

JVT-T087, FGS Complexity Reduction

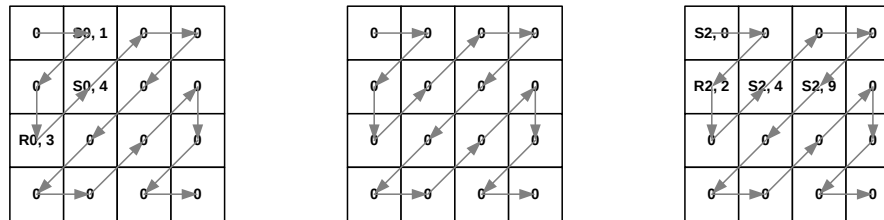
Part of AhG on Complexity Reduction

Yiliang Bao, Yan Ye
Qualcomm Inc.

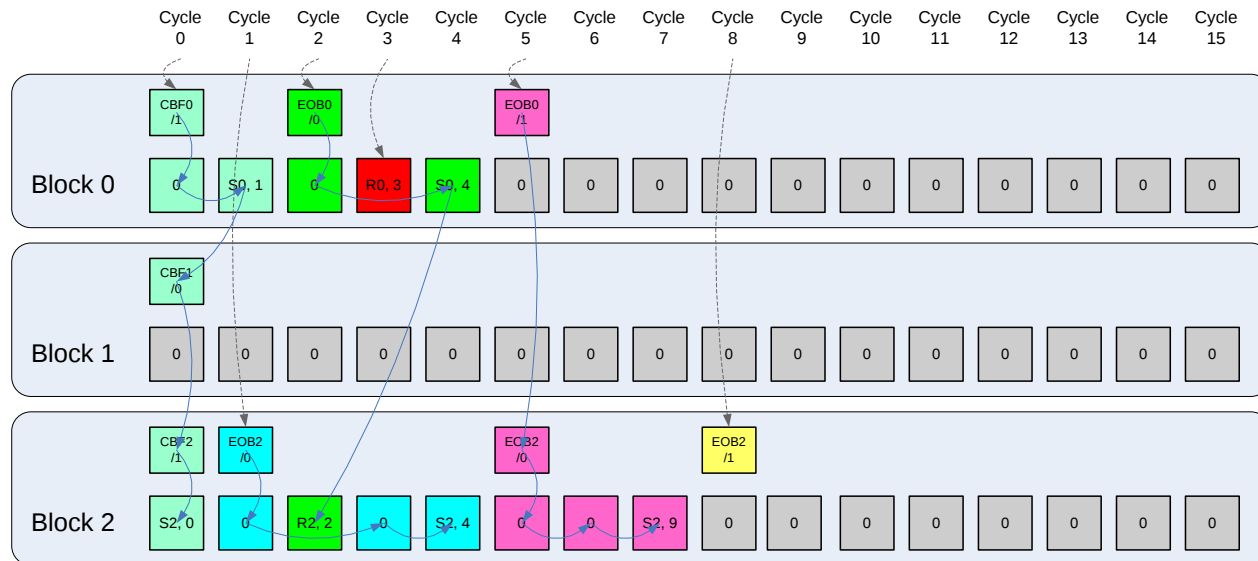
Overview

- This proposal addresses the complexity of FGS coder
- Introduce cycle-aligned fragment to make it possible to performance block-based coding of FGS layer
- Introduce Macroblock header in the FGS slice to simplify syntax specification

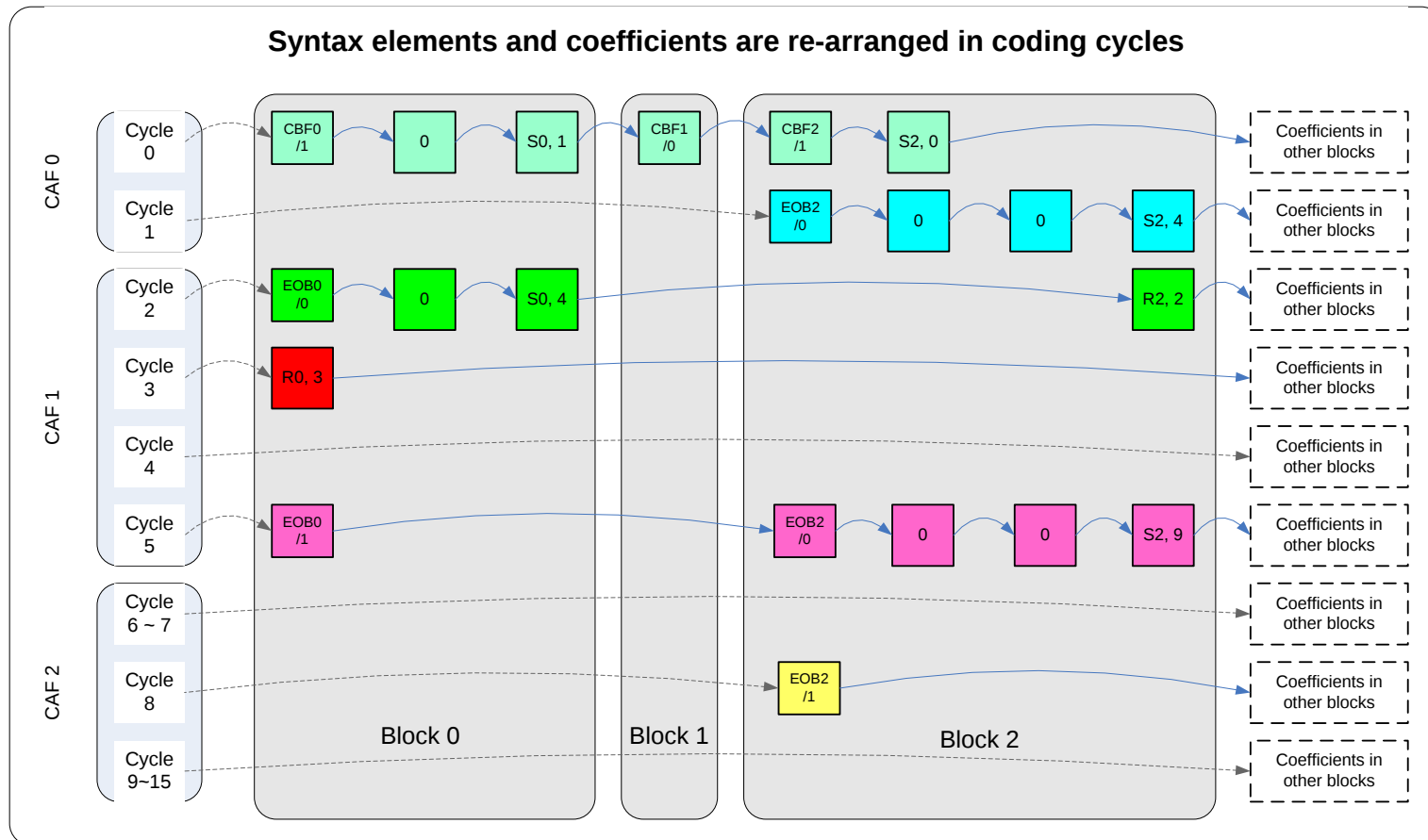
Review of FGS coding in JSVM



Coding order, the syntax elements and coefficients are arranged by blocks in zigzag order



Cycle-Aligned Fragment (CAF)

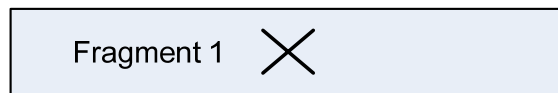
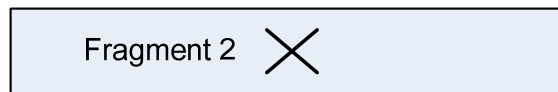


Cycle-Aligned Fragment, more details

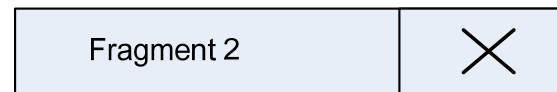
- Introduce fragments which are aligned with the cycle boundaries
 - A fragment is coded with fresh contexts to enable parallel processing of multiple fragments
- Process multiple fragments in parallel to achieve block by block coding
- Vector mode is used to adjust the fragment size and reduce the fragmentation overhead
 - Vector lengths need to be signaled in the slice header
- A natural extension of the following concepts
 - Subband coding order, fragment, and vector mode

CAF improves error resilience

- With normal fragment, error in a fragment will make all the remaining fragments in the same FGS layer un-decodable
- With Cycle-aligned fragment, only the corresponding parts in other fragments are affected



Normal fragment



Cycle-aligned fragment

Macroblock header in the FGS slice

- Currently CBP, MB delta QP, and Transform_8x8 flag are coded in various places
 - It is optimal because they are sent only when they are absolutely needed
- However the bits are shuffled only in the first cycle of the macroblock, the benefit is negligible
- Propose to define macroblock header in the FGS layer to group the motion refinement information and CBP, MB delta QP, and transform_8x8 flag
 - Similar to CGS macroblock header
 - Syntax is much simplified

New FGS Decoding Flow, with CAF & FGS MB Header

```
for( uiMbY = 0; uiMbY < uiLastMbY; uiMbY ++ ) {  
    for( uiMbX = 0; uiMbX < uiLastMbX; uiMbX ++ ) {  
  
        xDecodeMbHeader( pcMbDataAccessBL, pcMbDataAccessEL, iLastQP );  
  
        for( B8x8Idx c8x8Idx; c8x8Idx.isLegal(); c8x8Idx ++ ) {  
            for( S4x4Idx cIdx(c8x8Idx); cIdx.isLegal( c8x8Idx ); cIdx ++ ) {  
                xResidualBlock( *pcMbDataAccessEL, *pcMbDataAccessBL,  
                    cIdx, LUMA_SCAN, aiMaxPosLuma, uiNumFrag, uiMbCoeffsDecoded );  
            }  
        }  
  
        xResidualBlock( *pcMbDataAccessEL, *pcMbDataAccessBL,  
            CIdx(0), CHROMA_DC, aiMaxPosChromaDC, uiNumFrag, uiMbCoeffsDecoded );  
        xResidualBlock( *pcMbDataAccessEL, *pcMbDataAccessBL,  
            CIdx(4), CHROMA_DC, aiMaxPosChromaDC, uiNumFrag, uiMbCoeffsDecoded );  
  
        for( CIdx cCIdx; cCIdx.isLegal(); cCIdx ++ ) {  
            xResidualBlock( *pcMbDataAccessEL, *pcMbDataAccessBL,  
                cCIdx, CHROMA_AC, aiMaxPosChromaAC, uiNumFrag, uiMbCoeffsDecoded );  
        }  
    }  
}
```