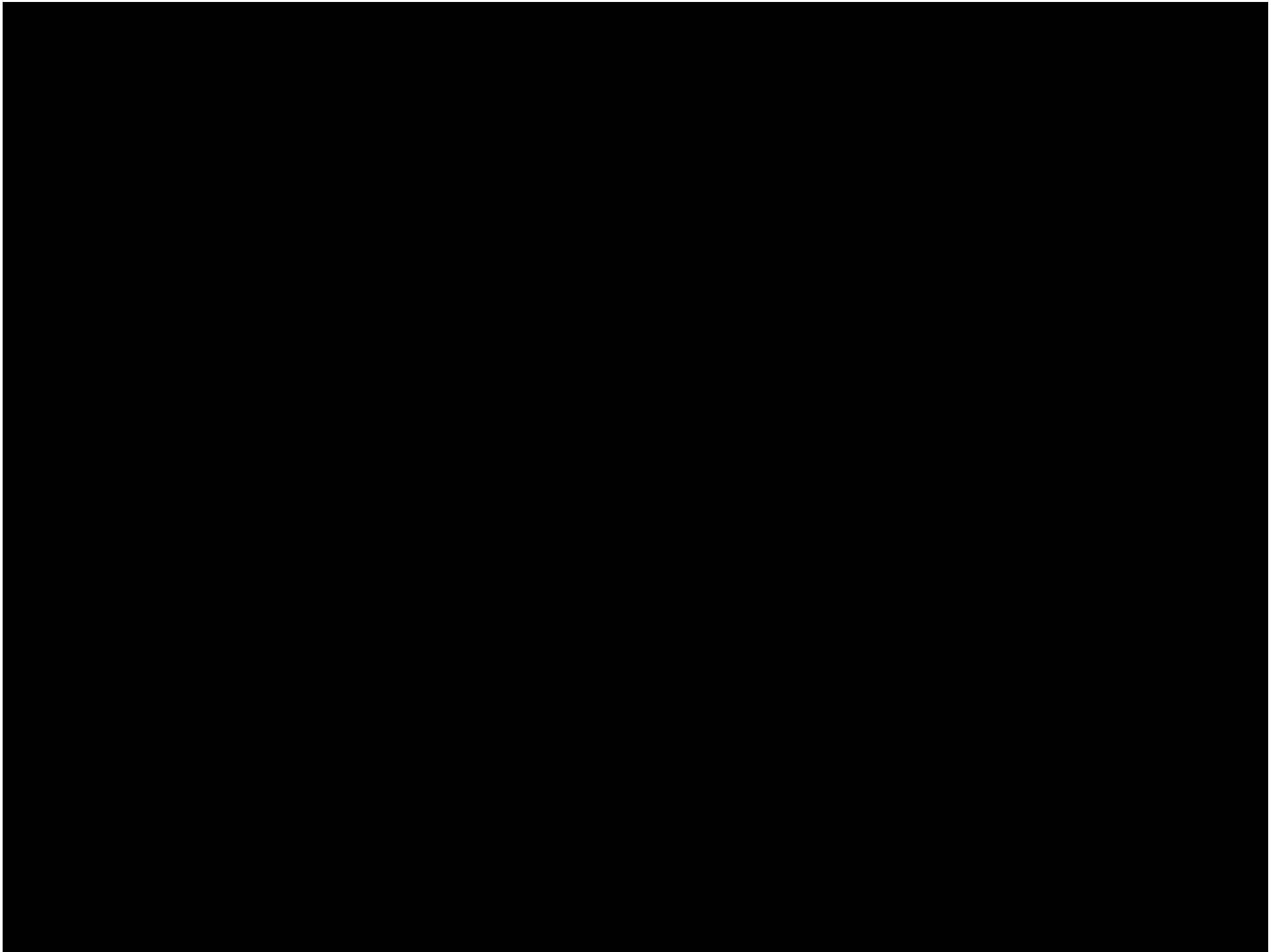




JVT-T067: Modification for Grouping of Refinement Coefficients

Masato Shima, Texas Instruments Japan

Justin Ridge, Nokia Inc.

Yiliang Bao, Qualcomm

Introduction: VLC refinement coding

◆ Before JSVM5

- JVT-Q040 based grouping had been used
- Significance pass and refinement pass were completely separated
- There was no difficulty in grouping and coding refinement symbols

Introduction: VLC refinement coding

◆ At Geneva meeting, 2 related contributions were adopted:

■ JVT-S077

- Introduced new and unified FGS coding scheme

■ JVT-S066

- Applied the new coding scheme to CAVLC
 - interleaving significance and refinement values
- Introduced ternary coding
- Removed “flushing” operations

Introduction: VLC refinement coding

- ◆ Grouping of refinement coefficients
 - got more complex, as significance values and refinement values are interleaved

(Example)

* Symbols, 0, 1, 0, are grouped and coded

Block 0 (Ref)	0
Block 1 (Sig)	...
Block 2 (Ref)	1
Block 3 (Sig)	...
Block 4 (Sig)	...
Block 5 (Ref)	0

← Option 2: group and code at first symbol

← Option 1: group and code at last symbol

Option 1

◆ Grouping and coding at last symbol

■ Encoder side:

- only need to buffer 2 ternary symbols

■ Decoder side:

- need to store the location of the refinement coefficients (most likely memory address)
- need to update the coefficients in the memory when decoding the symbols
 - causes random memory access

■ Considered as “encoder-friendly” solution

Option 1

- ◆ JVT-S066 allows grouping of refinement coefficients regardless of the location
 - a coefficient in the first macroblock of a slice and a coefficient in the last macroblock of the slice could be grouped
 - random memory access issue will make the implementation of VLC refinement coding more complex
 - it could be more complex than CABAC

Option 2

- ◆ Grouping and coding at first symbol
 - Decoder side:
 - only need to buffer 2 ternary symbols
 - Encoder side:
 - need to pre-scan the coefficients of entire subband to check where the last refinement coefficient exists
 - even for encoder side, it should be too complex
 - Considered as “decoder-friendly” solution

Current status

- ◆ Integration of VLC refinement coding adoptions at Geneva
 - All parts except this issue had been finalized and integrated into JSVM5_3 software and JSVM6 text
 - Agreed to find a solution to this issue by this meeting as an activity of the CAVLC AHG

Current status

- ◆ In JSVM software
 - “Encoder-friendly solution” has been integrated
 - leftover of JVT-S066
- ◆ In JSVM text
 - “Decoder-friendly solution” has been integrated
 - Simpler and clearer description on decoder side
- ◆ Some action to this issue is needed anyway!

Proposed Modifications

- ◆ Use the “decoder friendly solution”
 - to minimize the complexity of decoder side
- ◆ Restore ‘flush’ operation at every macroblock boundary of each cycle
 - to retain the complexity increase on encoder side by limiting the length of pre-scanning

Effect of Proposed Solution

- ◆ No random memory access
 - lead to minimize the complexity of decoder
- ◆ Reasonably small complexity increase on encoder side
 - limited to pre-scanning of a macroblock
- ◆ Some bit-rate increase comparing to 'encoder friendly solution'
 - due to the restored 'flush' operation

Coding Performance

- ◆ JSVM5.3 with proposed modification and JSVM5.3 are compared with JSVM5.2
 - To see the performance of completed CAVLC solution
- ◆ JVT-Q009, SNR test condition was used
 - CAVLC, No FRext mode, 3 FGS layers
- ◆ PSNR is unchanged
 - because only VLC has been modified

Coding Performance

Bit-rate difference in (%), (negative numbers are reductions)	JSVM5_3 vs. JSVM5_2	JSVM5_3 with proposed modification vs. JSVM5_2
QCIF 1FGS layer average	-1.87%	-1.24%
QCIF 2FGS layers average	-2.53%	-1.61%
QCIF 3FGS layers average	-2.73%	-1.67%
CIF 1FGS layer average	-3.29%	-2.75%
CIF 2FGS layers average	-3.24%	-2.44%
CIF 3FGS layers average	-2.81%	-1.90%
4CIF 1FGS layer average	-5.39%	-4.96%
4CIF 2FGS layers average	-3.80%	-3.24%
4CIF 3FGS layers average	-2.96%	-2.24%
total average	-3.01%	-2.24%

◆ Gain of JSVM5.3 : about 1.9%~5.4%

◆ W/ proposed solution: about 1.2%~5.0%

Coding Performance

- ◆ With the proposed modification, about 0.5~1.0% of bit-saving is given up
 - Comparing to “encoder friendly solution” in JSVM5.3
 - More details can be seen from JVT-T067.xls
- ◆ Using “vector mode” (adopted at previous meeting) will reduce the bit-rate increase
 - by eventually reducing number of ‘flush’
 - known as another tool for complexity reduction

Conclusion

- ◆ Proposed modifications for grouping of refinement coefficients were presented
- ◆ The proposed solution will:
 - minimize the complexity of decoder
 - by removing random memory access
 - retain the complexity of encoder
 - by limiting the length of pre-scanning
 - at the cost of some bit-rate increase
 - Considering the implementation of CAVLC, removing random memory access is more important than compression efficiency
- ◆ Therefore, we propose its incorporation into JSVM

