|  |  |
| --- | --- |
| **Joint Collaborative Team on 3D Video Coding Extension Development**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  2nd Meeting: Shanghai, CN, 13–19 Oct. 2012 | Document: JCT3V-B0036 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **3D-CE6.h: Simplified Depth Coding with an optional Depth Lookup Table** | | |
| *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 presents a modified intra-coding scheme for depth map coding in 3D video. As depth maps show unique characteristics like piecewise smooth regions bounded by sharp edges at depth discontinuities, new coding tools are required to approximate these signal characteristics. In the current 3DV-HTM software (version 4.0.1), there are two kinds of intra prediction modes for depth maps:   
1) The directional intra prediction known from HEVC and 2) the depth modeling modes (DMM). The latter can enhance the BD-rate of synthesized views especially when modeling the mentioned depth discontinuities. The addition of the four additional DMM prediction modes for depth increase the number of bits required to signal the optimal prediction mode for each coded block. Moreover, there are three DMM modes, which overlap with directional intra prediction modes and consequently result in signaling overhead.  
In this contribution an alternative intra coding scheme is proposed, which adds an alternative coding path for each CU. If the Simplified Depth Coding (SDC) mode is selected, there is only a set of four different prediction modes available. Due to depth maps’ signal characteristics these four prediction modes are sufficient to generate a sufficiently good prediction signal. After having selected the optimal prediction modes among the four, the resulting residual is not coded using transform. As the SDC prediction stage always results in one or two depth segments per coded block, a single residual DC depth value is coded for each of these segments. By skipping the transform step and coding the residual based on the pixel-domain information, ringing artifacts are eliminated for SDC-coded blocks.   
Moreover, the number of bits to signal the residual values for each segment can be further reduced by integrating a Depth Lookup Table (DLT), which maps depth values to valid depth values of the original depth map. The DLT is constructed based on an initial analysis of the input depth map and is then coded in the SPS. For sequences with strongly quantized depth maps DLT yields additional gain on top of the gain coming from DMC alone.

**Random Access Coding Performance**

The proposed intra coding scheme for depth maps results in an overall coding performance gain (including texture and depth) of 0.3 % BD-Rate savings without the DLT and 0.54 % BD-Rate including the Depth Lookup Table.

**All Intra Coding Performance**

For an all-intra test case, the proposed coding scheme yields **BD-Rate savings of 1.1 %** without the DLT and **2.0 %** including the Depth Lookup Table.

# Introduction

Depth maps, like in 3D video, show different signal characteristics compared to natural video data. Distortions in these depth maps have an indirect impact on the visual quality of the displayed video as they are used to synthesize new views of the same scene, while the depth information is never directly shown to the user. Compressing depth maps with algorithms optimized for natural 2D videos results in strong ringing artifacts along depth discontinuities, which then produce geometric distortions in the synthesized views. Thus, new compression algorithms have to be developed that are adapted to the signal characteristics of depth maps.

Previous work on compression of depth data regarded depth data as gray-colored video and compressed it with conventional transform-based video coding algorithms as found in H.264/AVC [[1](#Platzhalter1)]. It was shown [[2](#Jäg11)] that these conventional coding tools yield relatively high compression efficiency in terms of PSNR, but at the same time introduce ringing artifacts along sharp edges in the original depth maps. These artifacts result in geometric distortions in the view synthesis stage [[2](#Jäg11)].

More recent depth compression algorithms approximate the depth map's signal characteristics by partitioning into triangular meshes [[3](#Sar10)] or platelets [[4](#Mor06)] and modeling each segment by an appropriate 2D function. These purely model-based approaches can also be combined with conventional transform-based tools by introducing an additional coding mode, like the sparse-dyadic (SD) mode [[5](#Liu10)]. Here, an SD-coded block is partitioned into two segments, which are described by two constant depth values.

As the preservation of depth discontinuities is most important when compressing depth maps, another approach is to losslessly compress the location of these discontinuities and approximate the piecewise smooth regions, as previously proposed [[2](#Jäg11)]. The disadvantage of this approach is the inability of reaching low bitrates due to the lossless encoding of depth contours.

# General Concept

The proposed SDC coding approach is an extension of the intra coding mode, which is available in the HEVC-based 3DV-HTM reference software. For an SDC-coded block, the prediction mode is still INTRA. An additional SDC-Flag signals the usage of the Simplified Depth Coding mode. If a block is coded with SDC, partition size is always 2Nx2N and therefore not signaled in the bitstream. Instead of coding quantized transform coefficients SDC-coded blocks need to code the following types of information:

1. The **type of segmentation/prediction** of the current block. Possible values are
   1. DC (1 segment)
   2. DMM Mode 1 – Explicit Wedgelets (2 segments)
   3. DMM Mode 2 – Intra-Predicted Wedgelets (2 segments)
   4. Planar (1 segment)
2. For the two DMM modes, additional prediction information is coded, as described in [[6](#Tec12)]
3. For each resulting segment, a **residual value** (in the pixel domain) is signaled in the bitstream

Before coding, the residual values are optionally mapped to values, which are present in the original, uncompressed depth map by using a Depth Lookup Table (DLT). Consequently, residual values can be coded by signaling only the index into this lookup table, which reduces the bit depth of residual magnitudes.

# Depth Lookup Table

An analysis of the 3DV test sequences [[7](#MPE11)] has shown that the estimated depth maps do not utilize the full available depth range of . Only a small amount of different depth levels occur in those sequences due to strong quantization. In an initial analysis step, a dynamic depth lookup-table is constructed by analyzing a certain number of frames (e.g. an intra period like in our simulations) of the input sequence. This depth lookup-table is used during the coding process to reduce the effective signal bit-depth of the residual signal.

## Analysis Step

In the analysis step the encoder reads a pre-defined number of frames from the input video sequence to be coded and scans all pixels for available depth values. During this process a mapping table is generated that maps depth values to valid depth values based on the original uncompressed depth map.

**Algorithmic Details**

Input: Depth map of pixels at time instance

Output: Depth Lookup Table

Index Lookup Table

Depth Mapping Table

Number of valid depth values

Algorithm:

1. Initialization

* boolean vector for all depth values
* index counter

1. Process each pixel position in for multiple time instances :

* Set to mark valid depth values

1. Count number of values in 🡪
2. For each with :

* Set
* Set
* Set

1. For each with :

* Find and
* Set

Set

## ****Coding Step****

Instead of coding a residual depth value for a given coding unit, the depth value is mapped to its corresponding index in the list of valid depth maps. This mapping table needs to be transmitted to the decoder for the inverse lookup from index to valid depth value.

The advantage of using this lookup table is the reduced bit depth of the residual index for sequences with reduced depth range (e.g. all estimated depth maps in the MPEG 3DV activity [[7](#MPE11)]).

**Algorithmic Details**

Input: Original depth value

Predicted depth value

Index Lookup Table

Number of valid depth values

Output: Residual index to be coded

Algorithm:

The computed residual index is then coded with a significance flag, a sign flag and with bits for the magnitude of the residual index.

# Syntax and Semantics

In this section a general overview of the syntax and semantics that would need to be changed in the current working draft is described.

## Modifications to Video Parameter Set Syntax (VPS)

|  |  |
| --- | --- |
| vps\_extension( ) { | **Descriptor** |
| while( !byte\_aligned( ) ) |  |
| **vps\_extension\_byte\_alignment\_reserved\_one\_bit** | u(1) |
| **num\_additional\_layer\_operation\_points** | u(8) |
| **num\_additional\_profile\_level\_sets\_minus1** | u(8) |
| for( i = 0; i <= vps\_max\_layers\_minus1; i++ ) { |  |
| // mapping of layer ID to scalability dimension IDs |  |
| **reserved\_zero\_4bits\_num\_types**[ i ] | u(4) |
| **reserved\_zero\_4bits\_type**[ i ] | u(4) |
| **view\_id**[ i ] | u(8) |
| if ( i>0 ) |  |
| **num\_direct\_ref\_layers**[ i ] | u(6) |
| for( j = 0; j < num\_direct\_ref\_layers[ i ] && i; j++ ) |  |
| **ref\_layer\_id**[ i ][ j ] | u(6) |
| } |  |
| for( i = 0; i <= num\_additional\_profile\_level\_sets\_minus1; i++ ) |  |
| profile\_tier\_level( 1, vps\_max\_sub\_layers\_minus1 ) |  |
| for( i = 1; i <= num\_additional\_layer\_operation\_points; i++ ) { |  |
| op\_point( i ) |  |
| if (num\_additional\_profile\_level\_sets) |  |
| **profile\_level\_idx**[ i ] |  |
| **}** |  |
| for( i = 0; i <= vps\_max\_layers\_minus1; i++ ) { |  |
| if (i) { |  |
| **multi\_view\_mv\_pred\_flag**[ i ] | u(1) |
| **multi\_view\_residual\_pred\_flag**[ i ] | u(1) |
| } |  |
| if ( i%1) { |  |
| **enable\_dmm\_flag**[ i ] | u(1) |
| **use\_mvi\_flag**[ i ] | u(1) |
| **use\_dlt\_flag** | u(1) |
| if( use\_dlt\_flag ) { |  |
| **num\_depth\_values\_in\_dlt** | ue(v) |
| for ( i = 0; i < num\_depth\_values\_in\_dlt ; i++ ) { |  |
| **dlt\_depth\_value[** i **]** | ue(v) |
| } |  |
| } |  |
| } |  |
| **}** |  |
| } |  |

**use\_dlt\_flag** equal to 1 specified that depth lookup table is used and that residual values for SDC-coded coding units are to be interpreted as indexes of the depth lookup table. use\_dlt\_flag equal to 0 means that depth lookup table is not used and residual values for SDC-coded coding units are not to be interpreted as indexes.

**num\_depth\_values\_in\_dlt** specifies the number of different depth values and the number of elements in the depth lookup table.

**dlt\_depth\_value[** i **]** specifies a valid depth value and an entry in the depth lookup table.

## Modifications to Coding Unit

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize , ctDepth) { | **Descriptor** |
| if( transquant\_bypass\_enable\_flag ) { |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| } |  |
| if( slice\_type != I   && TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = -1) |  |
| **skip\_flag**[ x0 ][ y0 ] | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, log2CbSize ) |  |
| else { |  |
| if( TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = 1 ) { |  |
| nCbS = ( 1 << log2CbSize ) |  |
| if( slice\_type != I ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( PredMode = = MODE\_INTRA && depthFlag ) |  |
| **sdc\_flag** | ae(v) |
| if( sdc\_flag ) { |  |
| SdcPredMode = -1 |  |
| while( !sdc\_most\_probable\_mode\_flag ) { |  |
| SdcPredMode++ |  |
| **sdc\_most\_probable\_mode\_flag** | ae(v) |
| } |  |
| if( SdcPredMode = = 1 | | SdcPredMode = = 2 ) |  |
| **wedge\_full\_tab\_idx[**x0 **][**y0 **]** | ae(v) |
| if(SdcPredMode == 2 ) { |  |
| **dmm\_delta\_end** | ae(v) |
| if( dmm\_delta\_end ) { |  |
| **dmm\_delta\_end\_abs\_minus1** | ae(v) |
| **dmm\_delta\_end\_sign** | ae(v) |
| } |  |
| } |  |
| } else { |  |
| if( PredMode[ x0 ][ y0 ] != MODE\_INTRA | | log2CbSize = = Log2MinCbSize ) |  |
| **part\_mode** | ae(v) |
| if( PredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&  log2CbSize >= Log2MinIPCMCUSize &&  log2CbSize <= Log2MaxIPCMCUSize ) |  |
| **pcm\_flag** | ae(v) |
| if( pcm\_flag ) { |  |
| **num\_subsequent\_pcm** | tu(3) |
| NumPCMBlock = num\_subsequent\_pcm + 1 |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : 0 |  |
| if ( enable\_DMM\_flag && log2CbSize <= Log2DMMMaxSize ) { |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| **dmm\_flag[** x0 + i ][ y0+ j ] | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if (dmm\_flag**[** x0 + i ][ y0+ j ] ) |  |
| **dmm\_mode[** x0 + i ][ y0+ j ] | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WFULL ||   dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WFULLDELTA ) ) |  |
| **wedge\_full\_tab\_idx[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WPREDDIR ||   dmm\_mode**[** x0 + i ][ y0+ j ] = = MODE\_DMM\_WPREDDIRDELTA ) ) |  |
| **dmm\_delta\_end\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_delta\_end\_flag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_delta\_end\_abs\_minus1[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( dmm\_delta\_end\_flag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_delta\_end\_sign\_flag**[ x0 **][**y0 **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| DmmDeltaFlag**[**x0 + i **][**y0 + i **]** = ( dmm\_flag**[** x0 + i ][ y0+ j ] && (  dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WFULLDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WPREDDIRDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_WPREDTEXDELTA ||   dmm\_mode**[**x0 + i **][**y0 + i **]**  = = MODE\_DMM\_CPREDTEXDELTA **) )** |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[**x0 + i **][**y0 + i **] )** |  |
| **dmm\_dc\_1\_abs[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] && dmm\_dc\_1\_abs**[**x0 + i **][**y0 + i **]** != 0 ) |  |
| **dmm\_dc\_1\_sign\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] ) |  |
| **dmm\_dc\_2\_abs[**x0 + i **][**y0 + i **]** | ae(v) |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) |  |
| if ( DmmDeltaFlag**[** x0 + i ][ y0+ j ] && dmm\_dc\_2\_abs**[**x0 + i **][**y0 + i **]** != 0 ) |  |
| **dmm\_dc\_2\_sign\_flag[**x0 + i **][**y0 + i **]** | ae(v) |
| } else { |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) { |  |
| if( !dmm\_flag**[** x0 + i ][ y0+ j ] ) |  |
| **prev\_intra\_luma\_pred\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| } |  |
| for( j = 0; j <= pbOffset; j = j + pbOffset ) |  |
| for( i = 0; i <= pbOffset; i = i + pbOffset ) { |  |
| if( !dmm\_flag**[** x0 + i ][ y0+ j ] ) { |  |
| if( prev\_intra\_luma\_pred\_flag[ x0 + i ][ y0+ j ] ) |  |
| **mpm\_idx**[ x0 + i ][ y0+ j ] | ae(v) |
| else { |  |
| **rem\_intra\_luma\_pred\_mode**[ x0 + i ][ y0+ j ] | ae(v) |
| if( rem\_intra\_luma\_pred\_mode[ x0 + i ][ y0+ j ] == 31 ) { |  |
| **edge\_intra\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_intra\_flag[ x0 + i ][ y0+ j ] ) { |  |
| **edge\_start\_left\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| **edge\_start\_position**[ x0 + i ][ y0+ j ] | ae(v) |
| **edge\_count\_minus1**[ x0 + i ][ y0+ j ] | ae(v) |
| for( k = 0; k <= edge\_count\_minus1; k++ ) |  |
| **edge\_code**[k] | ae(v) |
| **edge\_dc\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_flag[ x0 + i ][ y0+ j ] ) { |  |
| **edge\_dc\_1\_abs**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_1\_abs[ x0 + i ][ y0+ j ] != 0 ) |  |
| **edge\_dc\_1\_sign\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| **edge\_dc\_2\_abs**[ x0 + i ][ y0+ j ] | ae(v) |
| if( edge\_dc\_2\_abs[ x0 + i ][ y0+ j ] != 0 ) |  |
| **edge\_dc\_2\_sign\_flag**[ x0 + i ][ y0+ j ] | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| **intra\_chroma\_pred\_mode**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } else { |  |
| if( PartMode = = PART\_2Nx2N ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else if( PartMode = = PART\_2NxN ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS, nCbS / 2 ) |  |
| } else if( PartMode = = PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS ) |  |
| } else if( PartMode = = PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 4 ), nCbS, nCbS \*3 / 4 ) |  |
| } else if( PartMode = = PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0, nCbS, nCbS \*3 / 4 ) |  |
| prediction\_unit( x0, y0 + ( nCbS \* 3 / 4 ), nCbS, nCbS / 4 ) |  |
| } else if( PartMode = = PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0, nCbS /4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS / 4 ), y0, nCbS \*3 / 4, nCbS) |  |
| } else if( PartMode = = PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0, nCbS \*3 / 4, nCbS ) |  |
| prediction\_unit( x0 + ( nCbS \* 3 / 4 ), y0, nCbS / 4, nCbS ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS / 2) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |
| if ( !depth\_flag && ViewId && multi\_view\_residual\_pred\_flag &&  ResidualCbfNonZero && PredMode != MODE\_INTRA) |  |
| **res\_pred\_flag** | ae(v) |
| if ( TextureModeDepth[ x0 >> Log2MinCbSize ][ y0 >> Log2MinCbSize ] = = ctDepth   && split\_coding\_unit\_flag[ x0 ][ y0 ] ) |  |
| coding\_tree( x0, y0, log2CbSize, ctDepth ) |  |
| if( !pcm\_flag ) { |  |
| if( PredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !(PartMode = = PART\_2Nx2N && merge\_flag[x0][y0]) ) |  |
| **no\_residual\_syntax\_flag** | ae(v) |
| if( !no\_residual\_syntax\_flag && !sdc\_flag ) { |  |
| MaxTrafoDepth = ( PredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0 x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| if( sdc\_flag ) { |  |
| SdcNumSegments = 1 |  |
| if( SdcPredMode = = 1 | | SdcPredMode = = 2 ) |  |
| SdcNumSegments = 2 |  |
| for ( i = 0; i < SdcNumSegments ; i++ ) { |  |
| **sdc\_residual\_flag**[ x0 ][ y0 ][ i ] | ae(v) |
| if( sdc\_residual\_flag ) { |  |
| **sdc\_residual\_sign\_flag**[ x0 ][ y0 ]**[** i **]** | ae(v) |
| DltDepthBits = BitDepthY |  |
| if( use\_dlt\_flag ) |  |
| DltDepthBits = Floor( Log2( num\_depth\_values\_in\_dlt ) ) |  |
| for ( j = 0; j < DltDepthBits ; j++ ) |  |
| **sdc\_residual\_mag**[ x0 ][ y0 ]**[** i **][** j **]** | ae(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

**sdc\_flag** equal to 0 specifies that simplified depth coding is not used. sdc\_flag equal to 1 specifies that simplified depth coding is used. If sdc\_flag is equal to 1, PartMode is set equal to PART\_2Nx2N.

**sdc\_most\_probable\_mode\_flag** equal to 1 specifies that the currently most probable SDC prediction mode (SdcPredMode) is to be used. sdc\_most\_probable\_mode\_flag equal to 0 specifies that the currently most probable prediction mode is not to be used, SdcPredMode is incremented by 1 and another sdc\_most\_probable\_mode\_flag is to be parsed. This procedure is continued until sdc\_most\_probable\_mode\_flag equals 1.

**Interpretation of sdc\_pred\_mode**

|  |  |
| --- | --- |
| SdcPredMode | Associated Intra Prediction Mode |
| 0 | Intra\_DC |
| 1 | Intra\_Planar |
| 2 | MODE\_DMM\_WFULL |
| 3 | MODE\_DMM\_WPREDDIR |

**sdc\_residual\_flag**[ x0 ][ y0 ][ i ] equal to 0 specifies that the residual is zero for segment i. sdc\_residual\_flag equal to 1 specifies that the residual is non-zero and the sdc\_residual\_sign\_flag and sdc\_residual\_mag[ i ] syntax elements are present for segment i.

**sdc\_residual\_sign\_flag**[ x0 ][ y0 ]**[** i **]** equal to 0 specifies that the residual for segment i is non-negative. sdc\_residual\_sign\_flag equal to 1 specifies that the residual for segment i is negative.

**sdc\_residual\_mag\_minus1**[ x0 ][ y0 ]**[** i **][** j **]** specifies the value of the bit at position j of magnitude minus 1 for segment i of the residual value with most significant bit first.

**sdc\_residual\_mag\_minus1**[ x0 ][ y0 ]**[** i **][** j **]** and **sdc\_residual\_sign\_flag**[ x0 ][ y0 ]**[** i **]** are used to derive sdc\_residual[ x0 ][ y0 ][ i ] for segment i as follows:

SdcResidual[ x0 ][ y0 ][i] = ( 1 − 2 \* **sdc\_residual\_sign\_flag**[ x0 ][ y0 ][ i ] ) \* ( **sdc\_residual\_mag\_minus1**[ x0 ][ y0 ][ i ] + 1)

# Decoding Process

## Depth Lookup Table Decoding Process

Outputs of this process are:

* a lookup table Idx2DepthValue[ i ] specifying the mapping of lookup table indexes to depth values with i = 0..num\_depth\_values\_in\_dlt – 1
* an inverse lookup table DepthValue2Idx[ d ] specifying the mapping of depth values d to lookup table indexes with d = 0..BitDepthY – 1

- If use\_dlt\_flag is equal to 1, the depth lookup table is derived as the following ordered steps.

1. For i = 0..num\_depth\_values\_in\_dlt – 1 the initial depth values in the lookup table are derived as:
   * Set Idx2DepthValue[ i ] equal to dlt\_depth\_value[ i ]
2. For d = 0..BitDepthY – 1 the inverse lookup table DepthValue2Idx[ d ] is derived as the following in ordered steps:
3. Set IdxL equal to 0
4. Set IdxU equal to num\_depth\_values\_in\_dlt – 1
5. For iL = 1..num\_depth\_values\_in\_dlt – 1, the lower DLT index IdxL is derived as follows:

* If Idx2DepthValue[ iL ] > d, the following applies in ordered steps:

1. IdxL = iL - 1
2. End the enclosing for loop
3. For iU = num\_depth\_values\_in\_dlt – 2..0, the upper DLT index IdxU is derived as follows:

* If Idx2DepthValue[ iU ] < p, the following applies in ordered steps:
* IdxU = iU + 1
* End the enclosing for loop

1. If | d - Idx2DepthValue[ IdxLL ] | < | d - Idx2DepthValue[ IdxU ] |, the following applies:

DepthValue2Idx[ d ] = IdxL

1. Otherwise, the following applies:

DepthValue2Idx[ d ] = IdxU

## Decoding Process for Simplified Depth Coding

### Derivation process for luma intra prediction mode

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current picture.

Output of this process is:

* a variable IntraPredMode[ xB ][ yB ] specifying the prediction mode for the current prediction block

IntraPredMode[ xB ][ yB ] labelled 0..34 represents directions of predictions as illustrated in Figure 8-1.

If sdc\_flag[ xB ][ yB ] is equal to 1, IntraPredMode[ xB ][ yB ] is derived as follows:

* If SdcPredMode[ xB ][ yB ] is equal to 0, IntraPredMode[ xB ][ yB ] is set equal to 1
* If SdcPredMode[ xB ][ yB ] is equal to 1, IntraPredMode[ xB ][ yB ] is set equal to 0
* If SdcPredMode[ xB ][ yB ] is equal to 2, IntraPredMode[ xB ][ yB ] is set equal to 35
* If SdcPredMode[ xB ][ yB ] is equal to 3, IntraPredMode[ xB ][ yB ] is set equal to 41

### Depth value reconstruction process

Inputs to this process are:

* a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current picture,
* a variable nT specifying the prediction size
* predicted samples predSamples[ x, y ], with x, y =0..nT−1
* a variable IntraPredMode[ xB ][ yB ] specifying the prediction mode of the current prediction block
* a variable Idx2DepthValue[ i ] specifying the mapping of lookup table indexes to depth values with i = 0..num\_depth\_values\_in\_dlt – 1, which is only needed if **use\_dlt\_flag** equal to 1
* a variable DepthValue2Idx[ d ] specifying the mapping of depth values d to lookup table indexes with d = 0..BitDepthY – 1, which is only needed if **use\_dlt\_flag** equal to 1

Output of this process is:

* reconstructed depth value samples p[ x, y ], with x, y = −1..2\*nT−1.

If IntraPredMode[ xB ][ yB ] is not equal to 0 and IntraPredMode[ xB ][ yB ] is not equal to 1, a binary segmentation pattern SdcSegmentationPattern[ x, y ] is derived as follows:

* A binary segmentation mask sdcSegmentationPattern [ x, y ], with x, y =0..nT−1, is derived by wedgelet list lookup, using wedge\_full\_tab\_idx[ x0 ][ y0 ]. Given the list of wedgelet patterns dmmWedgeletPatternList[ k ] for transform size nT and with k = 0..nWP entries, the following applies:

SdcSegmentationPattern = dmmWedgeletPatternList[wedge\_full\_tab\_idx[ x0 ][ y0 ]]

Otherwise, the binary segmentation pattern SdcSegmentationPattern[ x, y ] is derived as follows:

* SdcSegmentationPattern[ x, y ] is set equal to 0 for all x, y =0..nT−1

For x, y = 0..nT-1, the reconstructed depth value samples p[ x, y ] are derived as the following ordered steps:

1. SdcIdx = SdcSegmentationPattern[ x, y ]
2. If **use\_dlt\_flag** equal to 0, the reconstructed depth values are derived as follows:

* p[ x, y ] = predSamples[ x, y ] + SdcResidual[ xB ][ yB ][SdcIdx ]

Otherwise ( **use\_dlt\_flag** equal to 1 ), the reconstructed depth values are derived as the following ordered steps:

1. DltIdxPred = DepthValue2Idx[ predSamples[ x, y ] ]
2. DltIdxResi = SdcResidual[ xB ][ yB ][SdcIdx ]
3. p[ x, y ] = Idx2DepthValue[ DltIdxPred + DltIdxResi ]

# Details on Implementation

The proposed algorithm is implemented into the 3DV-HTM reference software 4.0.1. All modifications to the source code are encapsulated by preprocessor statements (RWTH\_SRC/RWTH\_SDC and RWTH\_DLT), which make it possible to (de-) activate the coding tools easily.

The algorithm and its implementation do not interfere with other coding tools of the reference software and can therefore be used in addition to all available tools.

# Complexity Discussion

SDC does not introduce significant additional computational complexity compared to the current 3DV-HTM reference software 4.0.1. Only at the encoder side the SDC coding path needs to be tested as an alternative to the existing intra prediction scheme.   
All the SDC-related algorithms and especially at the decoder-side are very low complex as there is neither a de-quantization nor an inverse transform for SDC-coded blocks required.

# Simulation Results

The DMC simulations were performed according the common test conditions [[8](#Hei11)]. For the All-Intra coding scenario configuration files from Core Experiment 6.h on Depth Map Intra Coding Tools were used.

## Random Access Configuration

### Without the Depth Lookup Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video only | synthesized only | coded & synthesized | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,0% | 0,1% | 0,0% | 105,2% | 96,3% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,0% | 0,1% | 0,0% | 106,0% | 98,1% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,0% | -0,4% | -0,3% | 104,8% | 97,4% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,0% | -0,6% | -0,5% | 103,6% | 97,5% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,0% | -0,3% | -0,1% | 104,6% | 96,9% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,0% | -0,2% | -0,2% | 102,5% | 99,2% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,0% | -0,7% | -0,5% | 103,0% | 94,3% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,0% | -0,1% | -0,1% | 105,3% | 97,3% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,0% | -0,5% | -0,3% | 103,4% | 97,0% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,0%** | **-0,3%** | **-0,2%** | **104,2%** | **97,1%** |

### With the Depth Lookup Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video only | synthesized only | coded & synthesized | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,0% | -0,3% | -0,3% | 105,3% | 98,3% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,0% | -0,3% | -0,3% | 103,9% | 97,2% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,0% | -1,1% | -1,0% | 107,8% | 96,8% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,0% | -0,6% | -0,5% | 104,7% | 99,2% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,0% | -0,4% | -0,3% | 103,2% | 96,7% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,0% | -0,2% | -0,2% | 106,7% | 97,0% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,0% | -0,7% | -0,5% | 108,0% | 97,2% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,0% | -0,6% | -0,5% | 105,7% | 97,4% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,0% | -0,5% | -0,4% | 105,7% | 97,5% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,0%** | **-0,5%** | **-0,4%** | **105,7%** | **97,5%** |

## All Intra Configuration

### Without the Depth Lookup Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video only | synthesized only | coded & synthesized | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,0% | -0,1% | -0,5% | 130,7% | 68,9% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,0% | -0,8% | -1,5% | 128,1% | 70,0% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,0% | -0,4% | -0,6% | 129,5% | 71,4% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,0% | -1,8% | -1,6% | 129,2% | 69,9% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,0% | -1,7% | -1,1% | 132,8% | 67,9% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,0% | -0,6% | -0,5% | 130,9% | 70,2% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,0% | -2,4% | -1,5% | 129,4% | 70,6% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,0% | -0,4% | -0,9% | 129,4% | 70,1% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,0% | -1,6% | -1,2% | 130,6% | 69,7% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,0%** | **-1,1%** | **-1,0%** | **130,1%** | **69,8%** |

### With the Depth Lookup Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video only | synthesized only | coded & synthesized | enc time | dec time |
| Balloons | 0,0% | 0,0% | 0,0% | 0,0% | -1,7% | -1,9% | 119,7% | 68,2% |
| Kendo | 0,0% | 0,0% | 0,0% | 0,0% | -2,5% | -2,9% | 120,1% | 69,4% |
| Newspaper\_CC | 0,0% | 0,0% | 0,0% | 0,0% | -2,2% | -2,0% | 122,6% | 70,8% |
| GT\_Fly | 0,0% | 0,0% | 0,0% | 0,0% | -1,8% | -1,6% | 119,3% | 80,1% |
| Poznan\_Hall2 | 0,0% | 0,0% | 0,0% | 0,0% | -2,9% | -2,1% | 126,2% | 71,4% |
| Poznan\_Street | 0,0% | 0,0% | 0,0% | 0,0% | -0,6% | -0,5% | 122,8% | 68,4% |
| Undo\_Dancer | 0,0% | 0,0% | 0,0% | 0,0% | -2,4% | -1,5% | 121,4% | 69,9% |
| 1024x768 | 0,0% | 0,0% | 0,0% | 0,0% | -2,1% | -2,3% | 120,8% | 69,5% |
| 1920x1088 | 0,0% | 0,0% | 0,0% | 0,0% | -1,9% | -1,4% | 122,4% | 72,5% |
| **average** | **0,0%** | **0,0%** | **0,0%** | **0,0%** | **-2,0%** | **-1,8%** | **121,7%** | **71,2%** |

# Cross Check

The cross check of the proposed intra coding tool was done by LG Electronics. 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 of the decoded texture and depth components perfectly match with those presented in this document.

In terms of synthesized views, the PSNR values of LG’s simulations are slightly higher than our own results. There seems to be a parameter issue with the view synthesis part on the side of RWTH Aachen University, which also occurred in another set of simulations for the cross check of LG’s proposal to CE6.  
This minimal difference in synthesis-PSNR values does not have an impact on the overall coding performance of the proposed Simplified Depth Coding tool.

# Conclusion

In this contribution an alternative intra-coding scheme for depth maps is proposed. The proposed Simplified Depth Coding method removes ringing artifacts as known from transform-based coding by directly signaling the pixel-domain information in the bitstream. The piecewise smooth regions of depth data are predicted by either constant depth values or by a planar mode, which is able to model depth gradients. In the subsequent residual coding step the proposed Depth Lookup Table maps the residual DC values for each segment to residual indices, which are then entropy coded.

# References

x

|  |  |
| --- | --- |
| [1] | P. and Smolic, A. and Muller, K. and Wiegand, T. Merkle, "Multi-view video plus depth representation and coding," *IEEE International Conference on Image Processing*, 2007. |
| [2] | F. Jäger, "Contour-based segmentation and coding for depth map compression," *Proceedings of IEEE Visual Communications and Image Processing*, 2011. |
| [3] | M. and Zia, W. and Diepold, K. Sarkis, "Fast Depth Map Compression and Meshing with Compressed Tritree," *Computer Vision*, 2010. |
| [4] | Y. and de With, P.H.N. and Farin, D. Morvan, "Platelet-based coding of depth maps for the transmission of multiview images," *Proceedings of SPIE Stereoscopic Displays and Applications*, 2006. |
| [5] | S. and Lai, P.L. and Tian, D. and Gomila, C. and Chen, C.W. Liu, "Sparse dyadic mode for depth map compression," *Proceedings of IEEE International Conference on Image Processing*, 2010. |
| [6] | Gerhard Tech et al., "Tentative working draft text for the HEVC compatible 3D video coding extension," JCT-3V, Stockholm, Sweden, JCT3V-A0012, 2012. |
| [7] | MPEG Video and Requirement Groups, "Call for Proposals on 3D Video Coding Technology," *MPEG output document N12036*, March 2011. |
| [8] | Dmytro Rusanovskyy Heiko Schwarz, "Common Test Conditions for HEVC- and AVC-based 3DV," ISO/IEC JTC1/SC29/WG11 MPEG, N12560 2012. |

x

# Patent rights declaration(s)

**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).**