|  |  |
| --- | --- |
| **Joint Collaborative Team on 3D Video Coding Extensions**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  7th Meeting: San José, US, 11–17 Jan. 2014 | Document: JCT3V-I0111\_r1 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **AHG7: Complexity assessment of inter-prediction tools** | | |
| *Status:* | Input Document to JCT-3V | | |
| *Purpose:* | Information | | |
| *Author(s) or Contact(s):* | Tomohiro Ikai  1-9-2 Nakase, Mihama-ku, Chiba-shi, Chiba 261-8520  JAPAN | Tel: Email: | +81-43-299-8526 [ikai.tomohiro@sharp.co.jp](mailto:ikai.tomohiro@sharp.co.jp) |
| *Source:* | Sharp Corporation | | |

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

# Abstract

This contribution reports a complexity assessment of inter prediction tools in terms of number of operation and memory bandwidth. This report has changes from the former report of JCT3V-G0208 in the following:

1. Include memory bandwidth analysis instead of data transfer rate analysis
2. Include depth access for memory bandwidth calculation
3. Consider memory pattern for memory bandwidth calculation
4. Add DBBP to the target tools

It is recommended to keep complexity in check and to improve the balance between performance and complexity until 3D-HEVC finalization.

Revision 1 fixed the number of operation for DBBP by adding contour derivation process and assuming common Cb / Cr decision be one decision instead of two decisions.

# Introduction

This complexity analysis includes depth access. Please recognize the following assumption about depth access.

**About depth access**

DoNBDV, VSP and DBBP needs four corner depth of CU, four corner depth of 4x8/8x4 subblock and whole depth respectively. While some implementation may fetch four corner depth of CU for DoNBDV, it can be safe to say that some implementation may fetch whole depth pixel of the CU which can be used in VSP and DBBP.

Therefore, partial depth access case (partial depth access) and full depth access case for DoNBDV are investigated.

# Results of complexity analisys

In terms of number of operations, the followings are observed.

* MC, ARP and VSP doesn’t increase number of operations
* IC shows 105 % number of operations
* DBBP shows 138 % number of operations

In terms of number of operations, the followings are observed.

* MC and IC shows 101% to 106% (110 % to 114 %) memory bandwidth
* ARP shows 122 % to 164 % (132 % to 172 %) memory bandwidth
* DBBP shows 111 % to 119 % (120 % to 128 %) memory bandwidth
* VSP doesn’t increase memory bandwidth

Table 1. Summary of complexity

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Number of operation [%] | | Data transfer rate [%] | | | | | | | |
| 3D-HEVC |  | Mult [%] | Add [%] | Partial depth access  in DoNBDV | | | | Full depth access  in DoNBDV | | | |
|  |  |  | 1x1 | 4x2 | 4x4 | 8x2 | 1x1 | 4x2 | 4x4 | 8x2 |
| MC | 100% | 100% | 101% | 103% | 105% | 106% | 110% | 113% | 111% | 114% |
| ARP | 46% | 38% | 122% | 138% | 133% | 164% | 132% | 148% | 139% | 172% |
| IC | 105% | 103% | 101% | 103% | 105% | 106% | 110% | 113% | 111% | 114% |
| VSP | 18% | 17% | 41% | 60% | 60% | 78% | 50% | 70% | 65% | 94% |
| DBBP | 100% | 140% | 111% | 116% | 116% | 119% | 120% | 125% | 123% | 128% |

Table 2. Summary of complexity (Partial depth access in DoNBDV, 1x1 memory pattern)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3D-HEVC | |  | CU | PU | Mult [%] | Add [%] | TotalBandwidth [%] |
| MC | 64x64 | 64x64 | 74% | 75% | 36% |
| 32x32 | 32x32 | 78% | 78% | 43% |
| 16x16 | 16x16 | 85% | 86% | 60% |
| 8x8 | 8x8 | 100% | 100% | 101% |
| Sub-block (8x8) | 64x64 | 64x64 | 100% | 100% | 100% |
| 32x32 | 32x32 | 100% | 100% | 100% |
| 16x16 | 16x16 | 100% | 100% | 100% |
| 8x8 | 8x8 | 100% | 100% | 101% |
| ARP | 64x64 | 64x64 | 43% | 36% | 93% |
| 32x32 | 32x32 | 43% | 37% | 97% |
| 16x16 | 16x16 | 44% | 37% | 105% |
| 8x8 | 8x8 | 46% | 38% | 122% |
| IC | 64x64 | 64x64 | 79% | 78% | 36% |
| 32x32 | 32x32 | 83% | 82% | 43% |
| 16x16 | 16x16 | 91% | 90% | 60% |
| 8x8 | 8x8 | 105% | 103% | 101% |
| VSP | 64x64 | 64x64 | 18% | 17% | 41% |
| 32x32 | 32x32 | 18% | 17% | 41% |
| 16x16 | 16x16 | 18% | 17% | 41% |
| 8x8 | 8x8 | 18% | 17% | 41% |
| DBBP | | 64x64 | 64x64 | 74% | 115% | 46% |
| 32x32 | 32x32 | 78% | 118% | 53% |
| 16x16 | 16x16 | 85% | 126% | 70% |
| 8x8 | 8x8 | 100% | 140% | 111% |

Table 3. Summary of complexity (Full depth access in DoNBDV, 8x2 memory pattern)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3D-HEVC |  | CU | PU | Mult [%] | Add [%] | TotalBandwidth [%] |
| MC | 64x64 | 64x64 | 74% | 75% | 30% |
| 32x32 | 32x32 | 78% | 78% | 39% |
| 16x16 | 16x16 | 85% | 86% | 59% |
| 8x8 | 8x8 | 100% | 100% | 114% |
| Sub-block (8x8) | 64x64 | 64x64 | 100% | 100% | 106% |
| 32x32 | 32x32 | 100% | 100% | 107% |
| 16x16 | 16x16 | 100% | 100% | 109% |
| 8x8 | 8x8 | 100% | 100% | 114% |
| ARP | 64x64 | 64x64 | 43% | 36% | 67% |
| 32x32 | 32x32 | 43% | 37% | 80% |
| 16x16 | 16x16 | 44% | 37% | 107% |
| 8x8 | 8x8 | 46% | 38% | 172% |
| IC | 64x64 | 64x64 | 79% | 78% | 30% |
| 32x32 | 32x32 | 83% | 82% | 39% |
| 16x16 | 16x16 | 91% | 90% | 59% |
| 8x8 | 8x8 | 105% | 103% | 114% |
| VSP | 64x64 | 64x64 | 18% | 17% | 73% |
| 32x32 | 32x32 | 18% | 17% | 74% |
| 16x16 | 16x16 | 18% | 17% | 76% |
| 8x8 | 8x8 | 18% | 17% | 81% |
| DBBP | 64x64 | 64x64 | 74% | 115% | 36% |
| 32x32 | 32x32 | 78% | 118% | 46% |
| 16x16 | 16x16 | 85% | 126% | 69% |
| 8x8 | 8x8 | 100% | 140% | 128% |

# References

[1] T. Ikai, “AHG7: Complexity assessment of motion compensation process”, JCT3V-G0208, JCT3V 7th Meeting: San José, US, 11–17 Jan. 2014

# Appendix

About counting number of operations in DBBP

**Current HTM (luma): Add = NxNx8 comparison + NxNx7 addiction**

**Current HTM (chroma): Add = N/2xN/2x10 comparison + N/2xN/2x10 addiction**

**Contour part(luma only): NxNx1 comparison NxNx1 addiction**

The values of the binary partition pattern contourPattern[ x ][ y ], with x, y = 0..nTbS − 1, are derived as specified by the following ordered steps:

* 1. The variable threshVal specifying a threshold for the segmentation of refSamples is derived as specified in the following:
     + The variable sumRefVals is set equal to 0.
     + For x = 0..nTbS − 1 and y = 0..nTbS − 1, when ( x % sampInt ) and ( y % sampInt ) are both equal to 0, the following applies:
       - 1. sumRefVals += refSamples[ x ][ y ] (I‑296)
     + The variable threshVal is set equal to ( sumRefVals >> ( 2 \* log2( nTbS / sampInt ) ) )
  2. The variable contourPattern[ x ][ y ] with x, y =0..nTbS − 1 specifying a binary partition pattern is derived as specified in the following:
     + For x = 0..nTbS − 1 and y = 0..nTbS − 1, the following applies:
       - 1. contourPattern[ x ][ y ] = ( refSamples[ x ][ y ] > threshVal ) (I‑297)

**DBBP blending part(luma): (**NxN comparison)

**DBBP blending part(chroma): (**2\*N/2xN/2 comparison)

for ( y = 0; y < nCbSL; y++ )  
 for( x = 0; x < nCbSL; x++ ) {  
 if( segMask[ x ][ y ] = = ( partIdx ! = segMask[ 0 ][ 0 ] ) )  
 PredSamplesDbbpL[ x ][ y ] = predSamplesL[ x ][ y ]  
 if( ( x % 2 = = 0 ) && ( y % 2 = = 0 ) ) {  
 PredSamplesDbbpCb[ x / 2 ][ y / 2 ] = predSamplesCb[ x / 2 ][ y / 2 ]  
 PredSamplesDbbpCr[ x / 2 ][ y / 2 ] = predSamplesCr[ x / 2 ][ y / 2 ]  
 }  
 }

**DBBP Filtering part (luma):** NxNx4+NxNx2 comparison NxNx4+NxNx2 addiction

**DBBP Filtering part (chroma):** N/2xN/2x4+2\*N/2xN/2x2 comparison N/2xN/2x2+2\*N/2xN/2x4 addiction

Note: (tFlag | | cFlag | | bFlag) can be replaced with comparison. Since comparison is probably more practical and there are no columns for logical operations, I used comparison for this operation.

for ( y = 0; y < nCbSX; y++ )  
 for( x = 0; x < nCbSX; x++ ) {  
 tFlag = segMask[ n \* x ][ Max( 0, n \* ( y − 1 ) ) ]  
 lFlag = segMask[ Max( 0, (n \* ( x − 1 ) ) ][ n \* y ]  
 bFlag = segMask[ n \* x ][ Min( n \* ( y + 1 ), nCbSL − 1 ) ]  
 rFlag = segMask[ Min( n \* ( x + 1 ), nCbSL − 1 ) ][ n\*y ]  
 cFlag = segMask[ n \* x ][ n \* y ]  
 filt = p[ x ][ y ]  
 ~~if( ( lFlag | | cFlag | | rFlag ) && ( !lFlag | | !cFlag | | !rFlag ) )~~  
 if( ( lFlag - cFlag ) + ( cFlag - rFlag ) )  
 filt = ( p[ Max( 0, x − 1 ) ][ y ] + ( filt << 1 ) + p[ Min( x + 1, nCbSX − 1 ) ][ y ] ) >> 2

~~if( ( tFlag | | cFlag | | bFlag ) && ( !tFlag | | !cFlag | | !bFlag ) )~~  
 if( ( tFlag - cFlag ) + ( cFlag - bFlag ) )

filt = ( p[ x ][ Max( 0, y − 1 ) ] + ( filt << 1 ) + p[ x ][ Min( y + 1, nCbSX − 1 ) ] ) >> 2 predSamples[ x ][ y ] = filt  
 }