|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  11th Meeting: Shanghai, CN, 10–19 Oct. 2012 | Document: JCTVC-K0106 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **On Tile Processing Order** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Pieter Kapsenberg (Intel Corporation)  Wenhao Zhang (Intel Corporation)  CH7-427 5000 W Chandler Blvd Chandler AZ, 85226 | Tel: Email: | +1 480 552 5481 pieter.kapsenberg@intel.com |
| *Source:* | Intel Corp | | |

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

# Abstract

This contribution claims that the current raster ordering of tiles within a picture poses a minor but unnecessary burden on hardware decoders. A solution is proposed that has no coding performance impact and reduces memory footprint; it is suggested that tiles should be processed column by column instead of row by row (vertical raster). This contribution includes a suggested HM-8.0 patch.

# Introduction

In a CU-level decoder, line buffers are used to store data that is needed when processing the coding tree block row below the current row. Due to the raster-scan order of coding tree blocks within a tile (or single-tile picture), these line buffers have the convenient property that every item written into it will be retrieved in the same order; it behaves exactly like a FIFO (first in, first out). This property allows for simple and efficient implementation of these line buffers.

However the current ordering of the tiles themselves actually breaks this property. In the current draft standard, tiles are ordered in the bitstream in raster scan. Example for a picture with 4 rows and 3 columns of tiles:



Figure 1 – Raster order tiles

In this example, data that is written to a line buffer on the bottom edge of tile 0 will not be needed until the top of tile 3 is processed. Between the bottom of tile 0 and the top of tile 3, the line buffer will need to be written to and read from to process tiles 1 and 2. Thus the FIFO-like nature of the line buffer is not present.

At least three non-normative solutions to this problem exist:

* Explicitly address each read and write to the line buffer, forgoing the FIFO design. This has the disadvantage of requiring address generation logic and odd-sized data can no longer be packed as the system memory granularity now is a concern (e.g. all transactions must be 128 bits).
* Use a separate line buffer for tile boundaries and tree block row boundaries. This has the disadvantage of requiring two buffers instead of one, duplicating logic.
* Process tiles in vertical raster order. The disadvantage here is that the entropy decoder must skip around in the bitstream, or the bitstream substreams must be re-ordered.

# Proposed solution: Vertical raster order of tiles

By ordering the tiles within the bitstream in vertical raster scan, a single-core decoder can use the same line buffer for the entire picture regardless of the number of tiles, without having to skip around in the bitstream:



Figure 2 – Vertical raster order tiles

This solution has the following two consequences:

* The size of the minimum horizontal line buffer needed is now a multiple of the widest tile, whereas before it was a multiple of picture width.
* The size of the minimum vertical column buffer needed is now a multiple of picture height, whereas before it was a multiple of the tallest tile.

The proposed solution results in a smaller overall memory footprint for pictures that are wider than they are tall, which is true for nearly all common formats. Additionally, X/Y treeblock counter update logic is simplified since the dependency on the height of tile rows has been removed (i.e. it always behaves as if there is only one row of tiles). Coding efficiency is not affected, and existing bitstreams can be converted without requiring re-encoding.

# Simulations results

Note: simulation results were run with the suggested fix in HM ticket #748 (<http://hevc.kw.bbc.co.uk/trac/ticket/748>). No change in coding performance is observed, as expected. Encode/decode times are not meaningful due to the unpredictable load on the compute clusters.

Only Class C and D 8-bit sequences were run, since this proposal does not impact coding performance.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **All Intra Main** | | | **All Intra HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B |  |  |  |  |  |  |
| Class C | 0.0% | 0.0% | 0.0% |  |  |  |
| Class D | 0.0% | 0.0% | 0.0% |  |  |  |
| Class E |  |  |  |  |  |  |
| **Overall** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Class F |  |  |  |  |  |  |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Random Access Main** | | | **Random Access HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B |  |  |  |  |  |  |
| Class C | 0.0% | 0.0% | 0.0% |  |  |  |
| Class D | 0.0% | 0.0% | 0.0% |  |  |  |
| Class E |  |  |  |  |  |  |
| **Overall** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Class F |  |  |  |  |  |  |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |
|  |  |  |  |  |  |  |
|  | **Low delay B Main** | | | **Low delay B HE10** | | |
|  | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |
| Class B |  |  |  |  |  |  |
| Class C | 0.0% | 0.0% | 0.0% |  |  |  |
| Class D | 0.0% | 0.0% | 0.0% |  |  |  |
| Class E |  |  |  |  |  |  |
| **Overall** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Class F |  |  |  |  |  |  |
| Enc Time[%] |  | | |  | | |
| Dec Time[%] |  | | |  | | |

# Draft Text changes for Solution

**6.5.1 Coding tree block raster and tile scanning conversion process**

<snip>

The list ctbAddrRStoTS is derived as follows:

for( ctbAddrRS = 0; ctbAddrRS < PicSizeInCtbsY; ctbAddrRS++ ) {  
 tbX = ctbAddrRS % PicWidthInCtbsY  
 tbY = ctbAddrRS / PicWidthInCtbsY  
 for( i = 0; i <= num\_tile\_columns\_minus1; i++ )  
 if( tbX >= colBd[ i ] )  
 tileX = i  
 **~~for( j = 0; j <= num\_tile\_rows\_minus1; j++ )  
 if( tbY >= rowBd[ j ] )  
 tileY = j~~** ctbAddrRStoTS[ ctbAddrRS ] = tbY \* colWidth[ tileX ]  
 for( i = 0; i < tileX; i++ )  
 ctbAddrRStoTS[ ctbAddrRS ] += PicHeightInCtbsY \* colWidth[ i ]  
 **~~for( j = 0; j < tileY; j++ )  
 ctbAddrRStoTS[ ctbAddrRS ] += PicWidthInCtbsY \* rowHeight[ j ]~~** ctbAddrRStoTS[ ctbAddrRS ] += tbX − colBd[ tileX ]  
 }

<snip>  
  
The list tileId is derived as follows:

for( i = 0, tIdx = 0; i <= num\_tile\_columns\_minus1; i++ )  
 for( j = 0; j <= num\_tile\_rows\_minus1; j++ , tIdx++ )  
 for( y = rowBd[ j ]; y < rowBd[ j + 1 ]; y++ )  
 for( x = colBd[ i ]; x < colBd[ i + 1 ]; x++ )  
 tileId[ ctbAddrRStoTS[ y\*PicWidthInCtbsY+ x ] ] = tIdx

# Patent rights declaration(s)

**Intel Corporation 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).**