|  |  |
| --- | --- |
| **Joint Collaborative Team on 3D Video Coding Extensions**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  4th Meeting: Incheon, KR, 20–26 Apr. 2013 | Document: JCT3V-D0054 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **CE6.H related: Results on Updating Mechanism for Coding of Depth Lookup Table (Delta-DLT)** | | |
| *Status:* | Input Document | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Fabian Jäger Institut für Nachrichtentechnik RWTH Aachen University | Tel: Email: | +49 (0) 241 80 27678 jaeger@ient.rwth-aachen.de |
| *Source:* | RWTH Aachen University | | |

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

# Abstract

This contribution proposes an update mechanism for the coding of the Depth Lookup Table (DLT) in the slice header. In B0036 the DLT was proposed as an efficient coding approach for sparse depth maps. In the current 3D-HEVC design, the DLT is coded in the SPS for each view independently. In this contribution it is proposed to code the DLT in the slice header and allow to update the DLT entries in the temporal and also from the base to the dependent views. By this fine-grained update-mechanism it is possible to adapt the current slice’s DLT entries to the original depth map characteristics.

# Introduction

In 3D video coding, a depth map for each view needs to be encoded besides the conventional video data. These depth maps show different signal characteristics compared to video data as they contain piecewise smooth regions bounded by strong edges. As depth maps are often estimated from texture data or are pre-processed, their histogram might be relatively sparse. As a result, a Depth Lookup Table (DLT) was proposed [1] to exploit the histogram characteristics by only signaling difference indexes of the DLT instead of signaling the residual depth values themselves. By this approach the bit depth of these residual values can be reduced, which consequently results in higher coding efficiency.

The DLT is constructed at the encoder by analyzing the histogram of the original, uncompressed depth map. This DLT is afterwards transmitted to the decoder to allow for the mapping of indexes to actual depth values. Histogram values of the depth maps may vary over time and therefore there is a requirement for an update mechanism. Moreover, in a multiview coding scenario, multiple depth maps may have different depth map histograms and in these cases such an update mechanism is also beneficial to the overall coding performance.

# DLT Coding in HTM 6

In the latest specification of the 3D extension for High Efficiency Video Coding, the DLT is only sent once per sequence, separately for all views. This approach keeps the overhead for the DLT signaling relatively low.

It was also proposed to signal the DLT in the slice header of each I-Slice of the base view[2]. In this approach, the DLT values are updated more regularly in the temporal direction to allow for histogram changes over time. In this case, all the dependent views inherit the base view’s DLT as it is assumed that the depth map histogram over all views is the same.

Signaling the lookup table only once per sequence and for each view separately results in a very low overhead for the DLT values, but is relatively inflexible in terms of temporal and spatial (inter-view) updating. Moreover, inter-view redundancies of the lookup tables are not exploited to reduce the required bitrate for the DLT.

The alternative solution, to signal the DLT in the slice header of I-Slices for the base view and inherit that DLT for the dependent views, lacks the ability to update the lookup table in the temporal direction more regularly and also does not allow for inter-view update of the DLT. The assumption that the DLT values are always the same for all coded views is in many cases too restrictive and results in reduced depth map quality in the dependent views. If a dependent view’s depth map shows different histogram characteristics compared to the base view’s, then the reconstruction of that depth map cannot even reach all original depth values due to the plain copy of the non-optimal DLT.

# Description of DLT Update Mechanism

The proposed solution for updating DLT values over time and between different views, aims at high flexibility for the encoder while reducing the required bitrate for the DLT at the same time. As depth map characteristics may vary significantly between different sequences and even within the same sequence (between frames and between views), high flexibility for coding the Depth Lookup Table is required.

The proposal uses a unified approach for updating the DLT by signaling only the differences between the lookup table, which is adapted to the current time instance and/or view, and the latest reference lookup table. Moreover, a single flag is proposed that allows for a simple copy of the latest reference DLT. This flag is efficient for sequences with very high correlation of the DLT between frames and/or between views.

## Overview of the Coding Approach

1. The DLT is explicitly coded only for the base view’s slice for random access points (RAP) to support random access coding scenarios.
2. All other slices (of the base and of the dependent views) must signal with a single flag whether the latest reference DLT (DLT at last RAP for current view) is just copied to the current slice.
3. If the DLT is not simply copied to the current slice, then a difference between the latest reference DLT and the current DLT is signaled. The difference contains both, values that are to be added to the DLT and values that are to be removed from the DLT. The updating may be computed by an XOR-like operation between the reference DLT and the update DLT.

## Detailed Description of Delta-DLT Coding

The proposed update mechanism allows for updating the DLT in the temporal and the spatial (inter-view) direction.

At the highest coding level (VPS) a single flag signals whether to use DLT as a coding tool or not. If this flag is set equal to 0, there is no information to be signaled in the slice header related to DLT. In this case, DLT coding is omitted completely. If the flag is set equal to 1, the signaling and updating mechanism for the Depth Lookup Table, as described below, is applied.

The bit length of the DLT index values has to be signaled (e.g. in the PPS) to make sure that the index values are always decodable, even in cases when an intermediate delta-DLT was lost.  
**Alternative solution**: The number of bits per DLT index can also be derived from the number of DLT entries of the latest reference DLT (from the last RAP).

In general, the Depth Lookup Table is explicitly signaled for all slices of the base view at RAP. This is required to allow for random access into the sequence. All other slice types can signal an update to the latest reference DLT, as illustrated in Figure 1.

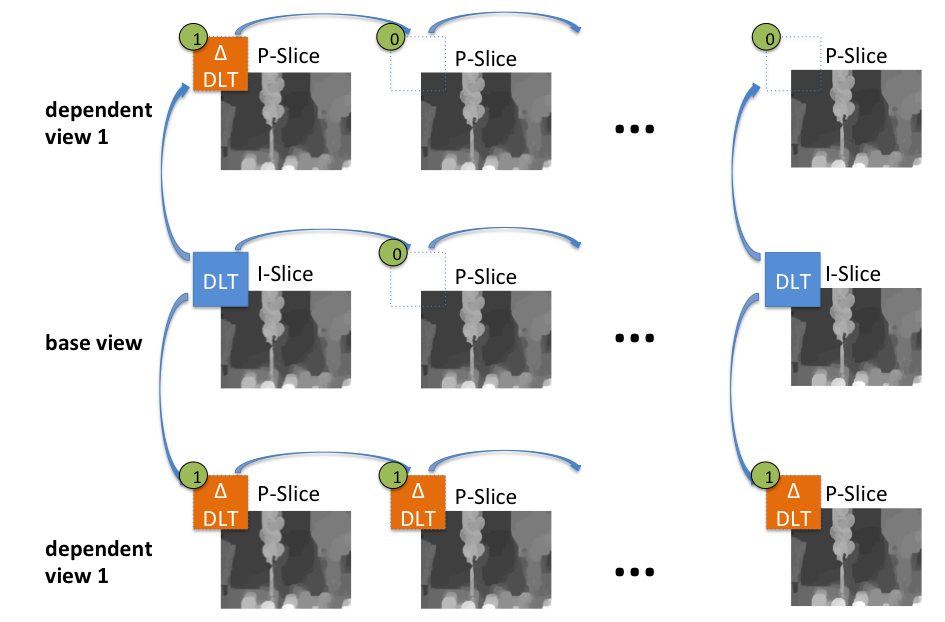


Figure 1: Updating scheme including flag (green circles) to signal whether the slice uses a delta-DLT to update the latest reference DLT. If the flag is set equal to 1, a delta-DLT is signaled for the particular slice. If the flag is set equal to 0, no delta-DLT is signaled, which consequently means that the reference DLT is simply copied.

If a delta-DLT is signaled, then the following procedure is used to compute the new DLT from the reference DLT and the delta-DLT:

* For each value in the delta-DLT it needs to be checked whether the particular value is also present in the reference DLT.
  + If the value is present in the reference DLT, that corresponding value is not added to the resulting DLT (value is **removed**)
  + If the value is not present in the reference DLT, that corresponding value is added to the resulting DLT (value is **added**)
* All other values of the reference DLT are just copied.

An example of the described procedure is illustrated in Figure 2. The internal or coded representation does not have an influence on this procedure as it works the same way for an integer representation or a bit string representation, as proposed in [3]. The described updating mechanism can be resembled by a simple XOR operation on the DLT in a bit string representation, as illustrated in Figure 3.

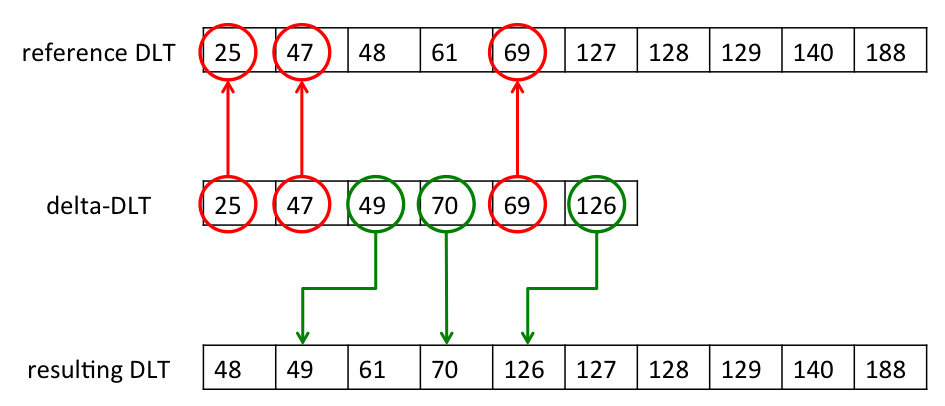


Figure 2: Updating procedure for integer representation. The delta-DLT contains both, values to be added to the reference DLT and values to be removed from the reference DLT.

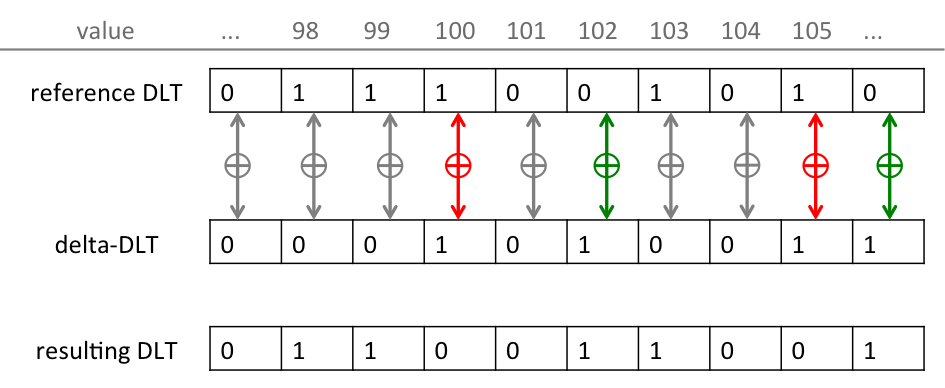


Figure 3: Updating procedure for bit string representation can be computed by simple XOR operation on each single bit of the string.

The proposed solution for updating the Depth Lookup Table in both dimensions, time and space, allows for flexible adaptation of the DLT to changing depth ranges over time and between views. Whenever this functionality is not required (e.g. when the correlation of the depth map histograms is high), the proposed solution allows to signal with a single flag to just copy the reference DLT instead of updating. By this approach the overhead for the additional flexibility is extremely low.

# Modifications to the Specification Text

## Video parameter set extension 2 syntax

|  |  |
| --- | --- |
| vps\_extension2( ) { | Descriptor |
| while( !byte\_aligned( ) ) |  |
| **vps\_extension\_byte\_alignment\_reserved\_one\_bit** | u(1) |
| for( i = 0; i <= vps\_max\_layers\_minus1; i++ ) { |  |
| … |  |
| if ( DepthLayerFlag[ layerId ] ) { |  |
| **vps\_dmm\_flag**[ layerId ] | u(1) |
| **lim\_qt\_pred\_flag**[ layerId ] | u(1) |
| **dlt\_flag**[ layerId ] | u(1) |
| ~~if( dlt\_flag[ layerId ] ) {~~ |  |
| **~~num\_depth\_values\_in\_dlt~~**~~[ layerId ]~~ | ~~ue(v)~~ |
| ~~for ( j = 0; j < num\_depth\_values\_in\_dlt[ layerId ] ; j++) {~~ |  |
| **~~dlt\_depth\_value~~**~~[ layerId ][ j ]~~ | ~~ue(v)~~ |
| ~~}~~ |  |
| ~~}~~ |  |
| } |  |
| } |  |
| **iv\_mv\_scaling\_flag** | u(1) |
| } |  |

## Video parameter set extension 2 semantics

**dlt\_flag**[ layerId ] equal to 1 specifies that depth lookup table is used and that residual values for simplified depth coded coding units are to be interpreted as indices of the depth lookup table for depth view components with layer\_id equal to layerId. dlt\_flag[ layerId ] equal to 0 specifies that depth lookup table is not used and residual values for simplified depth coded coding units are not to be interpreted as indices for depth view components with layer\_id equal to layerId. When dlt\_flag[ layerId ] is not present, it shall be inferred to be equal to 0.

**~~num\_depth\_values\_in\_dlt~~**~~[ layerId ] specifies the number of different depth values and the number of elements in the depth lookup table for depth view components of the current layer with layer\_id equal to layerId.~~

**~~dlt\_depth\_value[~~**~~layerId~~**~~]~~**~~[ j ] specifies the j-th entry in the depth lookup table for depth view components with layer\_id equal to layerId.~~

## Slice header extension syntax

|  |  |
| --- | --- |
| slice\_header\_extension( ) { | Descriptor |
| if( cp\_in\_slice\_header\_flag ) { |  |
| for ( i = 0; i < ViewIdx; i++ ) { |  |
| **cp\_scale**[ i ] | se(v) |
| **cp\_off**[ i ] | se(v) |
| **cp\_inv\_scale\_plus\_scale**[ i ] | se(v) |
| **cp\_inv\_off\_plus\_off**[ i ] | se(v) |
| } |  |
| } |  |
| if( dlt\_flag[ layerId ] ) { |  |
| if( RapPicFlag && ViewId[ layerId ] = = 0 ) { |  |
| **num\_depth\_values\_in\_dlt**[ layerId ] | ue(v) |
| for ( j = 0; j < num\_depth\_values\_in\_dlt[ layerId ] ; j++) { |  |
| **dlt\_depth\_value**[ layerId ][ j ] | ue(v) |
| } |  |
| } else { |  |
| **dlt\_update\_flag**[ layerId ] | u(1) |
| if( dlt\_update\_flag[ layerId ] ) { |  |
| **num\_delta\_values\_in\_dlt**[ layerId ] | ue(v) |
| for ( j = 0; j < num\_delta\_values\_in\_dlt[ layerId ] ; j++) { |  |
| **dlt\_delta\_value**[ layerId ][ j ] | ue(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

## Slice header extension semantics

**num\_depth\_values\_in\_dlt**[ layerId ] specifies the number of different depth values and the number of elements in the depth lookup table for depth view components of the current layer with layer\_id equal to layerId.

**dlt\_depth\_value[** layerId **]**[ j ] specifies the j-th entry in the depth lookup table for depth view components with layer\_id equal to layerId.

**dlt\_update\_flag**[ layerId ] equal to 1 specifies that the current reference depth lookup table is to be updated for depth view components with layer\_id equal to layerId. dlt\_update\_flag[ layerId ] equal to 0 specifies that no delta depth lookup table information is coded for the depth view components with layer\_id equal to layerId.

**num\_delta\_values\_in\_dlt**[ layerId ] specifies the number of different delta depth values and the number of elements in the delta depth lookup table for depth view components of the current layer with layer\_id equal to layerId.

**dlt\_delta\_value[** layerId **]**[ j ] specifies the j-th entry in the delta depth lookup table for depth view components with layer\_id equal to layerId.

## Decoding process for a depth lookup table

This process is only invoked when dlt\_flag[ nuh\_layer\_id ] is equal to 1 and when RapPicFlag is equal to 1 and when ViewId[ nuh\_layer\_id ] is equal to 0.

The list elements Idx2DepthValue[ i ] specifying the depth value of the i-th index in the lookup table with i ranging from 0 to num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] – 1, inclusive is derived as follows.

* For i = 0..num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] – 1 the elements in Idx2DepthValue are derived as follows.
  + Idx2DepthValue[ i ] is set equal to dlt\_depth\_value[ nuh\_layer\_id ][ i ]

The list elements DepthValue2Idx[ d ] specifying the index of depth values d in the lookup table with d ranging from 0 to BitDepthY – 1, inclusive are derived as specified in the following.

for (d = 0; d < BitDepthY; d++) {  
 idxLower = 0   
 for ( iL = 1, foundFlag = 0; iL < num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] && !foundFlag; iL++ )   
 if ( Idx2DepthValue[ iL ] > d ) {  
 idxLower = iL – 1  
 foundFlag = 1  
 }  
 idxUpper = num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] – 1  
 for ( iU = num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] – 2, foundFlag = 0; iU >= 0 && !foundFlag; iU++ )   
 if ( Idx2DepthValue[ iU ] < d ) {  
 idxUpper = iU + 1  
 foundFlag = 1  
 }  
 if ( Abs( d – Idx2DepthValue[ idxLower ] ) < Abs ( d – Idx2DepthValue[ idxUpper ] ) )   
 DepthValue2Idx[ d ] = idxLower  
 else  
 DepthValue2Idx[ d ] = idxUpper

The reference depth lookup table list for all layers RefDltValues[ i ][ j ] and the number of its entries RefDltNumValues[ i ] with i ranging from 0 to vps\_max\_layers\_minus1, inclusive and j ranging from 0 to num\_depth\_values\_in\_dlt[ nuh\_layer\_id ] – 1, inclusive are derived as follows:

RefDltNumValues[ i ] is set equal to num\_depth\_values\_in\_dlt[ nuh\_layer\_id ]

* For j = 0.. RefDltNumValues[ i ] – 1 the elements in RefDltValues[ i ][ j ] are derived as follows:
  + RefDltValues[ i ][ j ] is set equal to dlt\_depth\_value[ nuh\_layer\_id ][ j ]

## Decoding process for a depth lookup table update

This process is only invoked when dlt\_update\_flag[ nuh\_layer\_id ] is equal to 1.

The list elements Idx2DepthValue[ i ] specifying the depth value of the i-th index in the lookup table is derived as follows.

NumValuesInDlt is set equal to RefDltNumValues[ nuh\_layer\_id ].

* For i = 0..NumValuesInDlt – 1 the elements in Idx2DepthValue are initialized as follows.
  + Idx2DepthValue[ i ] is set equal to RefDltValues[ nuh\_layer\_id ][ i ]
* For i = 0..num\_delta\_values\_in\_dlt[ nuh\_layer\_id ] – 1, the elements of Idx2DepthValue are updated as follows.
  + When Idx2DepthValue does not contain dlt\_delta\_value[ nuh\_layer\_id ][ i ], then Idx2DepthValue is updated as specified in the following:
    - NewIdx = 0
    - for ( idx = 0, foundFlag = 0; idx < NumValuesInDlt && !foundFlag; idx++ )
      * if ( Idx2DepthValue[ idx ] > dlt\_delta\_value[ nuh\_layer\_id ][ i ] ) {
        + NewIdx = idx
        + foundFlag = 1
      * }
    - for ( idx = NumValuesInDlt – 1; idx >= NewIdx; idx = idx – 1 )
      * Idx2DepthValue[ idx + 1 ] = Idx2DepthValue[ idx ]
    - Idx2DepthValue[ NewIdx ] = dlt\_delta\_value[ nuh\_layer\_id ][ i ]
    - NumValuesInDlt = NumValuesInDlt + 1
  + Otherwise, Idx2DepthValue is updated as specified in the following:
    - OldIdx = 0
    - for ( idx = 0, foundFlag = 0; idx < NumValuesInDlt && !foundFlag; idx++ )
      * if ( Idx2DepthValue[ idx ] = = dlt\_delta\_value[ nuh\_layer\_id ][ i ] ) {
        + OldIdx = idx
        + foundFlag = 1
    - }
    - for ( idx = OldIdx; idx < NumValuesInDlt – 1; idx = idx + 1 )
      * Idx2DepthValue[ idx ] = Idx2DepthValue[ idx + 1 ]
    - NumValuesInDlt = NumValuesInDlt – 1

When RapPicFlag is equal to 1, the reference depth lookup table is derived as follows:

The reference depth lookup table list RefDltValues[ nuh\_layer\_id ][ i ] and the number of its entries RefDltNumValues[ nuh\_layer\_id ] i ranging from 0 to NumValuesInDlt  – 1, inclusive are derived as follows:

RefDltNumValues[ nuh\_layer\_id ] is set equal to NumValuesInDlt

* For i = 0.. RefDltNumValues[ nuh\_layer\_id ] – 1 the elements in RefDltValues[ nuh\_layer\_id ][ i ] are derived as follows:
  + RefDltValues[ nuh\_layer\_id ][ i ] is set equal to Idx2DepthValue[ i ]

# Simulation Results

## Configuration A (Delta-DLT)

Test configuration A just adds the update functionality to HTM 6.1. The anchor is HTM 6.1.

### Random Access Coding Configuration (CTC)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video PSNR / video bitrate | video PSNR / total bitrate | synth PSNR / total bitrate | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,01% | 0,19% | 0,22% | 99,2% | 101,5% |
| Kendo | 0,0% | 0,1% | -0,1% | 0,00% | 0,18% | 0,11% | 103,2% | 101,5% |
| Newspaper\_CC | 0,0% | 0,1% | 0,0% | 0,00% | 0,18% | 0,24% | 98,3% | 103,6% |
| GT\_Fly | 0,0% | -0,3% | -0,1% | -0,04% | 0,49% | 0,49% | 99,7% | 99,1% |
| Poznan\_Hall2 | 0,0% | 0,7% | -0,4% | 0,02% | 0,16% | 0,10% | 99,6% | 98,4% |
| Poznan\_Street | 0,0% | -0,2% | -0,1% | -0,03% | 0,32% | 0,31% | 98,0% | 101,0% |
| Undo\_Dancer | 0,0% | -0,1% | -0,4% | -0,07% | 0,08% | 0,00% | 102,1% | 103,2% |
| 1024x768 | 0,0% | 0,1% | -0,1% | 0,01% | 0,18% | 0,19% | 100,2% | 102,2% |
| 1920x1088 | 0,0% | 0,0% | -0,3% | -0,03% | 0,27% | 0,22% | 99,9% | 100,4% |
| **average** | **0,0%** | **0,0%** | **-0,2%** | **-0,02%** | **0,23%** | **0,21%** | **100,0%** | **101,2%** |

### All-Intra Coding Configuration (AI)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video PSNR / video bitrate | video PSNR / total bitrate | synth PSNR / total bitrate | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 99,2% | 101,2% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,00% | -0,21% | -0,41% | 98,4% | 100,9% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 101,2% | 99,5% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,00% | 0,02% | 0,02% | 100,7% | 98,2% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,00% | -0,01% | -0,04% | 100,1% | 101,5% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 99,6% | 105,2% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,17% | 99,3% | 96,6% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,00% | -0,07% | -0,14% | 99,9% | 100,5% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,04% | 99,9% | 100,4% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,00%** | **-0,03%** | **-0,04%** | **99,8%** | **100,4%** |

## Configuration B (Delta-DLT + Bit String Coding)

In configuration B, the proposed Delta-DLT modifications are combined with an alternative coding method for DLT entries, which were proposed in C0142. The anchor is HTM 6.1.

### Random Access Coding Configuration (CTC)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video PSNR / video bitrate | video PSNR / total bitrate | synth PSNR / total bitrate | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,01% | 0,11% | 0,14% | 99,5% | 99,2% |
| Kendo | 0,0% | 0,1% | -0,1% | 0,00% | 0,09% | 0,02% | 98,2% | 98,6% |
| Newspaper\_CC | 0,0% | 0,1% | 0,0% | 0,00% | 0,07% | 0,13% | 101,7% | 102,4% |
| GT\_Fly | 0,0% | -0,3% | -0,1% | -0,04% | 0,23% | 0,22% | 101,3% | 102,1% |
| Poznan\_Hall2 | 0,0% | 0,7% | -0,4% | 0,02% | 0,07% | 0,01% | 103,0% | 100,8% |
| Poznan\_Street | 0,0% | -0,2% | -0,1% | -0,03% | 0,05% | 0,04% | 98,6% | 100,1% |
| Undo\_Dancer | 0,0% | -0,1% | -0,4% | -0,07% | -0,03% | -0,12% | 98,2% | 99,4% |
| 1024x768 | 0,0% | 0,1% | -0,1% | 0,01% | 0,09% | 0,09% | 99,8% | 100,1% |
| 1920x1088 | 0,0% | 0,0% | -0,3% | -0,03% | 0,08% | 0,04% | 100,3% | 100,6% |
| **average** | **0,0%** | **0,0%** | **-0,2%** | **-0,02%** | **0,08%** | **0,06%** | **100,1%** | **100,4%** |

### All-Intra Coding Configuration (AI)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video PSNR / video bitrate | video PSNR / total bitrate | synth PSNR / total bitrate | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 98,2% | 99,5% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,00% | -0,21% | -0,41% | 98,7% | 99,2% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 103,1% | 101,8% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,00% | 0,02% | 0,02% | 100,4% | 102,3% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,00% | -0,01% | -0,04% | 98,6% | 100,4% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,00% | 98,8% | 101,2% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,17% | 100,0% | 98,6% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,00% | -0,07% | -0,14% | 100,0% | 100,2% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,00% | 0,00% | 0,04% | 99,5% | 100,6% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,00%** | **-0,03%** | **-0,04%** | **99,7%** | **100,4%** |

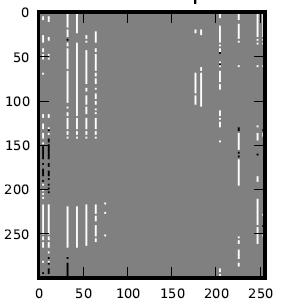
# Cross Check

The cross check of the proposed modifications (configurations A and B) to DLT coding was performed by Ghent University. They investigated the source code modifications and ran the simulations for verification of the presented results.

In their investigation they did not find any problems with the source code. Their simulation results perfectly match with those presented in this document.

# Multi-range DLT Update Coding

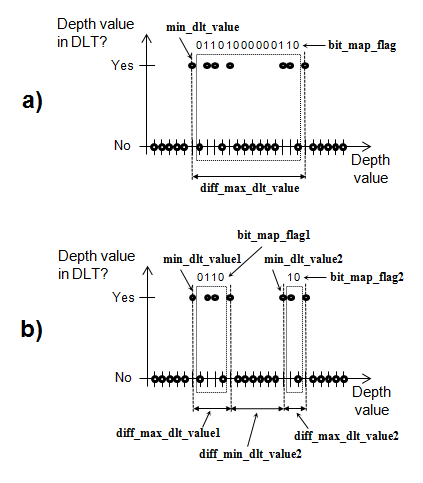
The original range-based approach (binary string) of RCBM [3] used to represent the values that are signaled in the inter-view DLT updates in the non-base views is modified. In the proposed method, the original single range representation can be divided into 2 sub-ranges in order to exploit the characteristics of the inter-view DLT updates (see Fig.1). The decision whether to use single or multi-range representation is made based on the number of bits required to represent the DLT using each representation. In this method, for each DLT update in the non-base view, first sub-range is signaled in the same way as the original single-range of RCBM. Second sub-range is signaled by indicating the offset to the previous sub-range and the width of the current sub-range (width is calculated based on the difference between minimum and maximum value in the sub-range). The values in the sub-range are signaled using the representation as in the binary string of RCBM. Consequently, the original range is divided into 2 sub-ranges, where the division point between the sub-ranges is determined by the largest gap between the signaled values in DLT (in this sense gap between signaled DLT values means a number of neighboring depth values that are not signaled in DLT). Fig. 2b shows an example of the proposed approach using 2 sub-ranges.



Time

Depth level

**Fig. 1. Examplary inter-view DLT update (Kendo, view 5).**

****

**Fig. 2. DLT coding: a) single-range, b) multi-range (with 2 sub-ranges).**

## Syntax Changes

|  |  |
| --- | --- |
| slice\_header() { |  |
| ... |  |
| **dlt\_flag**[ i ] | u(1) |
| if( dlt\_flag[ i ] ) { |  |
| **dlt\_update\_flag**[ i ] | u(1) |
| if( dlt\_update\_flag[ i ] ) { |  |
| **code\_singlerange\_dlt\_flag**[ i ] | u(1) |
| if(code\_singlerange\_dlt\_flag[ i ] ) { |  |
| **code\_full\_bit\_map\_flag**[i] | u(1) |
| if(!**code\_full\_bit\_map\_flag**[i]){ |  |
| **min\_dlt\_value**[i] | u(v) |
| **diff\_max\_dlt\_value**[i] | u(v) |
| } |  |
| for(j=0;j< MaxDltValue[i] - min\_dlt\_value[i] – 1;j++) |  |
| **bit\_map\_flag**[i][j] | u(1) |
| } |  |
| else { |  |
| **min\_dlt\_value1**[i] | u(v) |
| **diff\_max\_dlt\_value1**[i] | u(v) |
| for(j=0;j< MaxDltValue1[i] - min\_dlt\_value1[i] – 1;j++) |  |
| **bit\_map\_flag1**[i][j] | u(1) |
| **diff\_min\_dlt\_value2**[i] | u(v) |
| **diff\_max\_dlt\_value2**[i] | u(v) |
| for(j=0;j< MaxDltValue2[i] - MinDltValue2[i] – 1;j++) |  |
| **bit\_map\_flag2**[i][j] | u(1) |
| } |  |
| } |  |
| } |  |
| ... |  |
| } |  |

## Semantics

**code\_singlerange\_dlt\_flag**[i] specifies whether to code the single range or multi-range DLT (containing 2 sub-ranges) with layer\_id equal to i.

**min\_dlt\_value1**[i] specifies the smallest value in the depth lookup table for depth view components with layer\_id equal to i. The number of bits used to represent it is log2(MAX\_DEPTH\_VALUE + 1). When min\_dlt\_value[i] is not present, it shall be inferred to be -1.

**diff\_max\_dlt\_value1**[i] specifies the difference between the largest value in first sub-range of the depth lookup table and smallest value in the depth lookup table for depth view components with layer\_id equal to i. MaxDltValue1[i] is computed as follows: MaxDltValue1[i] = min\_dlt\_value1[i] + diff\_max\_dlt\_value1[i]. The number of bits used to represent diff\_max\_dlt\_value1[i] is log2(MAX\_DEPTH\_VALUE + 1 - min\_dlt\_value1[i]).

**bit\_map\_flag1**[i][j]specifies the j-th entry in the bit map for first sub-range depth view components with layer\_id equal to i.

**diff\_min\_dlt\_value2**[i] specifies the difference between the smallest value in second sub-range of the depth lookup table and largest value in the first sub-range of the depth lookup table for depth view components with layer\_id equal to i. MinDltValue2[i] is computed as follows: MinDltValue2[i] = MaxDltValue1[i] + diff\_min\_dlt\_value2[i]. The number of bits used to represent diff\_min\_dlt\_value2[i] is log2(MAX\_DEPTH\_VALUE + 1 - MaxDltValue1[i]).

**diff\_max\_dlt\_value2**[i] specifies the difference between the largest value in second sub-range of the depth lookup table and smallest value in the second sub-range of the depth lookup table for depth view components with layer\_id equal to i. MaxDltValue2[i] is computed as follows: MaxDltValue2[i] = MinDltValue2[i] + diff\_max\_dlt\_value2[i]. The number of bits used to represent diff\_max\_dlt\_value2[i] is log2(MAX\_DEPTH\_VALUE + 1 - MinDltValue2[i]).

**bit\_map\_flag2**[i][j]specifies the j-th entry in the bit map for second sub-range depth view components with layer\_id equal to i.

## Simulation Results

# Conclusion

# References

[1] F. Jäger, “3D-CE6.h Results on Simplified Depth Coding with an optional Depth Lookup Table,” Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) of ITU-T VCEG and ISO/IEC MPEG, Shanghai, China, JCT3V-B0036, 2012.

[2] I. Lim, H. C. Wey, and D. S. Park, “3D-CE6.h Related: Improved depth lookup table (DLT),” Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) of ITU-T VCEG and ISO/IEC MPEG, Geneva, Switzerland, JCT3V-C0093, 2013.

[3] K. Zhang, J. An, and S. Lei, “3D-CE6.h related: An efficient coding method for DLT in 3DVC,” Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) of ITU-T VCEG and ISO/IEC MPEG, Geneva, Switzerland, JCT3V-C0142, 2013.

# Patent rights declaration

**RWTH Aachen University may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**