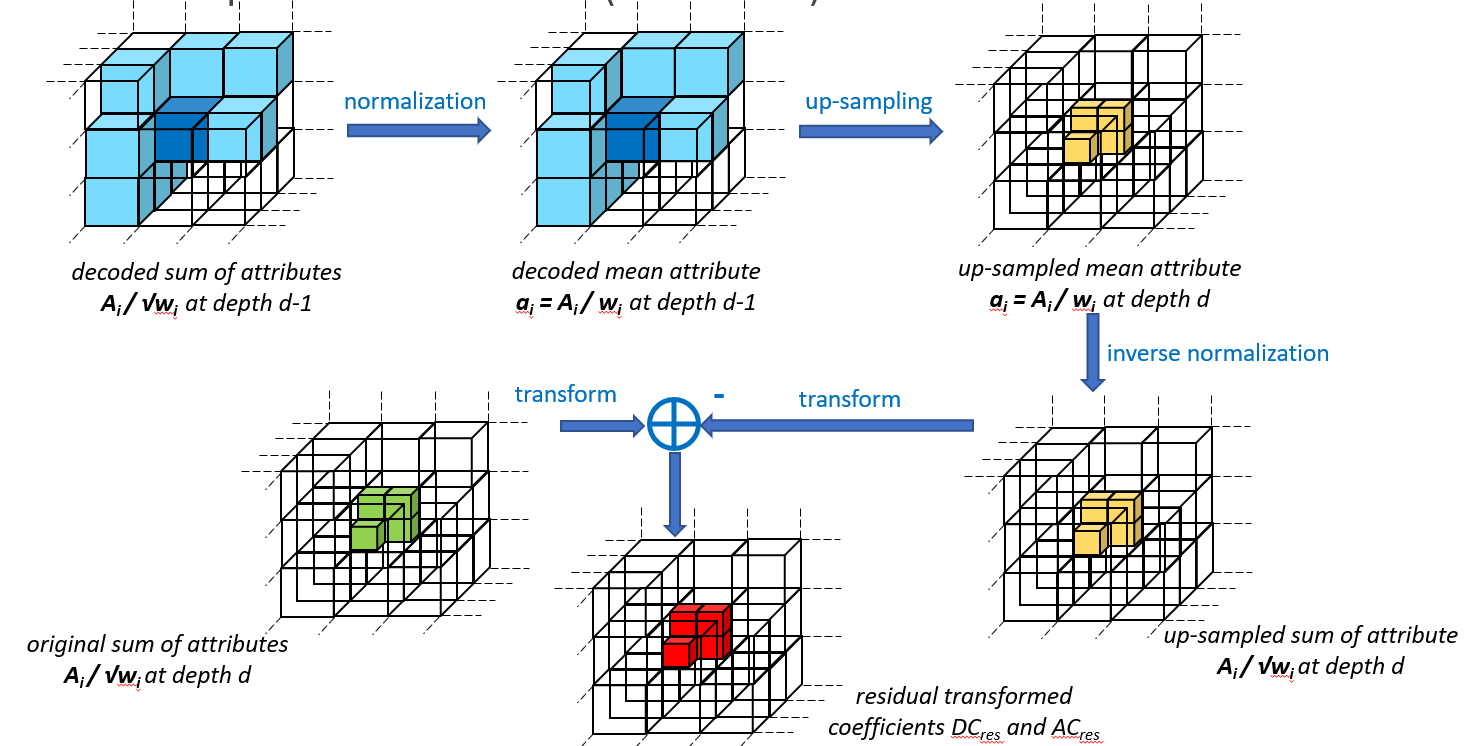


Figure : Prediction per 2x2x2 (encoder)

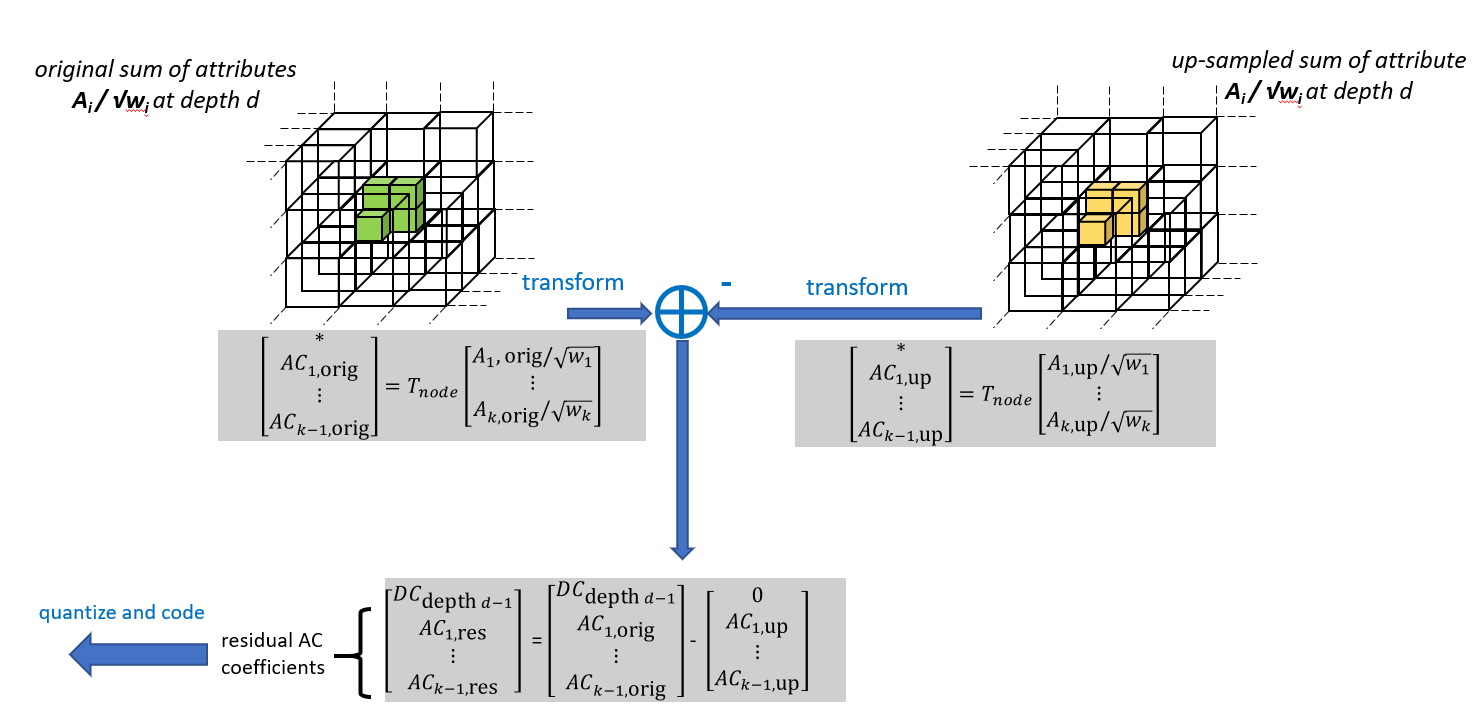


Figure : Computing the residual

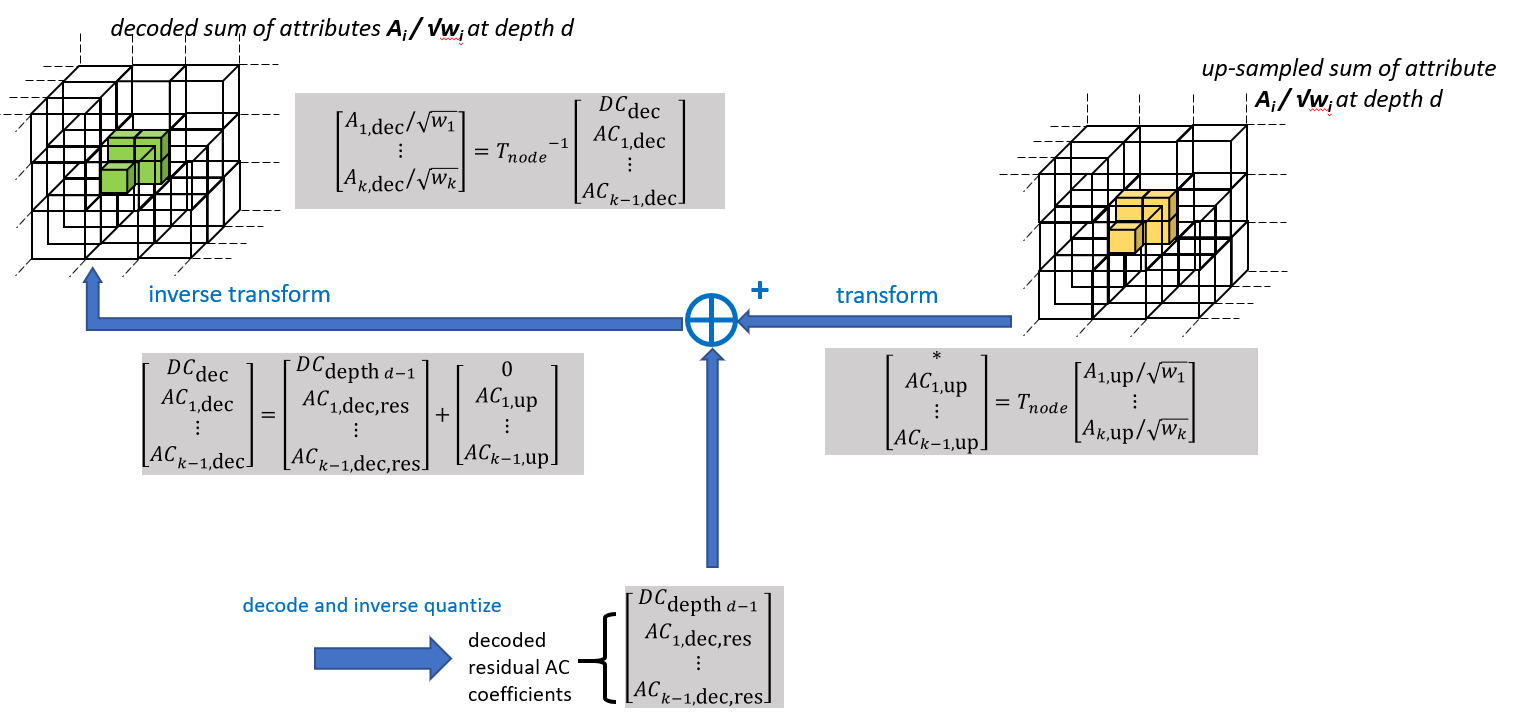


Figure : Decoding process

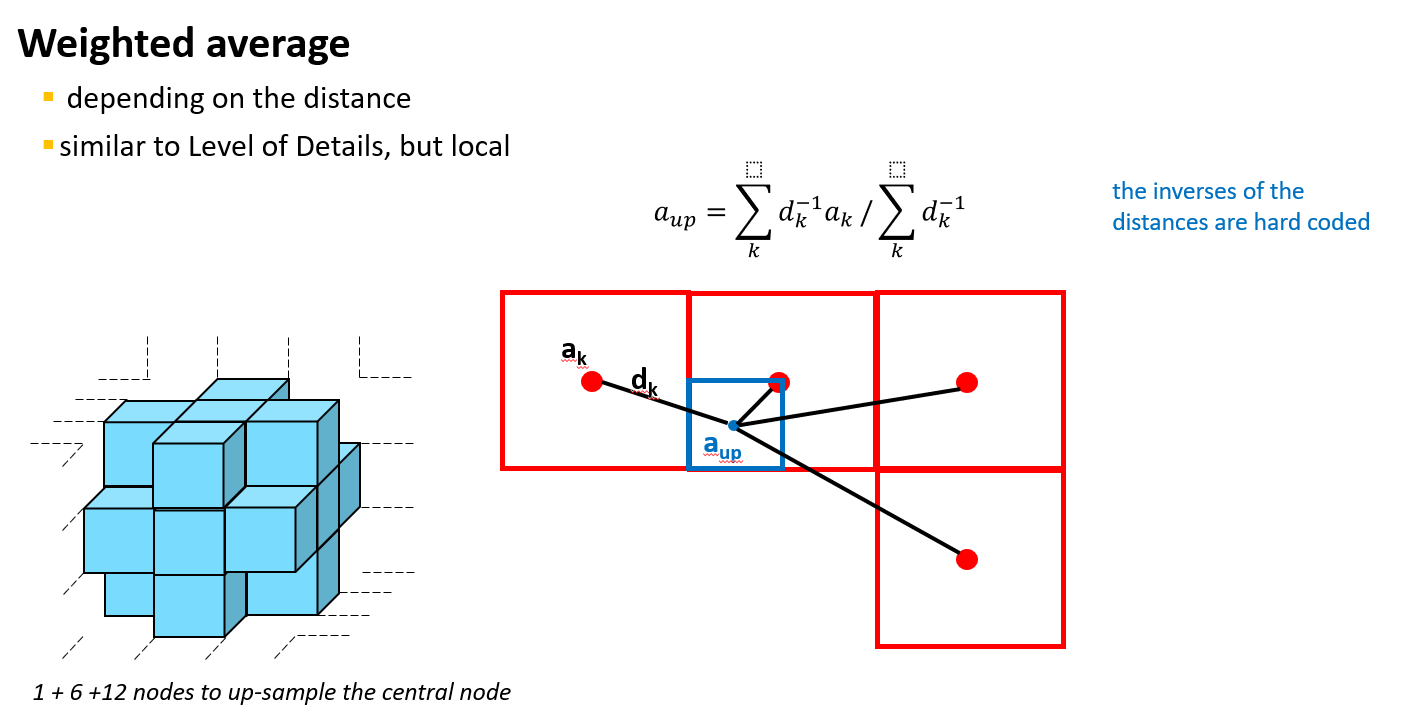


Figure : Up-sampling process

## Attribute Quantization

### Quantization table [35][36]

The quantization of lifting and RAHT transformed residual coefficients is performed by input signalled QP parameter. This is like the traditional AVC/HEVC/VVC QP based quantizer design whereas a QP parameter is used to determine the quantizer step size. QP parameter is based on a logarithmic structure with every increase of 1 in QP value quantizer step size is increased by approximately 12% and an increase by 6 results in an increase by a factor of 2.

(1)

= (2)

It is clear that to derive the quantizer step sizes of an input signal it is only necessary to store the 1st 6 values of quantizer step size as a function of QP, namely QP = 0 to 5. Accordingly, QP parameter based formulation and derivation is shown below:

(3) where

<< 8 =

(4)

The scaling factor is in accordance with the fixed-point scaling factor introduced in the fixed point RAHT implementation. It also provides enough granularity when changing the QP values.

QP table is shared in three attribute encoding methods: Predicting, Lifting, and RAHT. The decoding processing is modified, because QP table has the function of multiplication by 28. This can be said as quantization and left shift processing are merged as QP table.

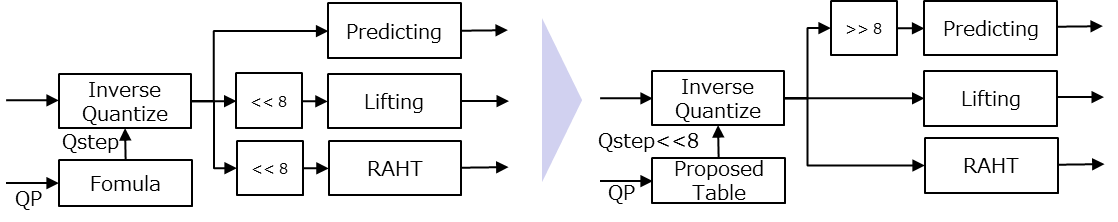


Figure 51 Modification of the Decoder process TMC13v5

### LUT-based quantization [47]

The general idea is to perform quantization using a forward quantization table and de-quantization using an inverse quantization table, as illustrated in the following block diagram.

QP

fLUT

gLUT

Derive InvQstep

Derive ForQstep

De-quantization

Quantization

quantized

value

value

reconstructed value

Illustration of LUT-based quantization and

De-quantization.

Given a quantization parameter QP, a forward and an inverse quantization step are derived (ForQstep and InvQstep) based on two different quantization tables (forward *fLUT* and inverse *gLUT*). The input value is quantized using ForQStep and dequantized using InvQstep. This scheme allows for the removal of the division operation in the quantization process, as will be described.

Consider the quantization table *gLUT*. This table is defined in previous section. The forward quantization table *fLUT* is derived as follows.

Currently N = 8. And it is considered M = 14, since it was enough to achieve the desired precision. In this case, *fLUT* becomes,

### Adaptive quantization scheme for RAHT

Given a dead-zone , a RAHT coefficient with weight is quantized as:

|  |
| --- |
| // increase the dead-zone by multiples of step-size |
|  |
|  |
| // apply default quantizer as in TMC13 |
|  |

where is a parameter chosen to differentiate between the low-frequency and the high frequency coefficients.

The inverse quantization is done by:

## Attribute Redisual and Entropy encoding

The quantized, transformed coefficients are entropy encoded using zero run length coding and an arithmetic coder.

### Attribute residual coding [37]

Zero run length coding for attribute residual value and a flag “isOne” which indicates if residual value is equal to 1 based on the current residual coding specification were introduced. In zero run length coding, the number of zero prior to each residual value is counted as zerorun, and then zerorun is encoded instead of encoding sequence of 0s. Figure 40 shows how to encode residual value with zero run length and isOne flag. A zerorun parameter is encoded by truncated unary code with 3 contexts, and a isOne flag is encoded with 7 contexts by using same way to encode isZero flag. This method is applied not only Predicting Transform and Lifting Transform but also RAHT.

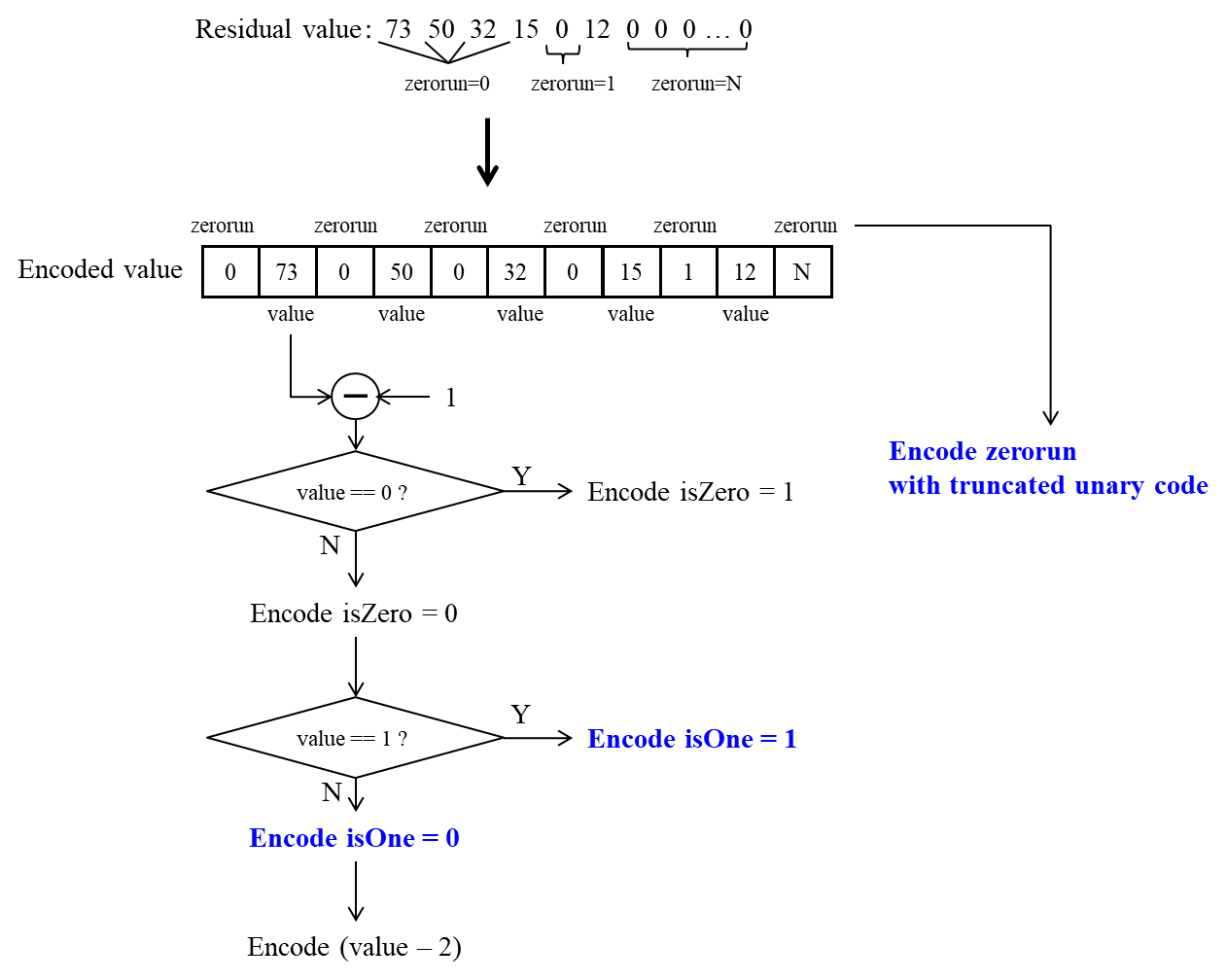


Figure 52: encoding flow for residual value

### Bytewise based binary coding for transformed coefficients [18]

G-PCC supports an efficient binarization scheme for transform coefficients, which requires only a binary arithmetic encoder making it more HW-friendly in terms of implementation. It leverages the binarization scheme described in section 0. First, the binarization approach for mono-dimensional attribute signals such reflectance is described. Next, the case of three-dimensional signal such as clours is described. Generalization to multi-dimensional attributes is straightforward.

#### Mono-dimensional attributes

Let C be the quantized coefficient to be encoded. First C is mapped to a positive number using the function described in Figure 53. Let M(C) be the mapped value. A binary value is then encoded to indicate whether M(C) is 0 or not. If C is not zero, then two cases are distinguished:

* If M(C) is higher or equal than alphabetSize (i.e., 256 the number of symbols supported by the technique described in [2]), then the value alphabetSize is encoded by using the method described in section 0. The difference between M(C) and alphabetSize is encoded by using an exponential Golomb coding
* Otherwise, the value of M(C) is encoded using the method described in 0.

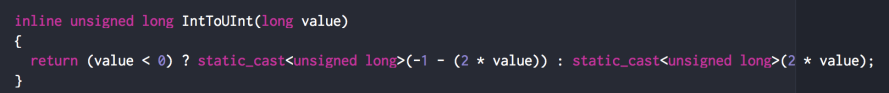


Figure 53: mapping signed integers to unsigned integers

#### Three-dimensional attributes

Let C1, C2, and C3 be the quantized coefficients to be encoded. First C1, C2 and C3 are mapped to a positive number as described above. Let M(C1), M(C2) and M(C3) be the mapped values.

M(C1) is encoded as described above. M(C2) is encoded as described above while choosing different contexts (see Figure 54) based on the condition M(C1) is zero or not . M(C3) is encoded as described above, while choosing different contexts (see Figure 54) based on the conditions M(C1) is zero or not and M(C2) is zero or not.

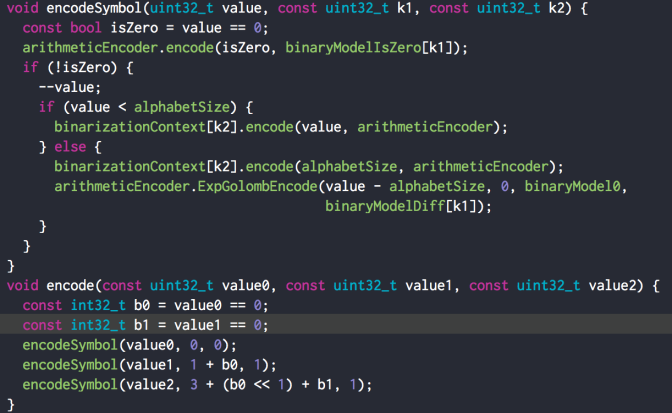


Figure 54: Binarization of a three-dimensional signal

## Functionality

### Slice and Tile partition process [43]

Figure 55 shows the process of slice and tile partition.

a) the original point cloud would be firstly quantized according to corresponding condition.

b) Then the quantized cloud is splitted into tiles, which are several cube areas with a certain side le*ngth TileSize*.

c) After that, the process of slice partition contains two steps.

First, do the preliminary slice partition with slice partition schemes in TMC13. Second, based on two parameters *MaxPointNum* and *MinPointNum*, do further merging and splitting operations on slices after first step, which try to get slices with suitable number of points. The modification on partition and quantization order could ensure the point number of each slice to meet the requirement.

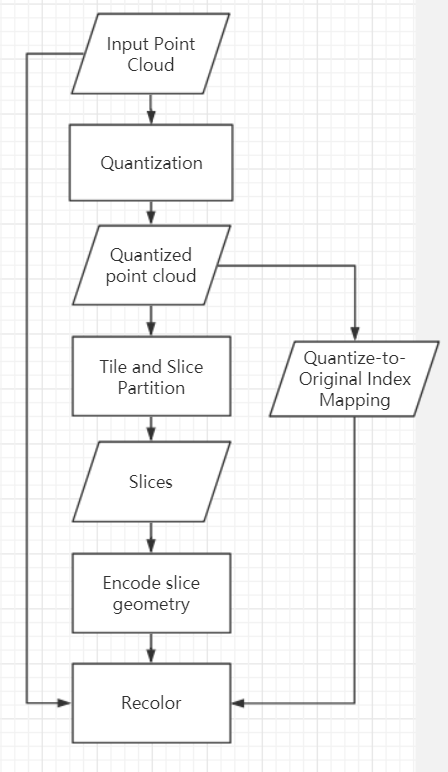


Figure Slice Partition process

### Tile partition schemes [43]

The tile partition schemes is as follows:

* Let tileMaps be a map of tile IDs to point indexes that correspond to the tile, initially empty.
* For each point in the reconstructed point cloud after quantization, determine the tile to which it belongs:

- Let tile\_origin = floor(pos / *TileSize*)

- Append point index to tileMaps[tile\_origin]

* Then the tile ID is simply the index into the tileMaps
* Do slice partition in each tile.

### Slice partition schemes [25] [43] [44]

The process of slice partition contains two steps. First, do the preliminary slice partition with slice partition schemes. Second, based on two parameters MaxPointNum and MinPointNum, do further merging and splitting operations on slices after first step, which try to get slices with suitable number of points. The method could adaptively determine the number of slices by datasets without per-sequence configurable parameters.

If the number of points is more than MaxPointNum, the following slice partition is done, otherwise the whole point cloud is compressed directly without partition.

TMC13 software supports two slice partition schemes in the encoder.

1. Uniform-Geometry partition along the longest edge
2. Assume the longest edge and the shortest edge as *maxEdge* and *minEdge*, slice number as *sliceNum*, slice size as *sliceSize*. The default value of *sliceNum* is set as *maxEdge* / *minEdge* and *sliceSize* is set as *minEdge*.
3. Clear *partitions.slices* and resize it to *sliceSize*. Divide the point cloud into *sliceNum* slices by uniform-Geometry partition scheme.
   * Evaluate the proportion of points smaller than the maxPointNum in all slices. Set a ratio threshold . If is bigger than , go to next step. Otherwise, back to procedure 1) and double the *sliceNum*.



Figure 56: Uniform-Geometry partition along the longest edge

When trisoup is used, first check whether the original slice partition interval is an integer multiple of the block size . If not, the partition interval will be rounded up to the nearest integer that is exactly divisible by the block size.

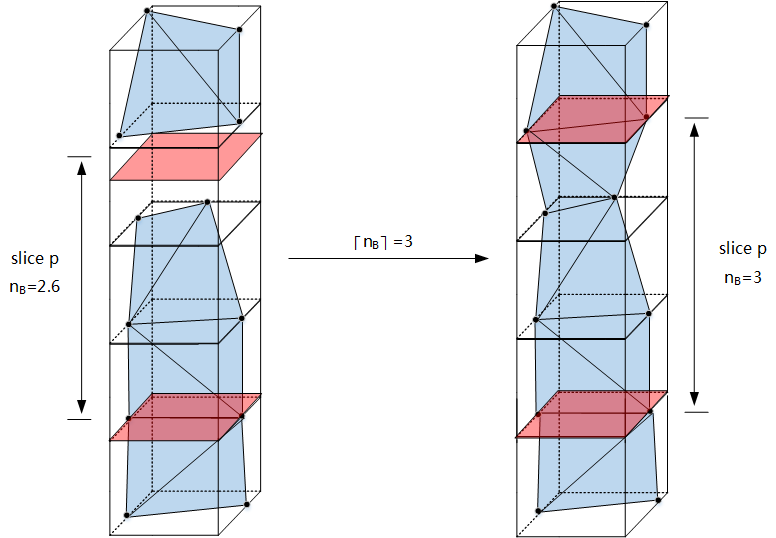


Figure : Partition interval when using trisoup

1. Uniform-Geometry partition using Octree
2. Set the depth of octree partition *depOctree* = 1 by default
3. Divide the input point cloud into 8depOctree slices by Octree partition scheme.
   * Evaluate the proportion of points smaller than the maxPointNum in all slices. Set a ratio threshold . If is bigger than , go to next step. Otherwise, back to procedure 1) and set *depOctree* += 1.



Figure 58: Uniform-Geometry partition using Octree

### Merging and splitting operation [43]

After slice partition, the point cloud of slice whose point count is more than *MaxPointNum* is splitted, and the point cloud of slice whose point count too few points is merged.

* + Split: If the point count (*Asize*) is more than M*axPointNum*, split the slice into *n* partitions, where *n* = ceil (*Asize* / *MaxPointNum*).
  + Merge: If the point count of current slice is less than *MinPointNum*, merge it with either prior slice or next one. The principles of choosing the direction to merge are as follows:

1. If current slice is in the first place, then the direction of merging is:

merge -> next slice

1. If current slice is at the end, then the direction of merging is:

merge -> prior slice

1. If current slice is neither the first nor the last one, then assume the point count of slice as *SumFront* and *SumNext* after merging with prior slice and next one respectively.

* If *SumFront* > *MaxPointNum* and *SumNext* > *MaxPointNum*,

or *SumFront*< *MaxPointNum* and *SumNext* < *MaxPointNum*,

then choose the merging direction with the one with more points.

* Otherwise, that is one of *SumFront* and *SumNext* is larger than *MaxPointNum*

and another is not, then choose the merging direction with the smaller one.

After merging, traverse all slices generated after merging, assuming the point count of merged slice as *SumMerged*, compare the *SumMerged* with *MaxPointNum*.

* + If *SumMerged* < *MinPointNum*, keep merging for current slice.
  + If *SumMerged* > *MaxPointNum*, split the merged slice with the interval as *MaxPointNum*.

If *MinPointNum <SumMerged< MaxPointNum,* keep current slice and check next slice.

### Header reduction by quantize minimum position using gsh\_box\_log2\_scale [30]

Minimum position value of the slice is quantized by gsh\_box\_log2\_scale bit shifting.

Slice origin is set to quantized minimum position.

In octree partition method, gsh\_box\_log2\_scale of every slice origin is set as below.

* gsh\_box\_log2\_scale = total\_bit – octree\_depth

By setting above way, shifted origin value become equal to Morton order of octree partition.

When the same gsh\_box\_log2\_scale is set to all slices, it is not necessary to store them in slice headers, so common log2\_scale is set to new syntax in GPS and it is indicated by a flag whether it is stored in GPS or geometry slice header.

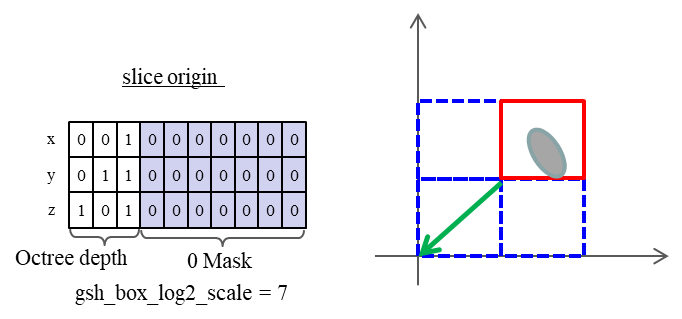


Figure 59: Header reduction by quantize position and scaling

### Slice-based control using Quantization Step delta [34]

The quantization parameters (QStepAPS) are stored in the Attribute Parameter Set (APS), and the parameters can be changed slice-by-slice.

By adding the Q step delta value (, ) in each attribute slice header, the Q step value can be changed for each slice. Q step delta value is the difference between QStepAPS and Q step value of each slice. Q step delta is present only if slice\_quant\_step\_present\_flag in APS is set to 1.

In the decoder, obtain QStepAPS from APS and Q step delta value from each attribute slice header, and Q step value can be calculated.

In the current specification, Qstep delta was replaced with QP delta.

As1

APS

Gs1

GPS

As2

Gs2

QStepslice1 = QStepAPS + ΔQStepslice1

QStepslice2 = QStepAPS + ΔQStepslice2

QStepAPS

ΔQStepslice1 ＝ QStepslice1 - QStepAPS

ΔQStepslice2 ＝ QStepslice2 - QStepAPS

Figure 60: The modified bitstream structure to support delta Qstep

### Attribute Layer Quantization Control using Delta QP Layer [42]

The attribute quantization parameter of each layer can be controlled by using this method. A delta QP layer parameter is implemented on each attribute slice header.

Figure 61 shows Delta QP layer implementation on Lifting/Predicting transform and specific delta QP is applied on each layer. The effective QP value for each layer in a particular slice is added between delta QP of that layer with QP for that slice. The final residue value will be the product of the attribute with quantization weight and effective QP value.

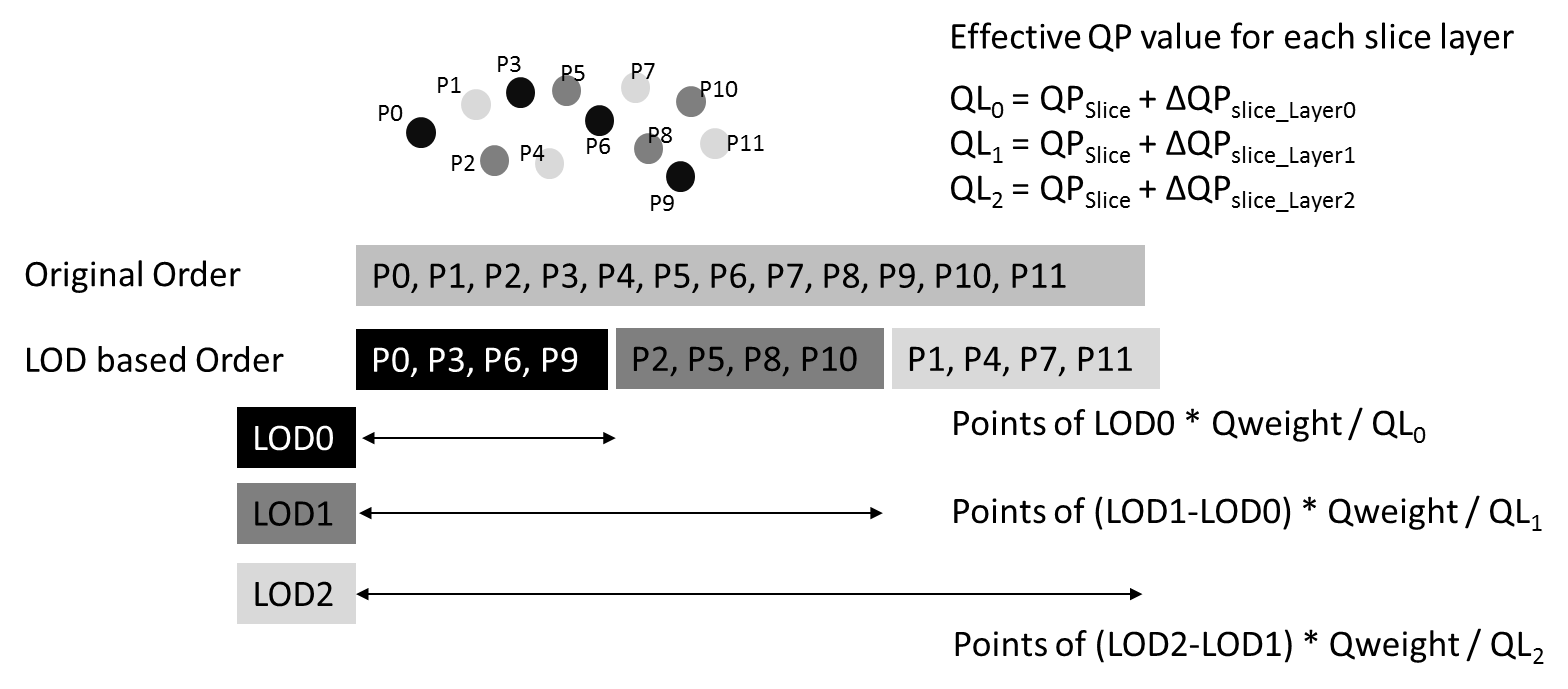


Figure 61 Operation of Delta Layer QP for Predicting/Lifting Transform

Figure 62 shows RAHT with delta QP for each layer and specific delta QP is applied on each layer. The effective QP value for each layer in a particular slice is added between delta QP of that layer with QP for that slice. The final residue value will be the square root of the attribute divided by weight and effective QP value.

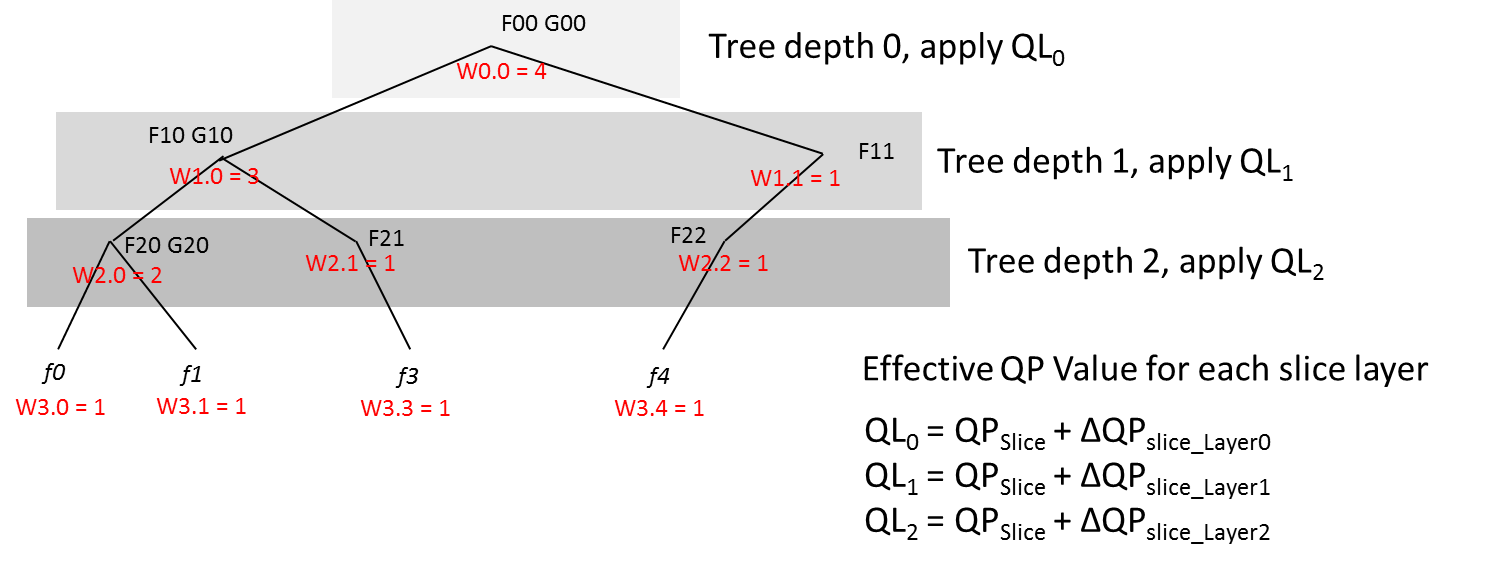
****

Figure 62 Operation of Delta Layer QP for RAHT

The quantization parameter (QPAPS) are stored in the Attribute Parameter Set (APS), and the parameters can be changed slice-by-slice and further layer by layer.

By adding the delta QP layer value (, …) in each attribute slice header, the QP value can be changed for each layer of a particular. Delta QP layer value is the difference between QPslice and effective QP value of that slice’s layer. Delta QP layer is present only if layer\_QP\_present\_flag in attribute slice header ASH is set to 1.

As1

APS

Gs1

GPS

As2

Gs2

QPslice1 = QPAPS + ΔQPslice1

QPslice2 = QPAPS + ΔQPslice2

QPfinal\_slice1\_LoD1 = QPSlice1 + ΔQPslice1\_Layer1

QPfinal\_slice1\_LoD2 = QPSlice1 + ΔQPslice1\_Layer2

QPfinal\_slice2\_LoD1 = QPSlice2 + ΔQPslice2\_Layer1

QPAPS

ΔQPslice1 ＝ QPslice1 - QPAPS

ΔQPslice1\_Layer1 ΔQPslice1\_Layer2 ΔQPslice1\_Layer3 ….

ΔQPslice2 ＝ QPslice2 - QPAPS

ΔQPslice1\_Layer1 ΔQPslice1\_Layer2 ΔQPslice1\_Layer3 ….

Figure 63: The modified bitstream structure to support layer delta QP

### Spatial scalability support [46]

The spatial scalability is important functionality for G-PCC. It is especially useful when the source point cloud is dense even in the local area as the Level of Detail (or, the octree depth) should be large enough to represent the original quality. With the spatial scalability, one can access a lower resolution point cloud as a thumbnail with less decoder complexity and/or with less bandwidth.

When the spatial scalability is needed, it is desirable to decode lower geometry and the corresponding attribute bitstream in a harmonized way as shown in Figure 64. Without the attribute (especially without the color attribute), it is difficult to understand the content at a glance.

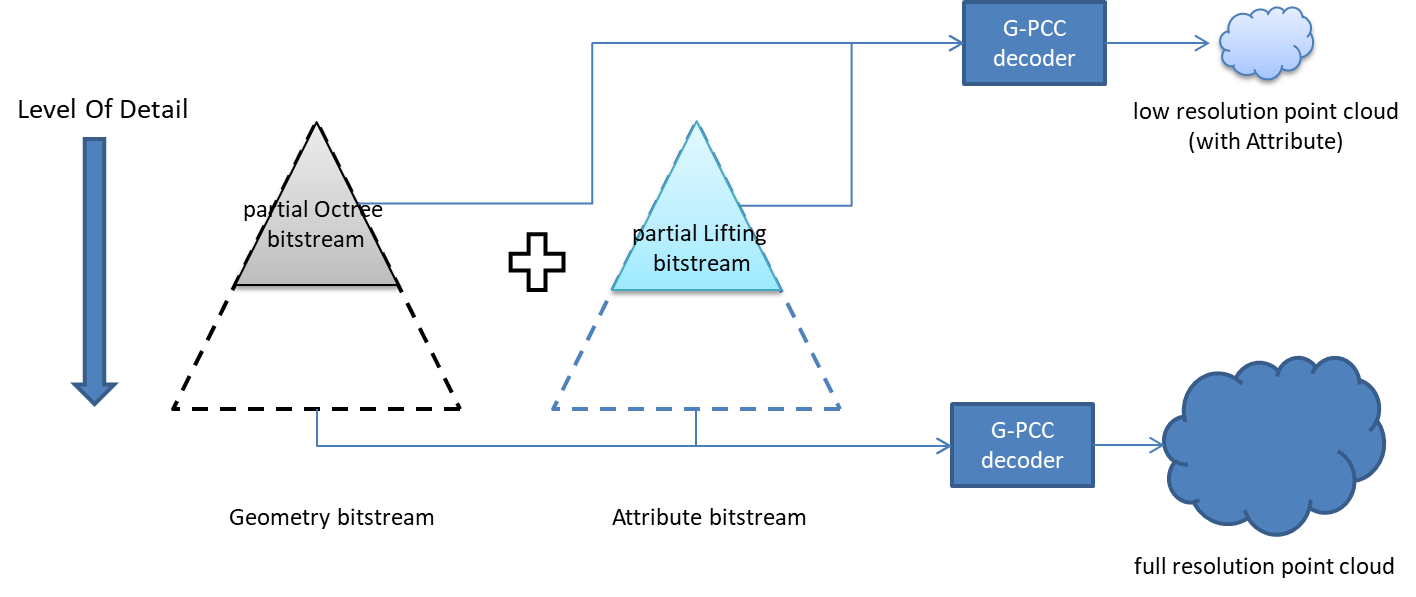


Figure 64: Extended geometry & attribute harmonized spatial scalability

To achieve the harmonized spatial scalability, the attribute decoder is extended to allow the lower resolution geometry point cloud from the partially decoded Octree bitstream, where the position is quantized as INT(pos/2k)\*2k.

The following subsections describe the extended changes for the Lifting scheme to support spatial scalability.

#### Octree harmonized LoD construction

The LoD generation process is extended to align with the geometry Octree structure.

Figure 65 and Figure 66 show the normal and the extended LoD generation process, respectively.

In the normal process, the distance between points is considered. In the extended one, the Octree structure is considered to separate the grouping. The motivation is to align the number of points for the geometry and the attribute.

By this, when the partial octree decoded point cloud is given to the attribute decoder, it can be constructed the Lifting LoD from the given level in the decoder side correctly.

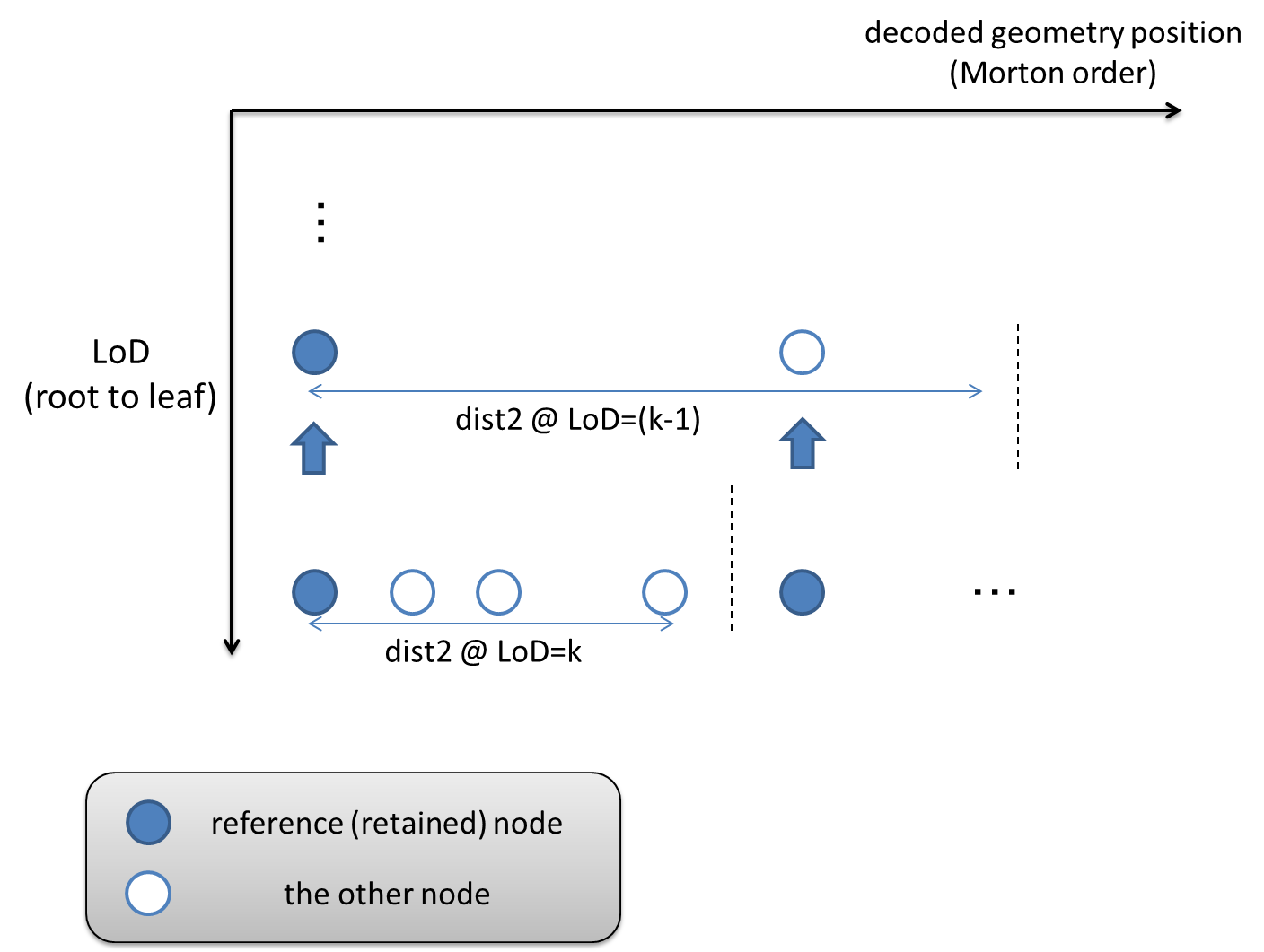


Figure 65: Normal LoD generation (Independent from Octree LoD)

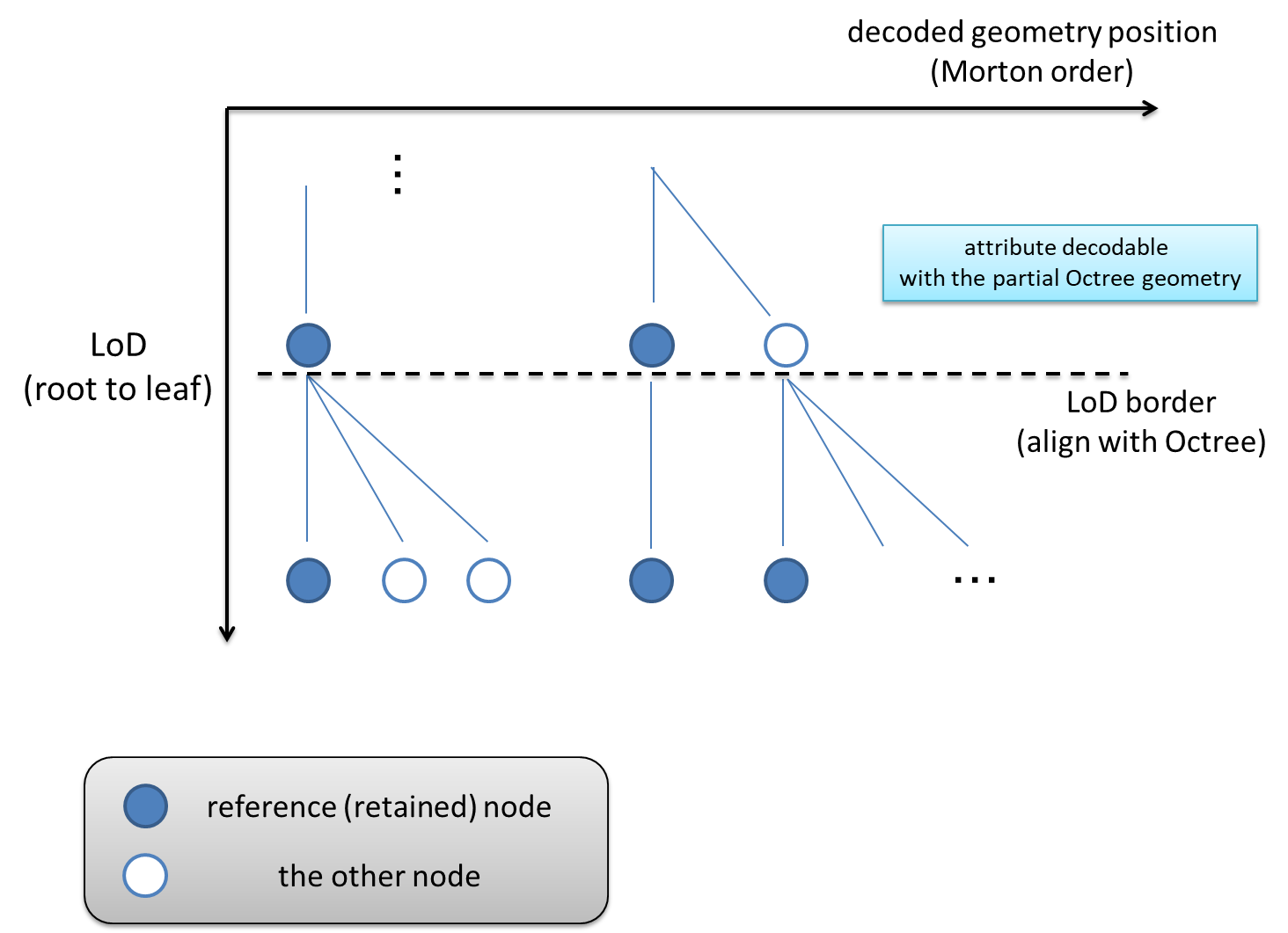


Figure 66: Extended LoD generation (align with Octree LoD)

#### Weight derivation

In the normal process, the weight value is derived from the sum of distances in the higher LoD level. When the partial octree decoded point cloud is given to the attribute decoder, the decoder cannot calculate the correct weight (because the lower LoD information is unknown).

In the extended one, instead of deriving from the distance, the weight is fixed by the point number in the LoD.

Wextended = (Total point number) / (Point number in current LOD)

This simplification assumes that the point distribution in the higher LoD is equally dense enough for the content where the spatial scalability is needed.

#### Distance normalization

Related to the previous subsection, the weight value is derived from the normalized distance based on LoD.

In the normal process, the distance is calculated as follows.

Dist = ( **P**ref − **P**i )2

, where **P** denotes the x,y,z position for the ref and the i

In the extended one, the distance is calculated as follows.

Dist = ( Quant(**P**ref, LoD) − Quant(**P**i) )2

, where Q(x, LoD) is defiend as ((x>>LoD)<<LoD).

With the normalization, the decoder can calculate the correct weight value with the partially decoded geometry.

### Combine Frame Coding [24]

In frame-based point cloud content, each frame may be relatively smaller in file size, which is less efficient for the I/O interface. Another issue is the overhead of initializing decoder becomes more significant in the edge device as well. The decoder need to run from the initial bounding box and do the dividing for each single frame, but for combine coding the process is conducted only once for each combined Group of Point cloud (GOP), which could be beneficial to less powerful devices.

The first issue could be easily addressed by concatenating the encoded bit-stream of consecutive frames. The second one, however, is inevitable unless the point clouds are combined before encoding. The proposed combine frame coding addresses both issues by introducing the encoding of frame index in the combined point cloud. Moreover, it improves coding efficiency largely so that it could be also beneficial for storage usage of frame-based point cloud content.

Shown below, frames from frame-based point cloud data, for example, Ford content, are combined to a single point cloud. In the Ford content, depending on the movement of the vehicle, some parts of the frame appear to be stationary, while others have moved.

When each of point cloud source is correlated to one another, individual Octree of each frame has a similar structure in the higher level.

In the leaf node of the combined frame, there are some duplicated points that of different frames.

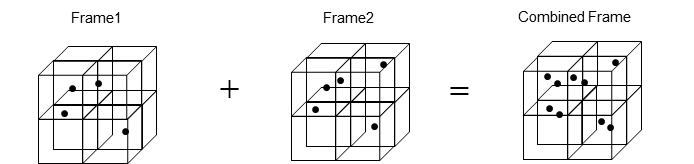


Figure 67: Image of Combine Point Cloud Input Data

A single bitstream is outputted after encoding the entire sequence. Group of Point cloud (GOP) is created and each frame inside the GOP is assigned a unique index, which we call the frame index. The frame indices, which are used in the decoder to reconstruct the input frames, are encoded using two different approaches. Frame index can also be encoded as an attribute. A new attribute is defined to represent frame index. The existing attribute coding method is used to encode it. Note the coding of frame index should be lossless in order to reconstruct correctly.

# References

1. FastAC fast arithmetic coding algorithm, <http://www.cipr.rpi.edu/research/SPIHT/>
2. Call for Proposals for Point Cloud Compression V2, ISO/IEC JTC1/SC29/WG11 w16736, Hobart, AU, April 2017.
3. https://cmake.org
4. <https://www.threadingbuildingblocks.org>
5. Queiroz and Chou, “Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform,” IEEE Trans. Image Processing, Aug. 2016.
6. Chou and Queiroz, "Transform Coder for Point Cloud Attributes," ISO/IEC JTC1/SC29/WG11 input document m38674, Geneva, May 2016.
7. [PCC] Inference of a mode using point location direct coding in TMC3, ISO/IEC JTC1/SC29/WG11 input document m42239, Gwangju, Korea, January 2018.
8. [PCC] Neighbour-dependent entropy coding of occupancy patterns in TMC3, ISO/IEC JTC1/SC29/WG11 input document m42238, Gwangju, Korea, January 2018.
9. Look ahead cube for efficient neighbours information retrieval in TMC13, ISO/IEC JTC1/SC29/WG11 input document m43591, Ljubjana, Slovenia, July 2018.
10. PCC Simplification of neighbour-dependent entropy coding in CE3.4, ISO/IEC JTC1/SC29/WG11 input document m42689, San Diego, US, April 2018.
11. [PCC] Intra mode for geometry coding in TMC3, ISO/IEC JTC1/SC29/WG11 input document m43600, Ljubljana, Slovenia, July 2018.
12. [PCC] A new binary entropy coder with update for geometry coding in TM3, ISO/IEC JTC1/SC29/WG11 input document m44750, Macau, China, October 2018.
13. Binarization of occupancy information in TMC13, ISO/IEC JTC1/SC29/WG11 input document m43592, Ljubljana, Slovenia, July 2018.
14. PCC TMC13 alternative entropy codecs, ISO/IEC JTC1/SC29/WG11 input document m43649, Ljubljana, Slovenia, July 2018.
15. “Dirac Specification version 2.2.3,” BBC, Sep. 2008.
16. “VC-2 Video Compression,” ST 2042-1, SMPTE, Nov. 2009.
17. PCC Adaptive predictor selection for attributes coding in TMC13 related to CE13.3, ISO/IEC JTC1/SC29/WG11 input document m43665, Ljubljana, Slovenia, July 2018.
18. Binarization of transform coefficients in TMC13, ISO/IEC JTC1/SC29/WG11 input document m43780, Ljubljana, Slovenia, July 2018.
19. Efficient implementation of the Lifting Scheme in TMC13, ISO/IEC JTC1/SC29/WG11 input document m43781, Ljubljana, Slovenia, July 2018.
20. A new neighbour node occupancy decision, ISO/IEC JTC1/SC29/WG11 input document m44752, Macau, China, October 2018
21. Entropy coding an octree node occupancy depending on neighbour’s child nodes, ISO/IEC JTC1/SC29/WG11 input document m44753, Macau, China, October 2018
22. Improved G-PCC lossless and near-lossless coding, ISO/IEC JTC1/SC29/WG11 input document m44899, Macau, China, October 2018
23. Binary-tree based level-of-details generation for attributes coding in G-PCC, ISO/IEC JTC1/SC29/WG11 input document m44940, Macau, China, October 2018
24. TMC13 new proposal on Combine Frame Coding, ISO/IEC JTC1/SC29/WG11 input document m44813, Macau, China, October 2018
25. G-PCC TMC13 CE13.2 report on point cloud tile and slice based coding, ISO/IEC JTC1/SC29/WG11 input document m45867, Marrakech, Morocco, January 2019
26. PCC CE1.3 recolor method, ISO/IEC JTC1/SC29/WG11 input document m42538, San Diego, US, April 2018
27. [PCC] On further reduction of neighbour configurations, ISO/IEC JTC1/SC29/WG11 input document m46148, Marrakech, Morocco, January 2019
28. [PCC] On an improvement of advanced neighbours, ISO/IEC JTC1/SC29/WG11 input document m46149, Marrakech, Morocco, January 2019
29. [PCC] On the reduction of states related to advanced neighbours in OBUF, ISO/IEC JTC1/SC29/WG11 input document m46150, Marrakech, Morocco, January 2019
30. [G-PCC] CE13.2 Report on geometry slice header reduction in slice partition, ISO/IEC JTC1/SC29/WG11 input document m47398, Geneva, CH, March 2019
31. Reference structure modification on attribute predicting transform in TMC13, ISO/IEC JTC1/SC29 WG11 input document m46107, Marrakech, MA, January 2019
32. [G-PCC] [New proposal] Fixed-point implementation of G-PCC, ISO/IEC JTC1/SC29/WG11 input document m44900, Macau, China, October 2018
33. [G-PCC][New Proposal] Efficient Low-Complexity LOD Generation, ISO/IEC JTC1/SC29 WG11 input document m46188, Marrakech, MA, January 2019
34. [G-PCC] CE13.16 Report on Slice-based quantization control, ISO/IEC JTC1/SC29/WG11 input document m47399, Geneva, CH, March 2019
35. [G-PCC] Quantization Parameter table in Attribute Coding, ISO/IEC JTC1/SC29/WG11 input document m47401, Geneva, CH, March 2019
36. [G-PCC] New contribution on quantization parameter definition, ISO/IEC JTC1/SC29/WG11 input document m47507, Geneva, CH, March 2019
37. [G-PCC] Attribute residual coding in TMC13, ISO/IEC JTC1/SC29 WG11 input document m46108, Marrakech, MA, January 2019
38. [G-PCC][New proposal]Non-normative Subband Adaptive, ISO/IEC JTC1/SC29/WG11 input document m45019, Macau, China, October 2018
39. G. Sandri, P. A. Chou, M. Krivokuća, and R. L. De Queiroz, *Integer Alternative for the Region-Adaptive Hierarchical Transform,* submitted for publication in the IEEE Signal Processing Letters. Copyright held by IEEE.
40. Fixed-point version of RAHT with rate-distortion impact tests, ISO/IEC JTC1/SC29/WG11 input document m44486, Macau, China, October 2018
41. G-PCC CE13.14 Report on implementation of fixed-point RAHT in TMC13, ISO/IEC JTC1/SC29 WG11 input document m46209, Marrakech, MA, January 2019
42. [G-PCC] CE13.19 Report on Attribute Layer Quantization Control, ISO/IEC JTC1/SC29/WG11 input document m47834, Gothenburg, Sweden, July 2019.
43. [G-PCC] CE13.2 report on tile and slice based coding, ISO/IEC JTC1/SC29/WG11 input document m48892, Gothenburg, Sweden, July 2019.
44. [G-PCC] [CE13.2-related] An improvement for the uniform-geometry slice partition along the longest edge, ISO/IEC JTC1/SC29/WG11 input document m49121, Gothenburg, Sweden, July 2019.
45. G-PCC Bypass coding of bypass bins, ISO/IEC JTC1/SC29/WG11 input document m47827, Geneva, Switzerland, March 2019.
46. [G-PCC] Spatial scalability support for G-PCC, ISO/IEC JTC1/SC29/WG11 input document m47352, Geneva, Switzerland, March 2019.
47. [G-PCC] (New) Lifting and RAHT harmonization, ISO/IEC JTC1/SC29/WG11 input document m48918, Gothenburg, Sweden, July 2019.
48. [G-PCC][New proposal] Improved color transfer, ISO/IEC JTC1/SC29/WG11 input document m49407, Gothenburg, Sweden, July 2019.
49. On an improvement of RAHT to exploit attribute correlation, ISO/IEC JTC1/SC29/WG11 input document m47378, Geneva, Switzerland, March 2019.

Appendix A: Mathematical Functions

the greatest integer less than or equal to x.

the least integer greater than or equal to x.

.

the base-2 logarithm of x.

= the maximum of x1, …, xN.

the trigonometric inverse tangent function, operating on an argument x, with  
an output value in the range of −π÷2 to π÷2, inclusive, in units of radians.

Appendix B: RAHT

This Appendix provides details of how to transform a list of attributes , into a list of transform coefficients , , given a list of associated voxel locations , as side information, and how to invert the transform.

RAHT and its inverse are performed with respect to a hierarchy defined by the Morton codes of the voxel locations. The Morton code of -bit non-negative integer coordinates , , and is a -bit non-negative integer obtained by interleaving the bits of , , and . To be specific, the Morton code of non-negative -bit integers coordinates

where are the bits of , , and from (high order) to (low order), is the non-negative -bit integer

where are the bits of from (high order) to (low order).

Let denote the -bit prefix of . Let be such a prefix. Define the block at level with prefix to be the set of all points for which . Two blocks at level are sibling blocks if they have the same -bit prefix. The union of two sibling blocks at level is a block at level called their parent block.

The Region Adaptive Haar Transform of the sequence , and its inverse, can now be defined recursively as follows.

**Base case:**

Let be the attribute of a point and let be its transform. Then .

**Recursion:**

Consider two sibling blocks and their parent block. Let and be the attributes of the points in the sibling blocks, listed in increasing Morton order, and let and be their respective transforms. Similarly, let be the attributes of all points in their parent block, listed in increasing Morton order, and let be its transform. Then

where and .

In other words, the transform of the parent block is the concatenation of the two sibling blocks, with the exception that the first (DC) components of the transforms of the two sibling blocks are replaced by their weighted sum and difference, and inversely the transforms of the two sibling blocks are copied from the first and last parts of the transform of the parent block, with the exception that the DC components of the transforms of the two sibling blocks are replaced by their weighted difference and sum, namely

and

It is not difficult to show that these are inverses of each other. These are known as Givens rotations.