Video Compression

INSTRUCTOR: YAN-TSUNG PENG

DEPT. OF COMPUTER SCIENCE, NCCU

Floating-point to Integer DCT

- Floating point operation is expensive
- For example, a 4x4 DCT transform is

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ & \cdots & & \\ & \cdots & & \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.65328 & 0.27060 & 0.27060 & -0.65328 \\ & \cdots & & \\ & \cdots & & \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Make an integer approximation of DCT

Scale up by 26

Integer DCT

- Forward DCT

$$\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix} = \boxed{\begin{bmatrix} 13 & 13 & 13 & 13 \\ 17 & -7 & -7 & -17 \\ 13 & -13 & -13 & 13 \\ 7 & -17 & 17 & -7 \end{bmatrix}} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Scale up by 26

- Backward DCT

cancel up the scaling 26^2

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \boxed{\frac{1}{676}} \boxed{\begin{bmatrix} 13 & 13 & 13 & 13 \\ 17 & -7 & -7 & -17 \\ 13 & -13 & -13 & 13 \\ 7 & -17 & 17 & -7 \end{bmatrix}} \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

Scale up by 26

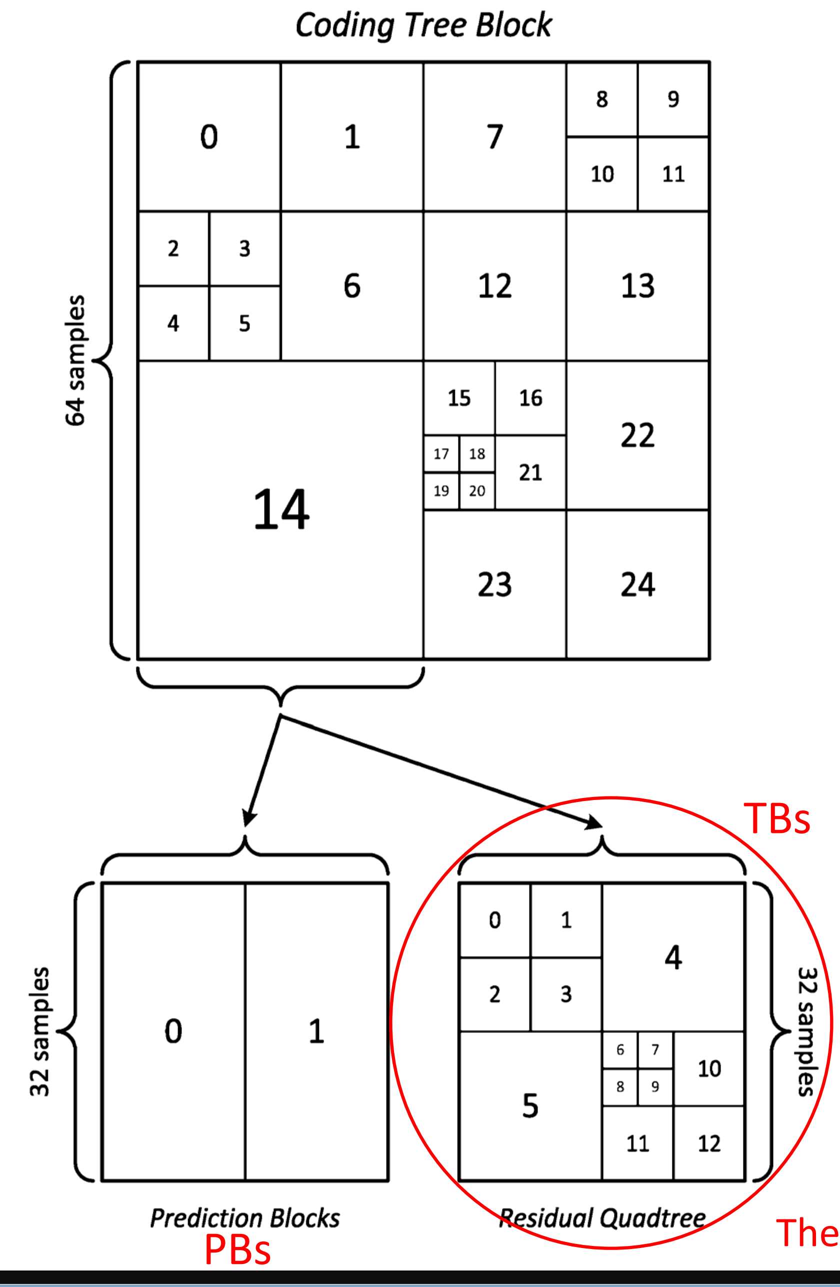
Transform Coding for H.264 and H.265

- In H.264/MPEG-4 AVC
  - Prediction block sizes range from 4x4 to 16x16
  - Transform block sizes are 4x4 and 8x8
- Adopted first by H.265/HEVC, a **Quadtree Structure** is designed for subdividing a frame into different block sizes for prediction and residual coding
  - Prediction block sizes range from 4x4 to 64x64
  - Transform block sizes range from 4x4 to 32x32

Nguyen et al., Transform Coding Techniques in HEVC, *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING*, 2013

Quadtree Structure for Transform Coding

- For HEVC, each frame is partitioned into a grid of square blocks, referred to as *coding tree blocks* (CTBs), each of which represents the root of a quadtree, referred to as *coding quadtree.*
  - A coding tree is like a MB that has three blocks (one luma and two chroma blocks), which consists of coding tree blocks.
- A coding quadtree can be further partitioned into smaller square blocks, called *coding blocks (CB).*
- For each CB as a leaf of a coding quadtree, the prediction mode is determined as *intra* or *inter.*
- For each CB, the coding quadtree is further partitioned into **prediction blocks (PBs) and transform blocks (TBs).**

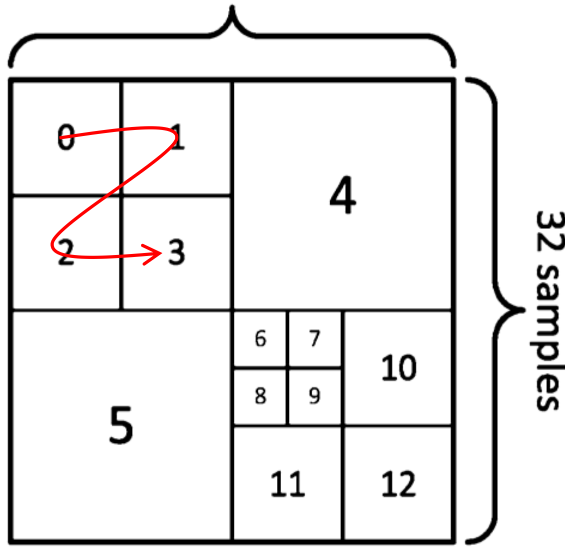


- Different profiles of HEVC have different minimum and maximum CB sizes
- The smallest possible TB size is restricted by 4x4 or the maximum allowed depth of the RQT, while the largest is 32x32

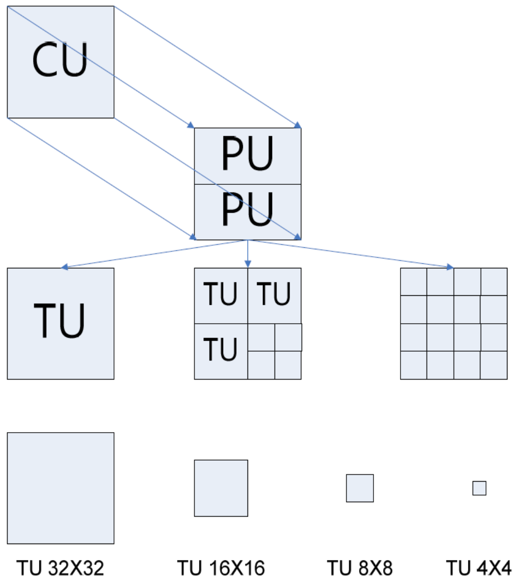
The coding quadtree for TBs is also called the residual quadtree (RQT)

Transform Coefficient Level Coding

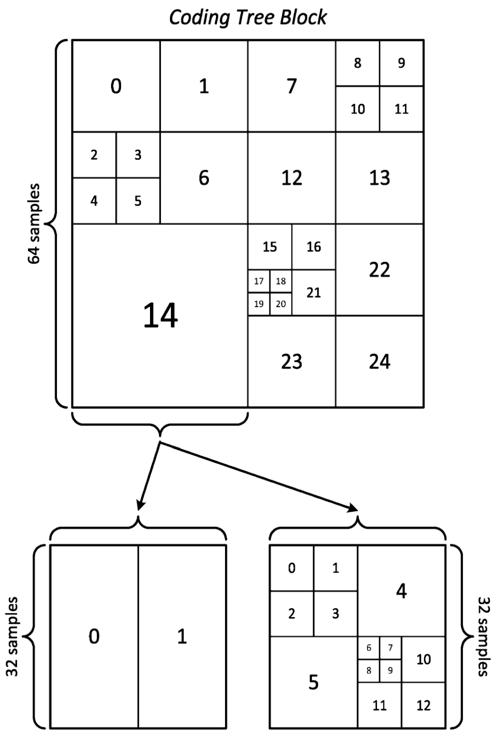
- In HEVC, Transform Coefficient Level Coding improves throughput and coding efficiency
- We traversing the coding quadtree to process CBs in a depth-first manner.



Residual Quadtree



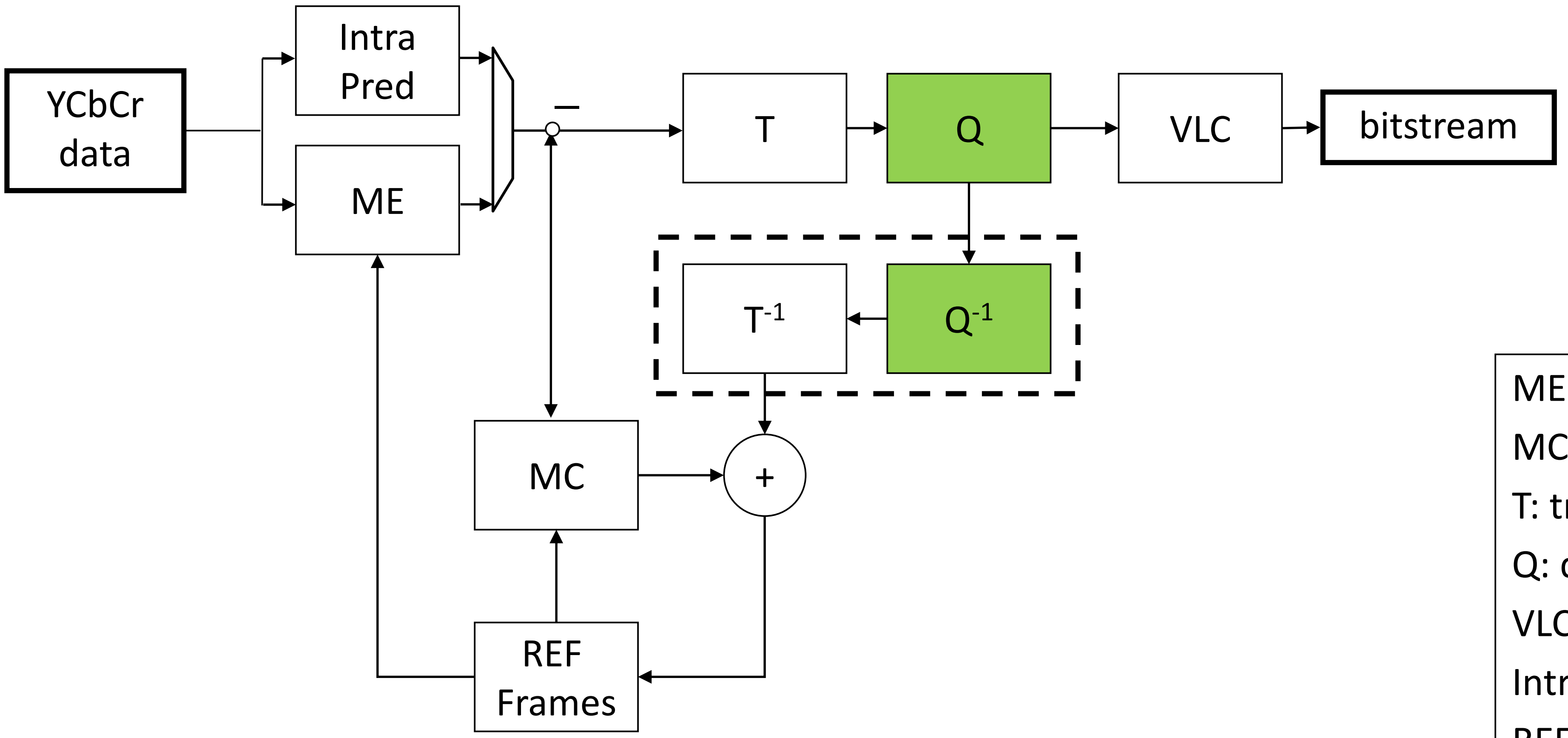
RQT (residual quadtree) structure



Reference and figures Nguyen et al., Transform Coding Techniques in HEVC, *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING*, 2013

Reference and figures: Park *et al.*, 2-D Large Inverse Transform (16x16, 32x32) for HEVC (High Efficiency Video Coding), JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, 2012

Video Encoder Diagram



ME: motion estimation
MC: motion compensation
T: transform coding
Q: quantization
VLC: variable length coding
Intra Pred: Intra prediction
REF frame: reference frame

Quantization Process (MPEG4)

- A common quantization process:

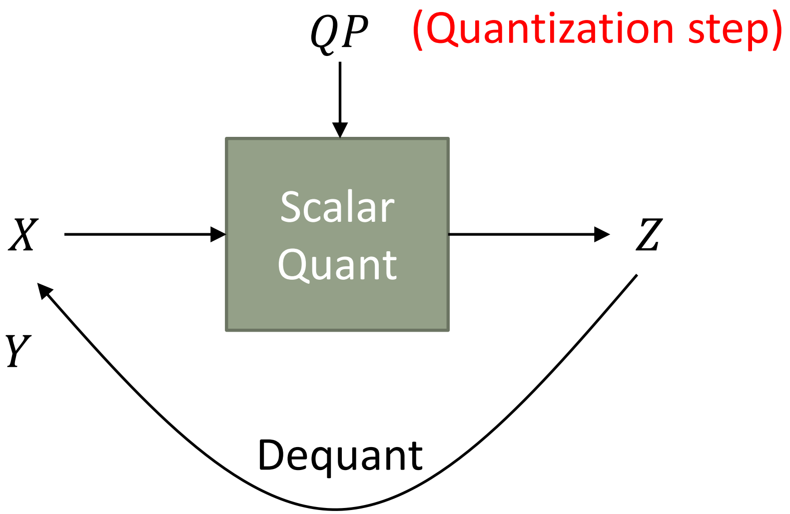
$$\begin{bmatrix} -415 & -33 & -58 & 35 & 58 & -51 & -15 & -12 \\ 5 & -34 & 49 & 18 & 27 & 1 & -5 & 3 \\ -46 & 14 & 80 & -35 & -50 & 19 & 7 & -18 \\ -53 & 21 & 34 & -20 & 2 & 34 & 36 & 12 \\ 9 & -2 & 9 & -5 & -32 & -15 & 45 & 37 \\ -8 & 15 & -16 & 7 & -8 & 11 & 4 & 7 \\ 19 & -28 & -2 & -26 & -2 & 7 & -44 & -21 \\ 18 & 25 & -12 & -44 & 35 & 48 & -37 & -3 \end{bmatrix} \;./\; \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

quantization matrix

$$\text{round}\left(-\frac{415}{16}\right) = -26$$

Scalar Quantization

Quantization is a process that maps a signal within a certain range to a quantized signal within a reduced range.



X: original signal

Z: quantized signal

Y: dequantized signal

A uniform quantizer is $Z = \mathrm{round}(\frac{X}{QP})$

$$Y = Z \times QP$$

Quantization error: $|Y - X|$

Scalar Quantization

$$Z = \mathrm{round}(\frac{X}{QP}) \qquad Y = Z \times QP$$

	Y			
X	QP=1	QP=2	QP=3	QP=5
-4	-4	-4	-3	-5
-3	-3	-2	-3	-5
-2	-2	-2	-3	0
-1	-1	0	0	0
0	0	0	0	0
1	1	0	0	0

Overview of Quantization in H.264

- Quantization in H.264 involves reducing the precision of the transform coefficients to achieve **high compression ratios**.
- It involves a set of predefined **quantization matrices** that dictate how much each coefficient is scaled.

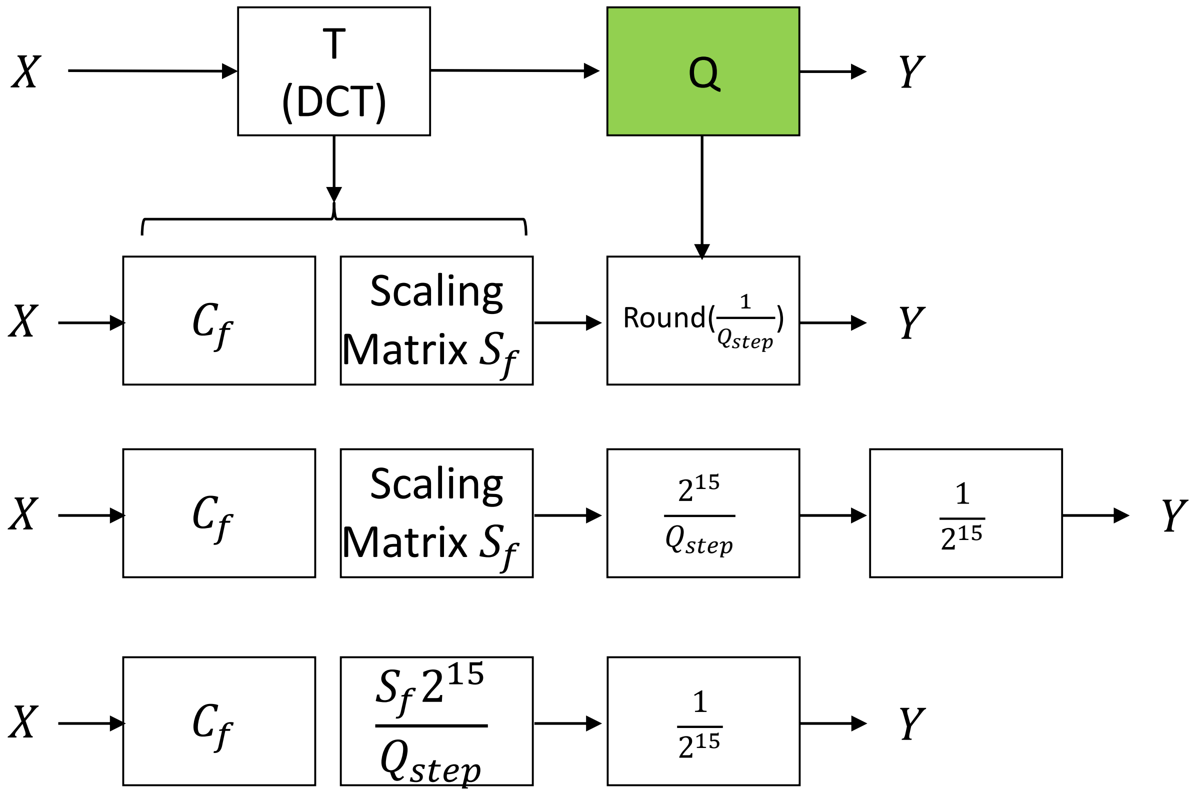
Quantization $c_q = \mathrm{round}(\frac{c}{q})$

c: transformed coefficient
q: a quantization step size

Dequantization $c_{dq} = c_q \times q$

Quantization Process (H.264)

To minimize the computation, the transform and quantization processes are combined and simplified.



Qstep(QP) = Qstep(QP%6) · 2floor(QP/6)

Quantization Process (H.264)

Applying 2D DCT to a 4x4 block X

$$a = \frac{1}{2}$$

$$Y = AXA^T \qquad b = \sqrt{\frac{1}{2}}\cos\frac{\pi}{8}$$

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix} \qquad c = \sqrt{\frac{1}{2}}\cos\frac{3\pi}{8}$$

where it’s rows are orthonormal.

Multiply by 2.5 and round to the nearest integer

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & 1 \end{bmatrix}$$

For minimizing the complexity of implementing the transform (requiring only additions and binary shifts)

Reference: https://www.vcodex.com/h264avc-4x4-transform-and-quantization/

Qstep(QP) = Qstep(QP%6) · 2floor(QP/6)

Quantization Process (H.264)

Applying 2D DCT to a 4x4 block X

$$Y = AXA^T$$

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix} \qquad a = \frac{1}{2} \qquad b = \sqrt{\frac{1}{2}}\cos\frac{\pi}{8} \qquad c = \sqrt{\frac{1}{2}}\cos\frac{3\pi}{8}$$

where it's rows are orthonormal.

$$A \rightarrow C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & 1 \end{bmatrix}$$

For row orthonormality, multiply c_{ij} by $\frac{1}{\sqrt{\sum_j c_{ij}^2}}$

$$R_f = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \end{bmatrix}$$

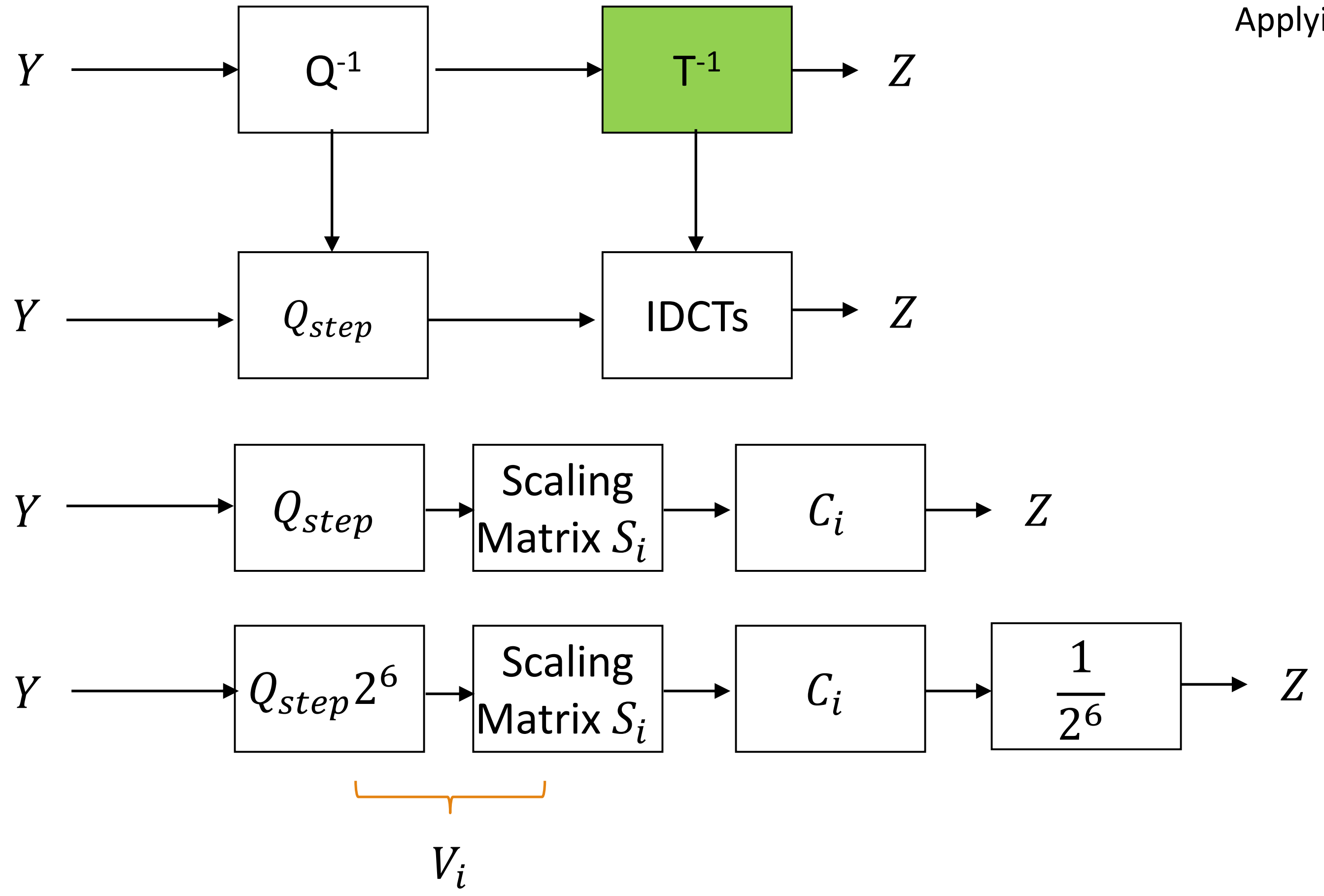
$$A_1 = C_f \cdot R_f \ \ (\cdot : \mathrm{element-wise\ multiplication})$$

$$Y = A_1 X A_1^T = [C_f \cdot R_f]X[C_f \cdot R_f]^T = C_f X C_f^T \cdot [R_f \cdot R_f^T]$$

$$Y = C_f X C_f^T \cdot S_f$$

$$S_f = R_f \cdot R_f^T = \begin{bmatrix} 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \\ 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \end{bmatrix}$$

Inverse Quantization



Applying 2D IDCT to a 4x4 block Y

$$Z = A^T Y A$$

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix} \qquad a = \frac{1}{2} \qquad b = \sqrt{\frac{1}{2}}\cos\frac{\pi}{8} \qquad c = \sqrt{\frac{1}{2}}\cos\frac{3\pi}{8}$$

rounding

$$C_i = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

For row orthonormality

$$R_i = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \end{bmatrix} \rightarrow A_2 = C_i \cdot R_i$$

$$Z = A_2^T Y A_2 = [C_i \cdot R_i]^T Y [C_i \cdot R_i] = C_i^T [Y \cdot R_i^T \cdot R_i]\, C_i$$

$$S_i = R_i^T \cdot R_i$$

Reference: https://www.vcodex.com/h264avc-4x4-transform-and-quantization/

Final Derivation

Q $\quad X \rightarrow \boxed{C_f} \rightarrow \boxed{\frac{S_f 2^{15}}{Q_{step}}} \rightarrow \boxed{\frac{1}{2^{15}}} \rightarrow Y$

IQ $\quad Y \rightarrow \underbrace{\boxed{Q_{step} 2^6} \rightarrow \boxed{S_i}}_{V_i} \rightarrow \boxed{C_i} \rightarrow \boxed{\frac{1}{2^6}} \rightarrow Z$

$$V_i = \begin{bmatrix} 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \\ 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \end{bmatrix} \rightarrow \text{QP } 0$$

QP	Qstep
0	0.625
1	0.702
2	0.787
3	0.884
4	0.992
5	1.114

Any value of Qstep can be derived from the first 6 values in the table (QP0 – QP5) as follows:

Qstep(QP) = Qstep(QP%6) · 2floor(QP/6)

Reordering

- Reorder 2-D signals to 1-D signals for run-level encoding
  - making nonzero coefficients and zero coefficients clustered
  - A large number of zero values could be encoded more compactly

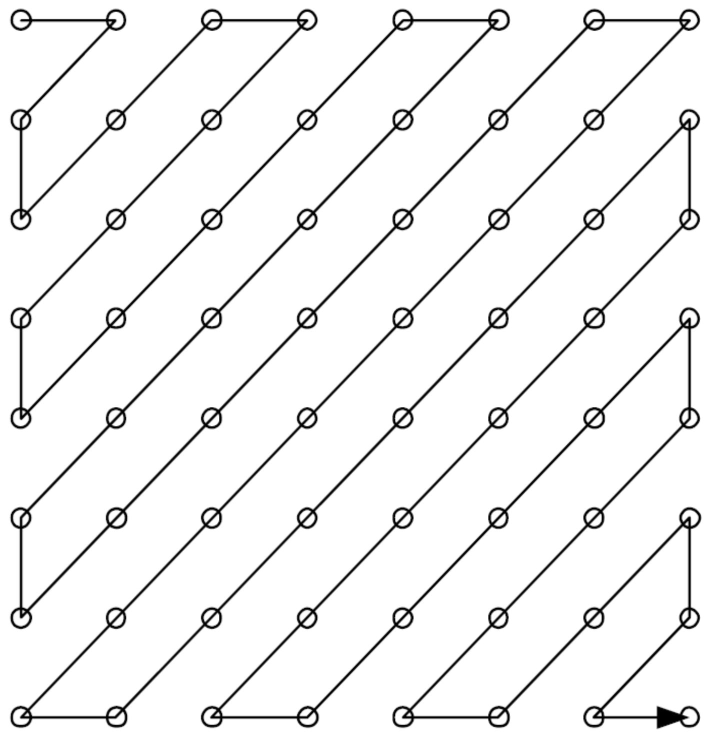
<u>run-level encoding</u>

Input array: 32,0,0,−2,-5,6,1,0,0,0,0,−3, . . .

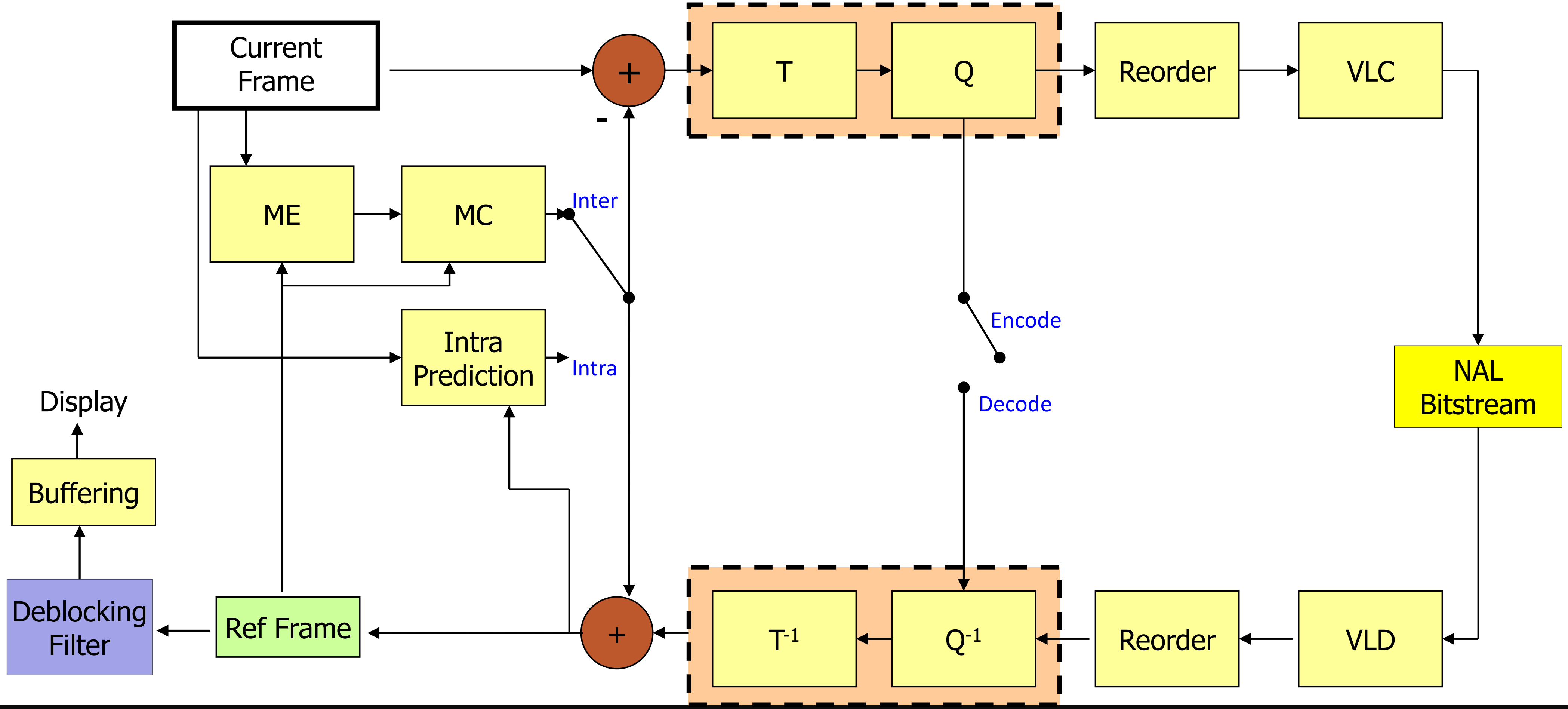
Output array: (0, 32),(2, -2),(0,-5),(0,6),(0,1),(4,−3), . . .

$$(x, y) = \begin{cases} x\text{: number of zeros prior to y} \\ y\text{: signal value} \end{cases}$$

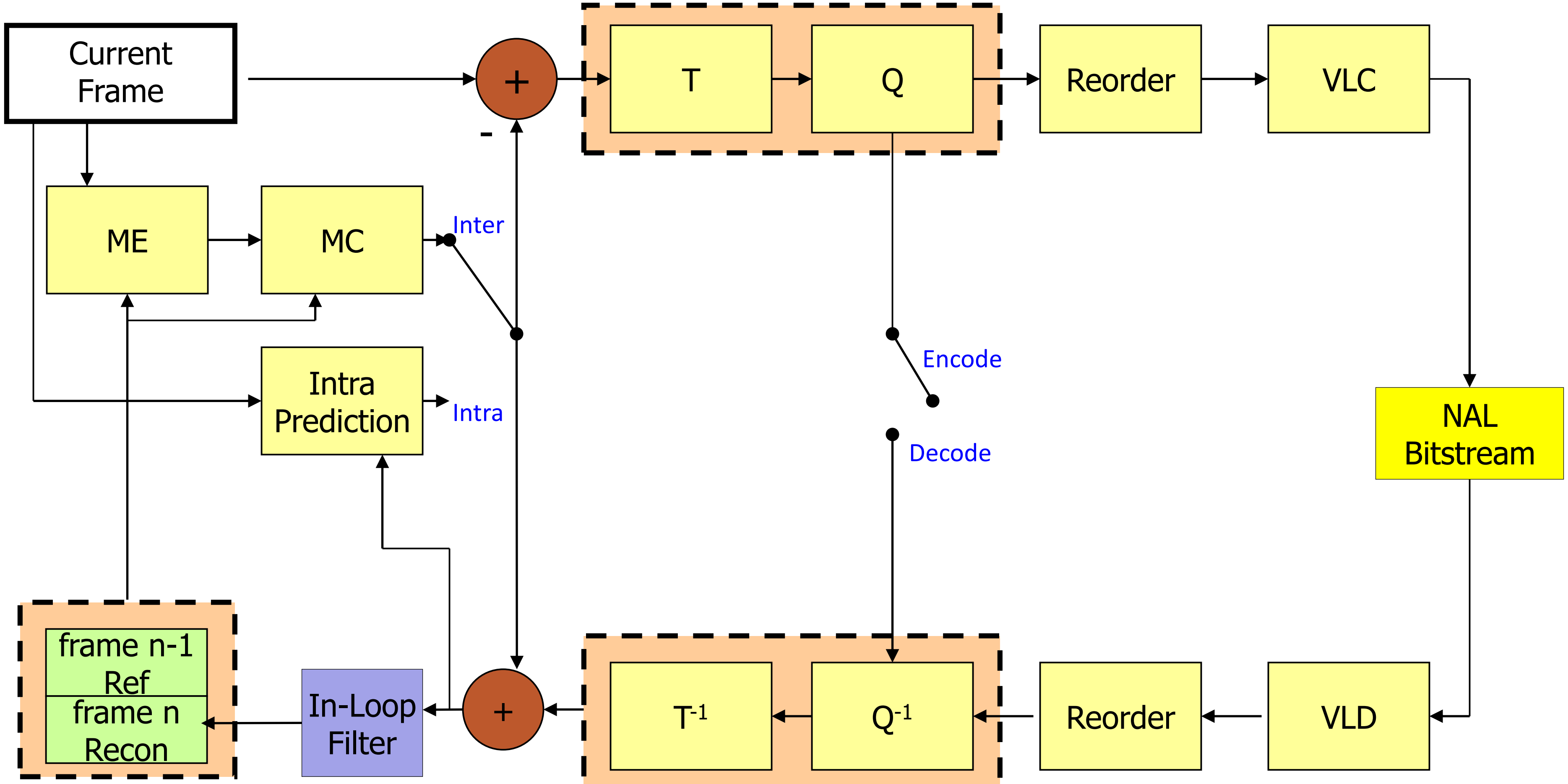
- Zigzag scan order



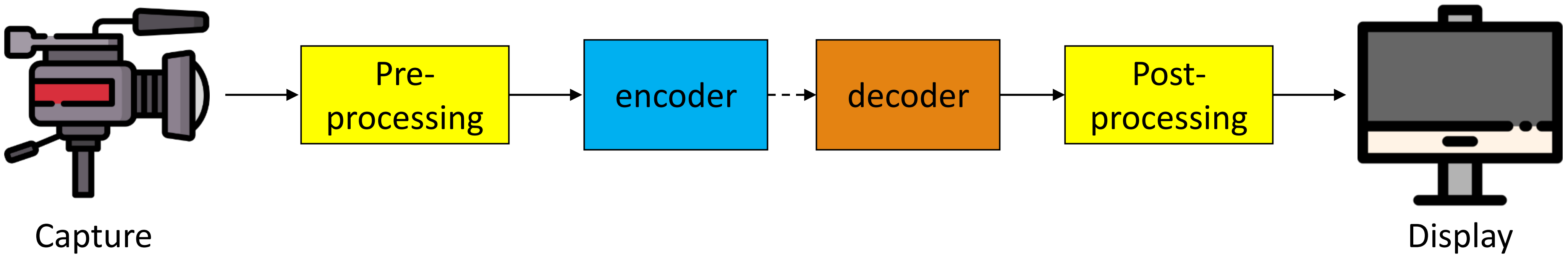
Deblocking Filtering (MPEG4)



Deblocking Filtering (In-loop Filter)



Pre-Filtering



- DCT-based Transform coding works well for smooth and noise-free video
- However, noise or unwanted camera movements can be detrimental to the video quality
- Pre-filtering the captured video frames before encoding can be helpful in terms of compression efficiency

Smooth Filtering

- Due to block-based compression for videos, they may suffer from blocking artifacts using a large QP, affecting their visual quality
- To remove blocking artifacts, one can apply smoothing filter to decoded frames
- Input image convolved with a smooth kernel

  http://setosa.io/ev/image-kernels

$$\begin{array}{|c|c|c|c|}\hline 2 & 5 & 1 & 5 \\ \hline 7 & 1 & 3 & 2 \\ \hline 1 & 9 & 9 & 2 \\ \hline 6 & 7 & 1 & 2 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array}$$

Input image

smooth kernel

Examples of blocking artifacts

Gaussian Filter

Gaussian Filtering

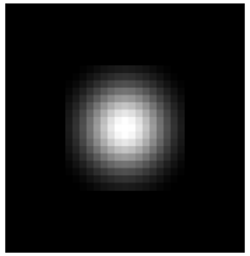
$$I_{GF}(x) = \frac{1}{W_{GF}} \sum_{y \in \Omega(x)} G_\sigma(\|y - x\|) I(y)$$

$$\sigma \uparrow \ \rightarrow \quad more\ blurry$$

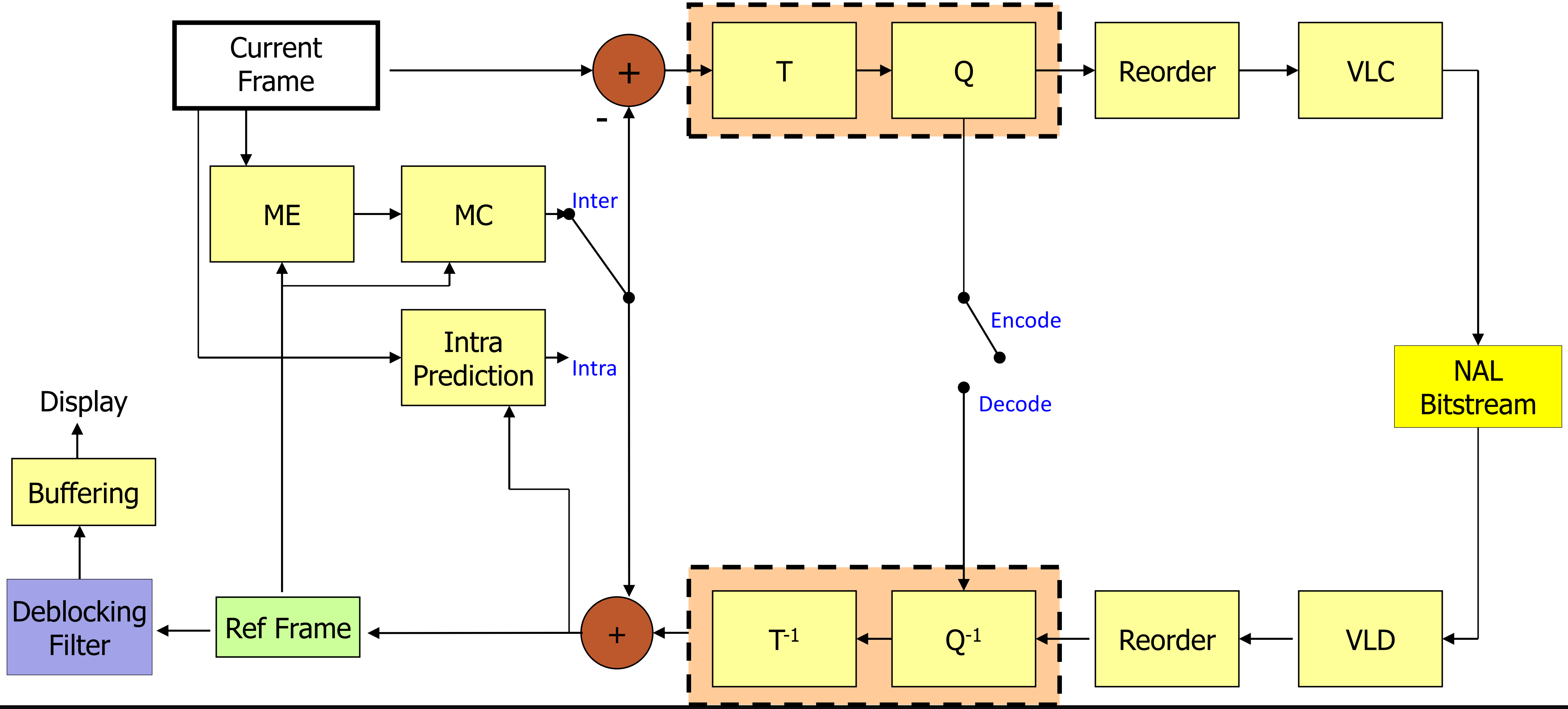
$$W_{GF} = \sum_{y \in \Omega(x)} G_\sigma(\|y - x\|)$$

$$G_\sigma(k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{k^2}{-2\sigma^2}}$$

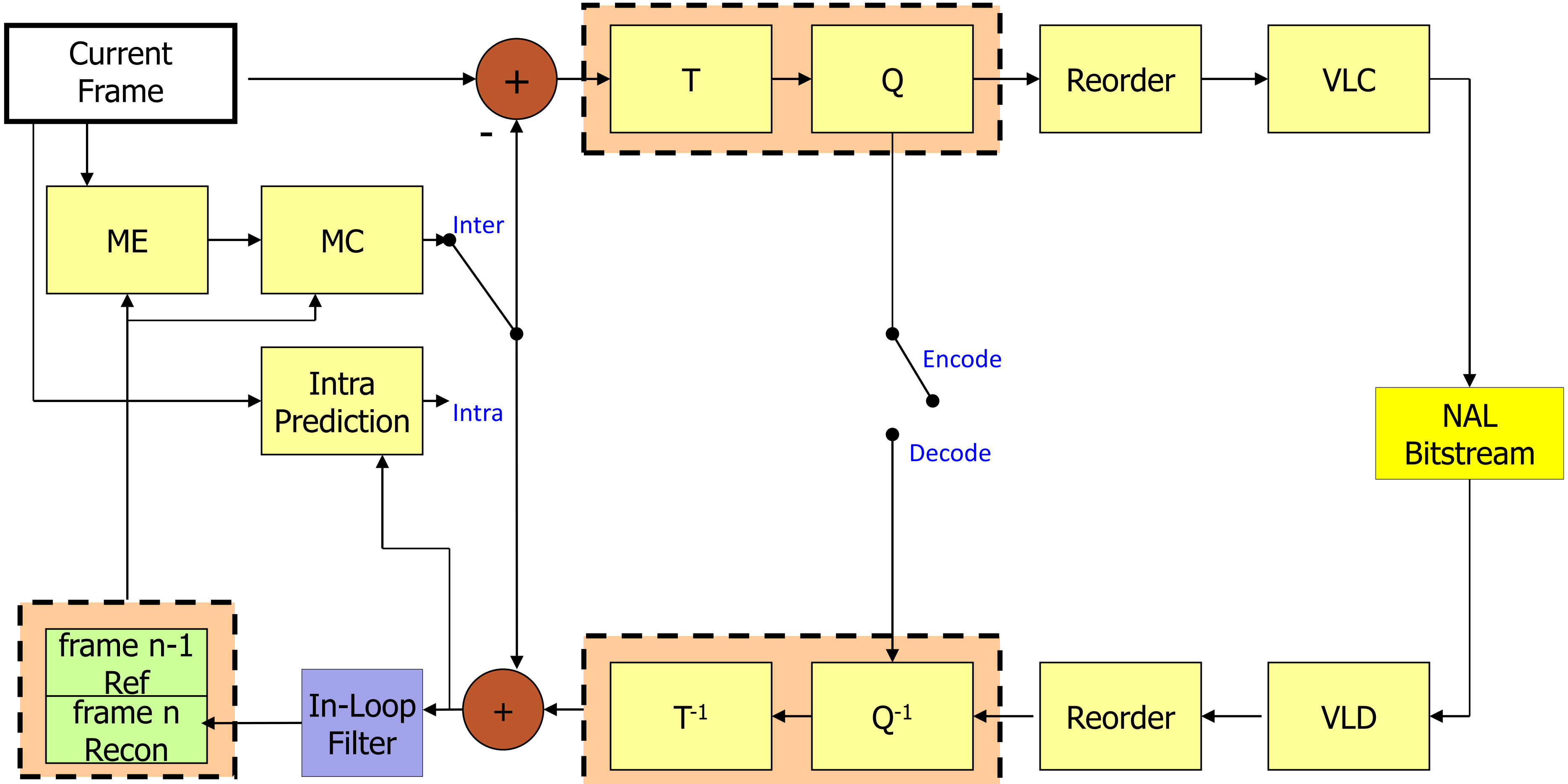
Normalized Gaussian



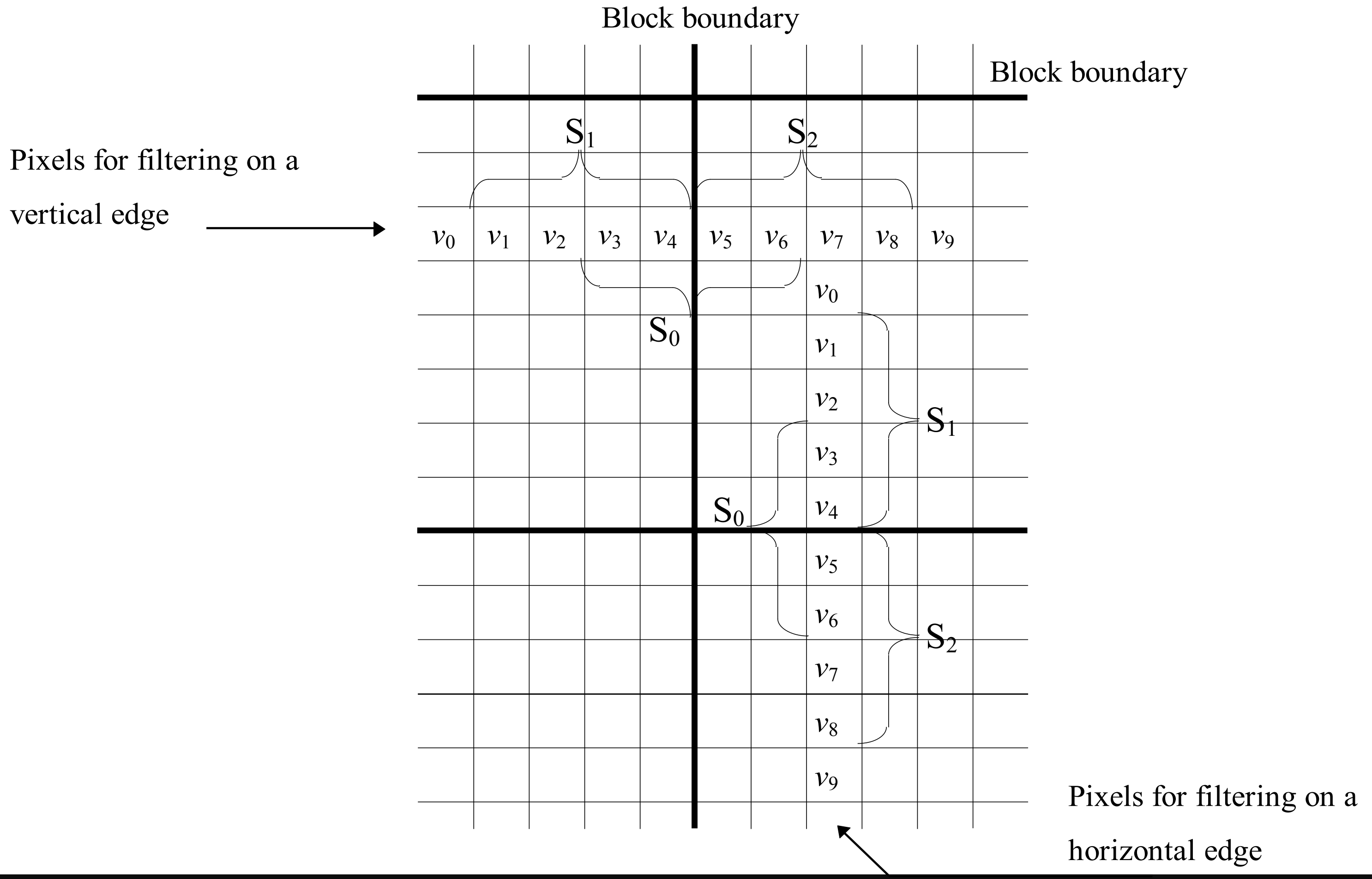
Deblocking Filtering (MPEG4)



Deblocking Filtering (In-loop Filter)



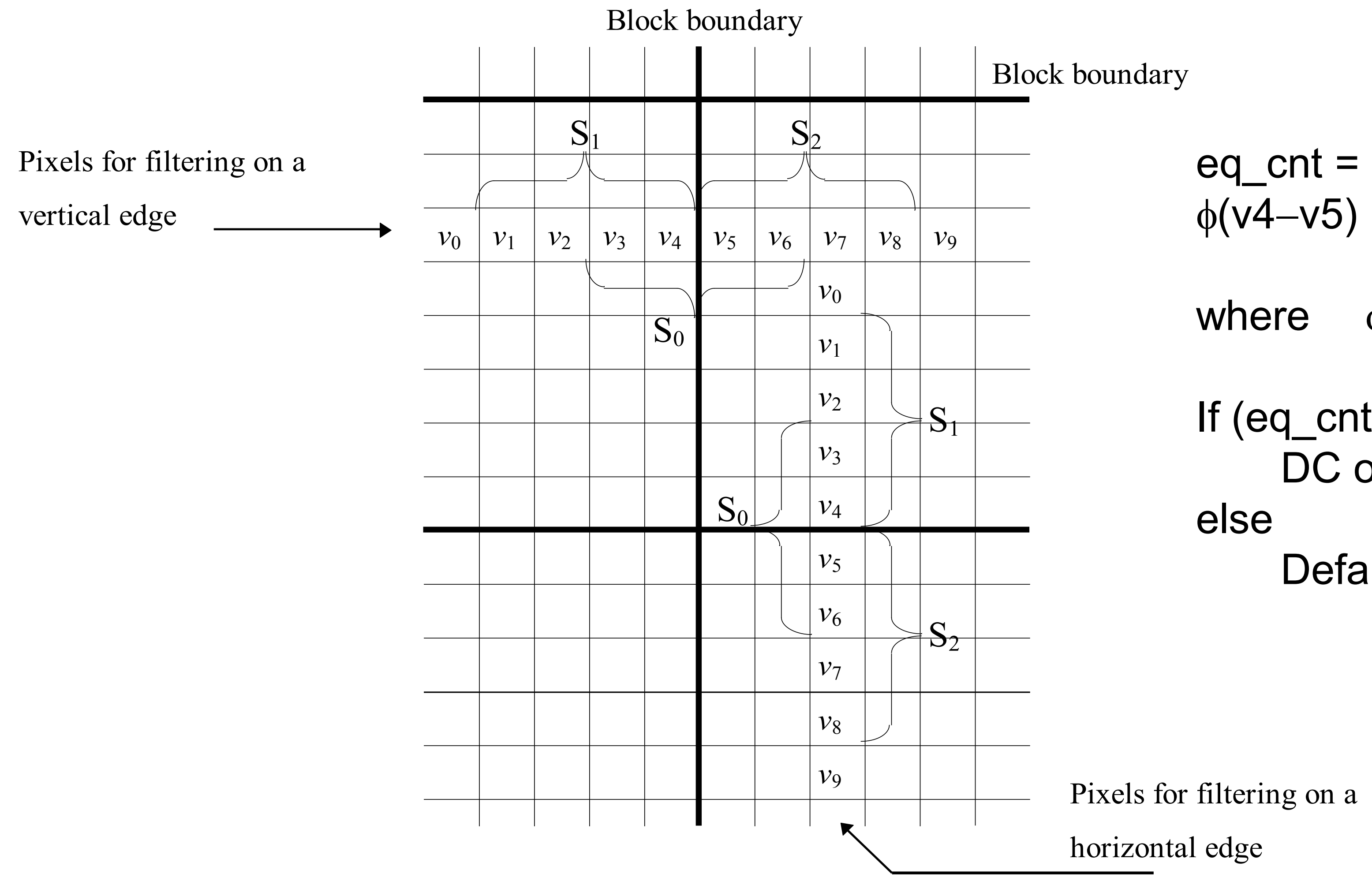
Deblocking Filtering (MPEG 4)



It operates along the 8x8 block edges at the decoder as a post-processing step

Ref: Text of ISO/IEC 14496-2:2001 (Unifying N2502, N3307, N3056, and N3664) - w4350_14496-2_2nd

DC offset mode vs Default mode



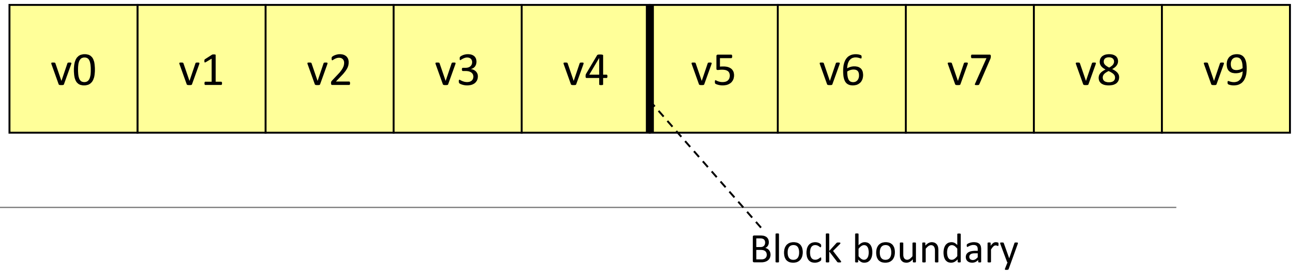
eq_cnt = $\phi(v0-v1) + \phi(v1-v2) + \phi(v2-v3) + \phi(v3-v4) + \phi(v4-v5) + \phi(v5-v6) + \phi(v6-v7) + \phi(v7-v8) + \phi(v8-v9)$,

where $\phi(\gamma) = 1$ if $|\gamma| \leq$ THR1 and 0 otherwise.

If (eq_cnt $\geq$ THR2)
  DC offset mode is applied, (very smooth region)
else
  Default mode is applied.

Generally, THR1 = 2 and THR2 = 6

Default Mode



- Only replacing v_4 and v_5 as

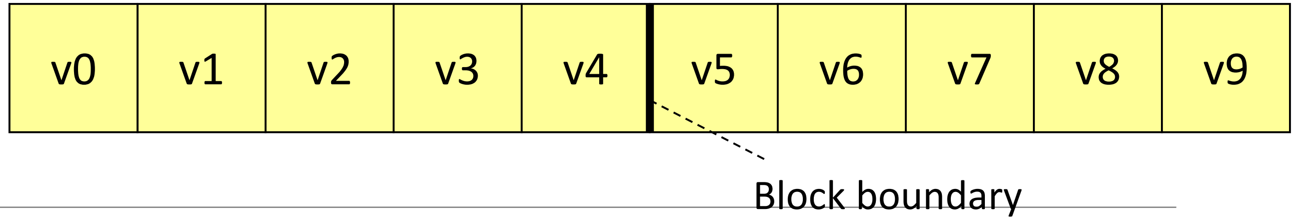
$$v_4' = v_4 - d,$$
$$v_5' = v_5 + d,$$

$$d = \mathrm{CLIP}(5 \cdot (a_{3,0}' - a_{3,0}) // 8,\ 0,\ (v_4 - v_5)/2) \cdot \delta(|a_{3,0}| < QP),$$
$$a_{3,0}' = \mathrm{SIGN}(a_{3,0}) \cdot \mathrm{MIN}(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|).$$

- Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ are calculated by an inner product of the approximated DCT kernel [2 -5 5 -2] as

a3,0 = ([2 -5 5 -2] • [v3 v4 v5 v6]^T) // 8,
a3,1 = ([2 -5 5 -2] • [v1 v2 v3 v4]^T) // 8,
a3,2 = ([2 -5 5 -2] • [v5 v6 v7 v8]^T) // 8.

DC offset mode



- For very smooth regions, the default mode is not strong enough to smooth the blocking artifiact due to DC offset

max = MAX (v1, v2, v3, v4, v5, v6, v7, v8),
min = MIN (v1, v2, v3, v4, v5, v6, v7, v8),
if (|max–min| < 2·QP) {

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$$

Replacing v_i with v'_i

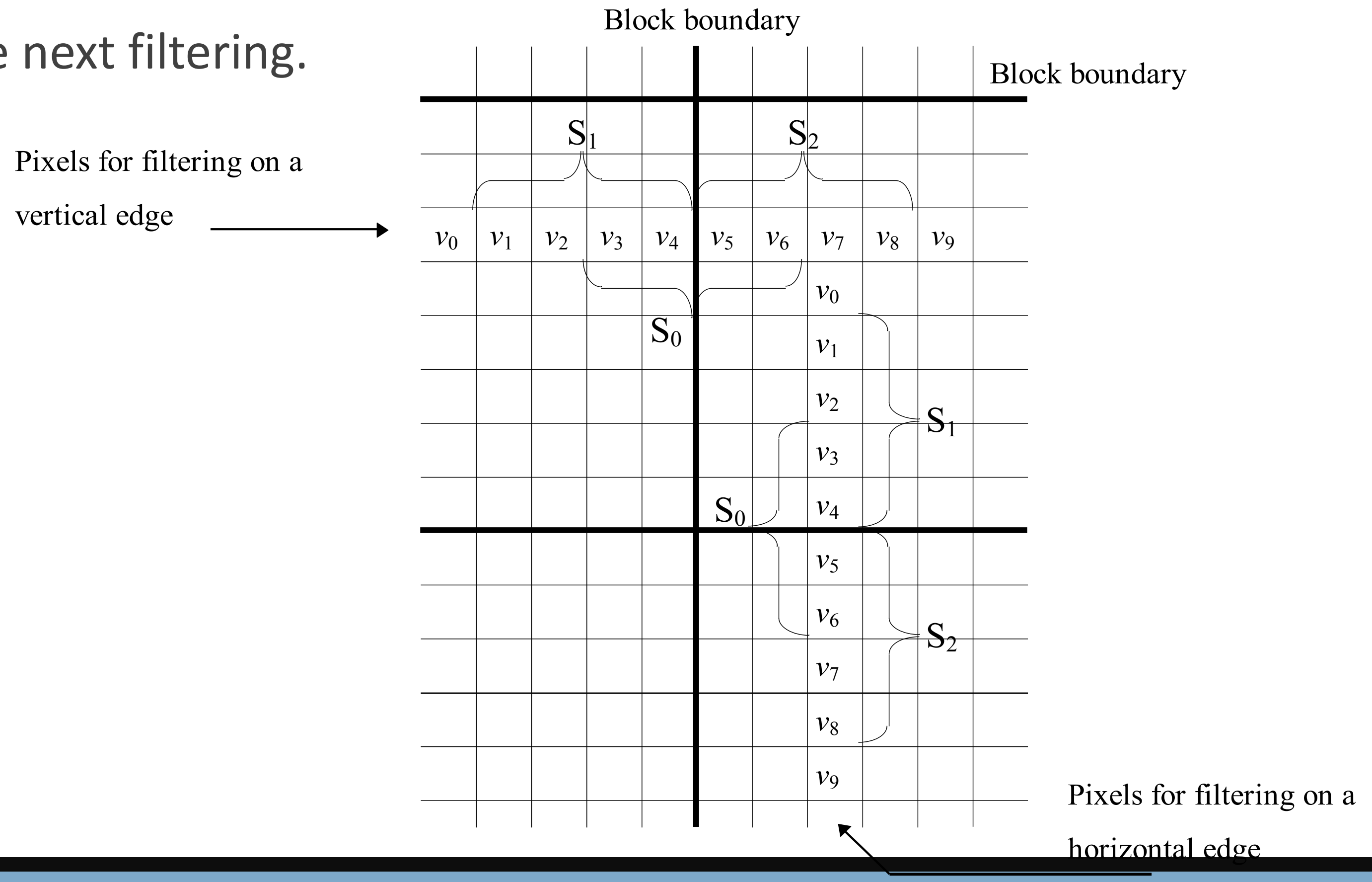
$$p_m = \begin{cases} (|v_1 - v_0| < QP) ? v_0 : v_1, & if \quad m < 1 \\ v_m, & if\, 1 \le m \le 8 \\ (|v_8 - v_9| < QP) ? v_9 : v_8, & if \quad m > 8 \end{cases}$$

$$\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$$

} else
No change will be done.

Deblocking Filtering

- The filtering is done first along the horizontal edges and then for the vertical edges.
- Filtered pixels are used for the next filtering.

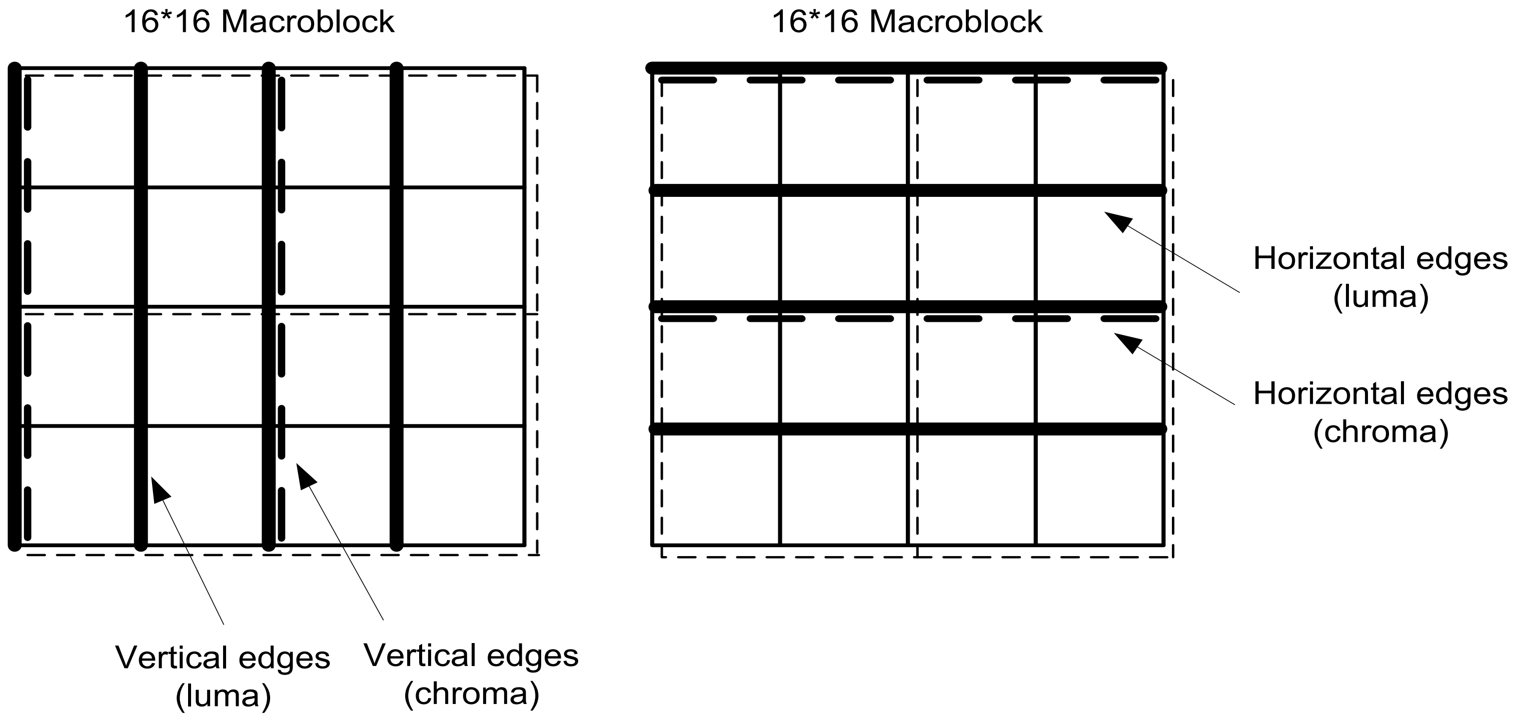


Purpose & Advantage of In-loop Filter

- Reduce blocking distortion
- Improve the appearance of decoded frames subjectively and objectively
- Improve compression performance
  - More faithful reproduction of the original frame than a blocky, unfiltered image.

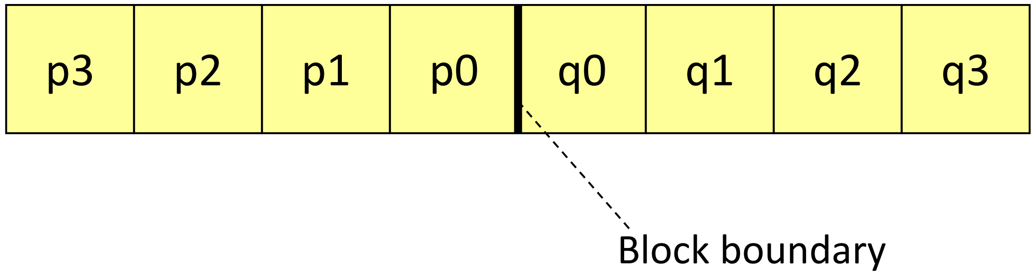
In-Loop Filter (ILF)

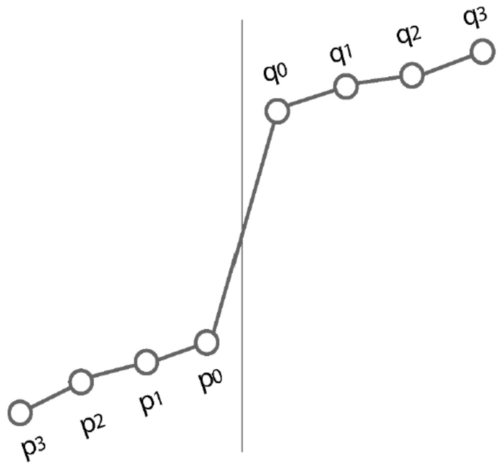
- Using H.264 as an example
- Performed on MB basis
- Filtering is applied to 4x4 block edges of a frame



In-Loop Filter

- Filtering is applied to both luma and chroma components
- Vertical edges are filtered first, and then horizontal edges
- Horizontal filtering first, and then vertical filtering
- Filter samples across a 4x4 block horizontal or vertical boundary

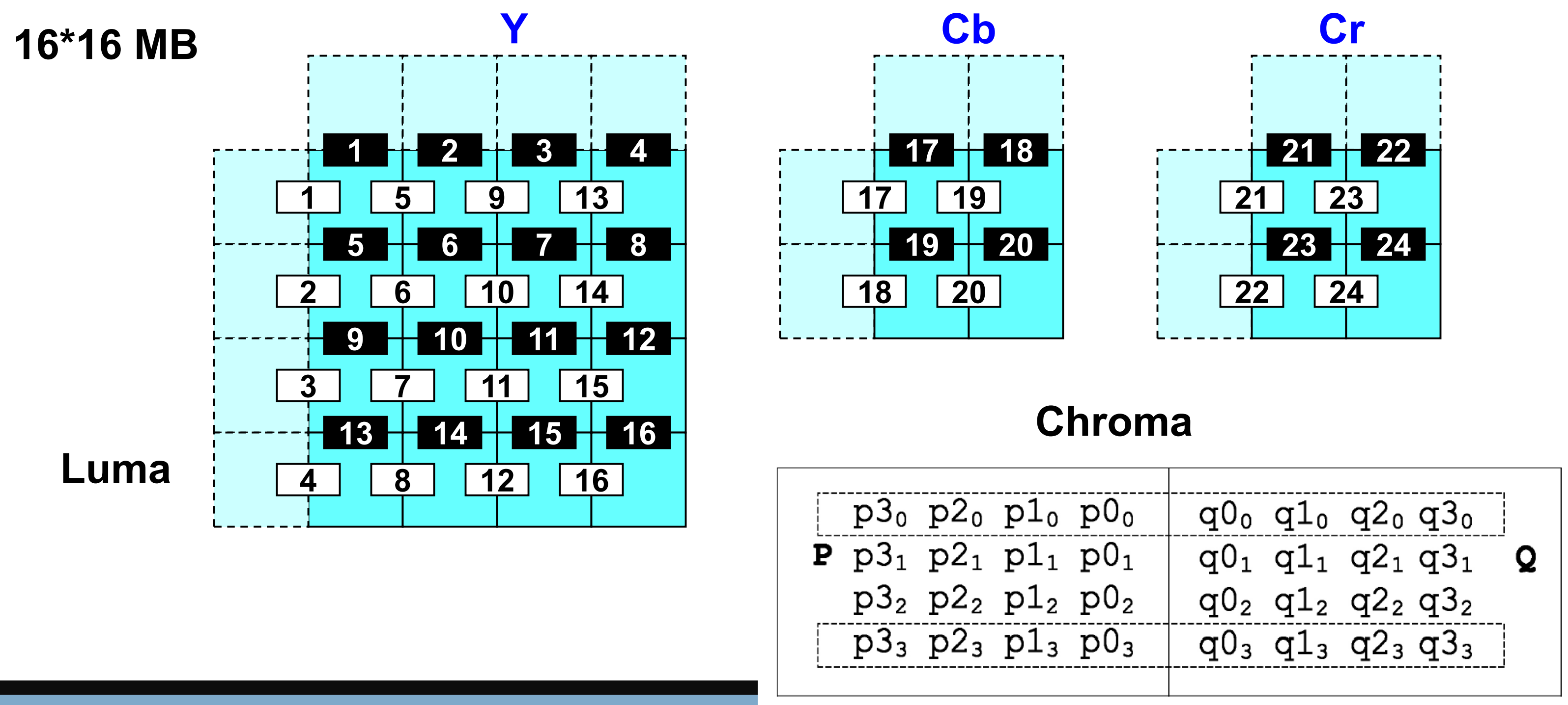




Ex: a block boundary with a blocking artifact

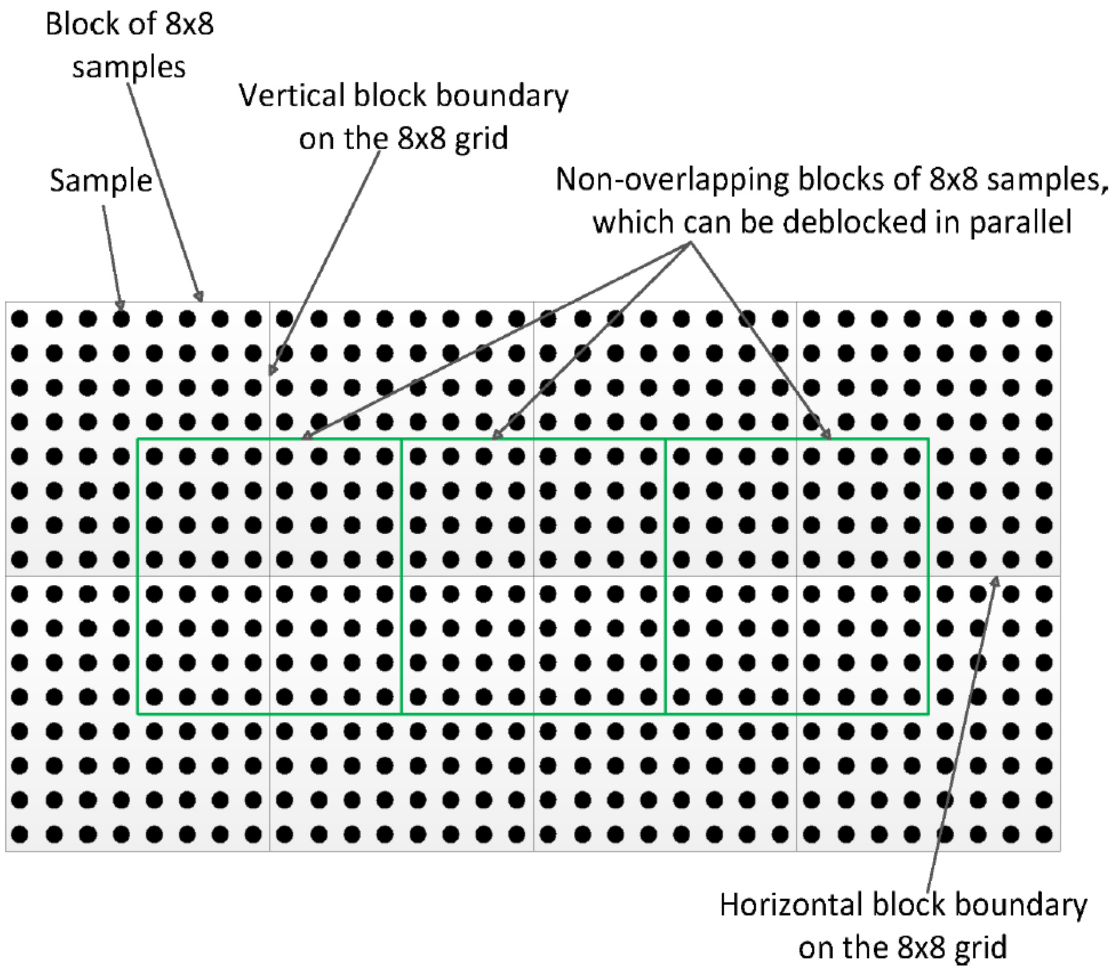
Pixel Processing Order of ILF

- The white label with black number denotes the horizontal filtering on vertical edges
- The black label denotes the vertical filtering on horizontal edges



ILF for 8x8 Block Grid

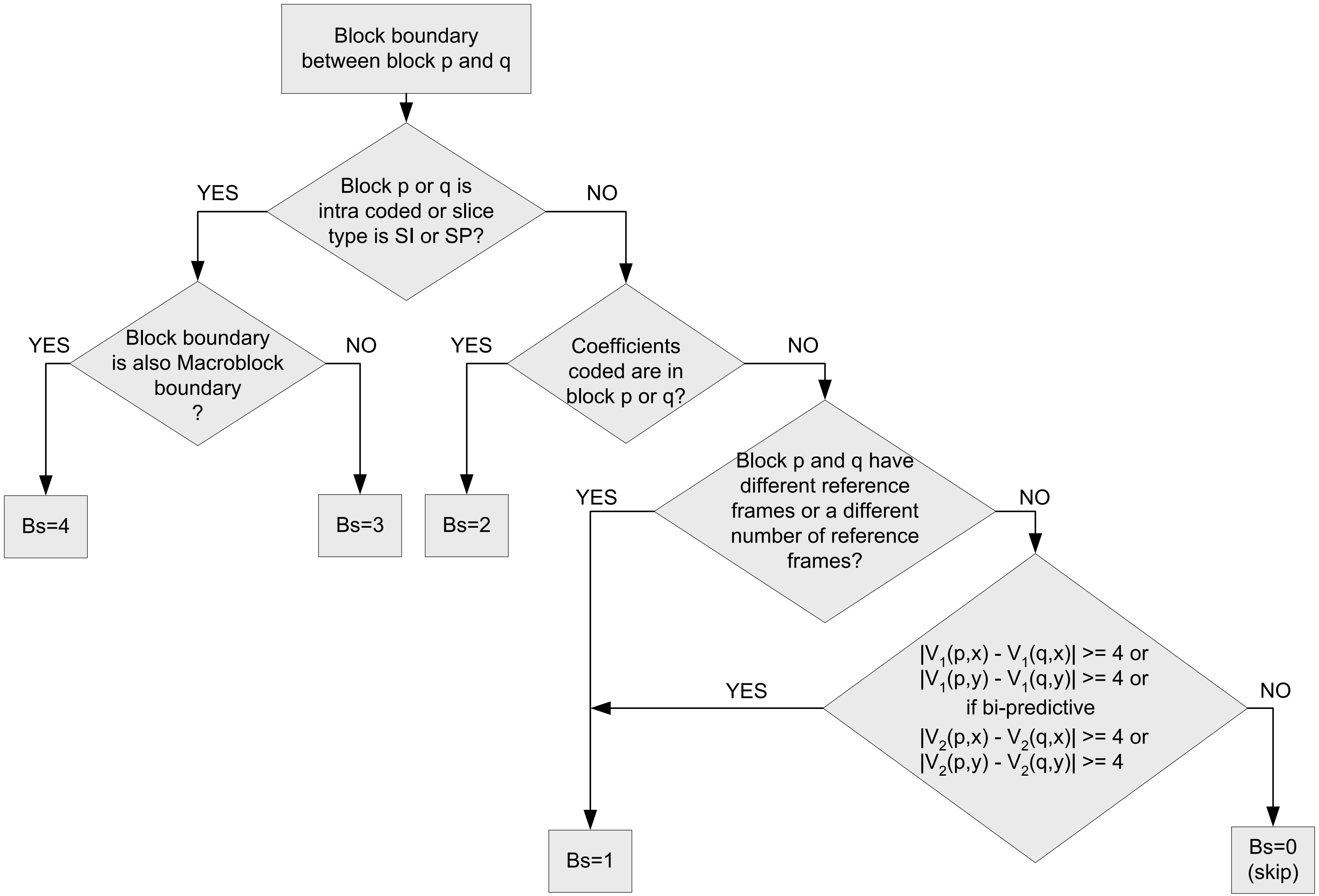
- Horizontal and vertical block boundaries on the 8 × 8 grid
- Nonoverlapping blocks of the 8 × 8 samples can be filtered in parallel.



Boundary Filtering Strength

- The ‘strength’ of the filter depends on
  - Quantization parameter
  - Coding modes of neighboring blocks
  - Motion vectors
  - Code block pattern
  - Reference pictures

Boundary Filtering Strength Determination



Threshold for Block Boundary

- Filter Decision
  - BS > 0 and
  - |p0-q0|<α && |p1-p0|<β && |q1-q0|<β (not a real edge)

p3	p2	p1	p0	q0	q1	q2	q3

- α and ß are thresholds defined to ‘switch off’ the filter determined by QP (0~51)

$$qPav = (qP_p + qP_q + 1) >> 1$$

Derivation of indexA and indexB from offset dependent threshold variables α and β

	indexA (for α) or indexB (for β)																									
qPav	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

	indexA (for α) or indexB (for β)																									
qPav	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

Filtering Process-Weaker Filtering

- BS less than 4
  - Luma
    - $C = C0+ap+aq$
    - $ap = ABS(p2-p0)$, $aq = ABS(q2-q0)$
    - $\triangle = Clip(-C, C, (((q0-p0)<<2+(p1-q1)+4)>>3))$
    - $P0 = Clip(0, 255, p0+\triangle)$
    - $Q0 = Clip(0, 255, q0+\triangle)$
    - $P1 = p1+Clip(-C0, C0, (p2+((p0+q0+1)>>1)-(p1<<1))>>1)$
    - $Q1 = q1+Clip(-C0, C0, (q2+((p0+q0+1)>>1)-(q1<<1))>>1)$
  - Chroma
    - $C = C0+1$
    - $P0 = Clip(0, 255, p0+\triangle)$
    - $Q0 = Clip(0, 255, q0+\triangle)$

p3	p2	p1	p0	q0	q1	q2	q3

Filtering Process-Strongest Filtering

- Luma
  - $Q0 = (p1+2*p0+2*q0+2*q1+q2+4)>>3$
  - $Q1 = (p0+q0+q1+q2+2)>>2$
  - $Q2 = (2*q3+3*q2+q1+q0+p0+4)>>3$
  - $P0 = (p2+2*p1+2*p0+2*q0+q1+4)>>3$
  - $P1 = (p2+p1+p0+q0+2)>>2$
  - $P2 = (2*p3+3*p2+p1+p0+q0+4)>>3$
- Chroma
  - $P0 = (2*p1+p0+q1+2)>>2$
  - $Q0 = (2*q1+q0+p1+2)>>2$

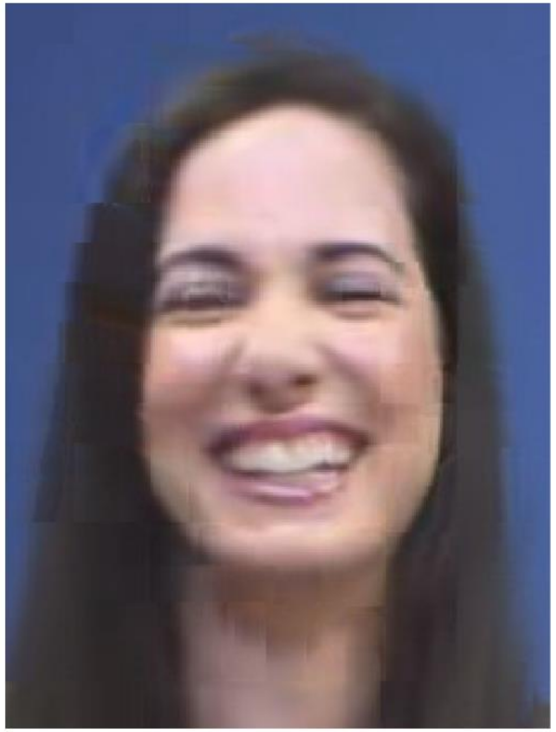
p3	p2	p1	p0	q0	q1	q2	q3

Deblocking Results

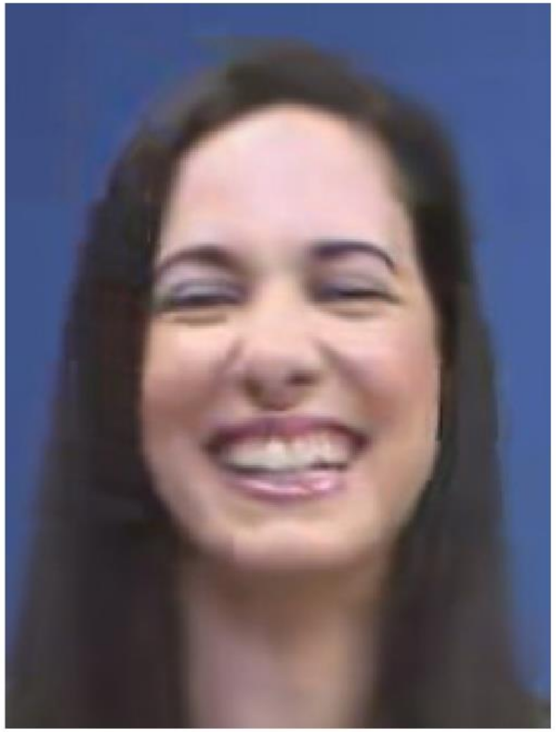
Before

After

Deblocking Results



Before



After