The MPEG4/AVC standard: description and basic tasks splitting

Isabelle Hurbain[1]
Centre de recherche en informatique
École des Mines de Paris
hurbain@cri.ensmp.fr

January 7, 2004

[1]35, rue Saint-Honoré, 77305 Fontainbleau cedex, France.

Contents

Abstract

This document presents the MPEG4/AVC (or H.264 or H.26L) standard and splits the whole algorithme into basic tasks. These basic tasks will then be implemented in a library to provide an optimized version of these tasks for different architectures, the same way BLAS does with linear algebra. We will first describe in detail what is MPEG4 encoding. This description is mainly based on [4], [5], [1], and the reference implementation of the MPEG4/AVC encoder, JM.

Chapter 1

The MPEG4/AVC standard

1.1 Structure of video

1.1.1 Structure of frames

A MPEG4-AVC video sequence is a sequence of coded pictures. A coded picture can be an entire frame, or a single field. Generally, a frame is composed of two interleaved fields - a top field, that contains even-numbered rows, and a bottom field, that contains odd-numbered rows of the picture.

1.1.2 Color space and sampling

The human vision does not works in terms of Red Green Blue as it is often used by image systems. Instead, it works in terms of luminosity and color. That is why MPEG encodings (in general) use a $YCbCr$ color space. Y is called luma, and represents brightness. C_b and C_r are the chroma components and represent the deviation of the color from grey on respectively the blue-yellow axis and the red-green axis.

The human vision system is more sensible to luma than to chroma. Consequently, MPEG4/AVC encodes the components in a way that a chroma component has four times less samples than the luma component (one of two samples in horizontal and vertical direction). This sampling is called 4:2:0 sampling.

1.1.3 Macroblocks

A picture is partitioned into fixed-size macroblocks that contains 16×16 luma samples and 8×8 chroma components. The macroblocks are used by most of the coding algorithm.

1.1.4 Slices and macroblock ordering

Slices are a sequence of macroblocks. They are self-contained in the sense that they do not need information from other slices to be decoded. Information from other slices may however be needed for the deblocking filter (see 1.4).

There are two ways to order macroblocks:

- raster scan
- flexible macroblock ordering (FMO)

Raster scan orders blocks from left to right and from top to bottom. An example of raster scan can be seen on figure 1.1.

H.264/AVC defines how pictures can be partitioned using FMO. FMO defines a *macroblock to slice group map*, that assigns a group to each macroblock of the picture. This way, a picture can be split into many patterns, such as interleaved slices, one or more “foreground” slice and a “leftover” slice group, or checkerboard mapping. This may be especially useful for networking applications, or for quality compression issues (we can imagine that “foreground” slices are less compressed than “leftover” and this way have a better quality).

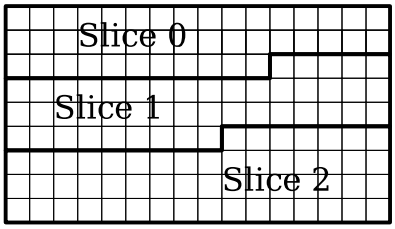

Figure 1.1: Raster scan example

1.1.5 Slice types

Each slice can be coded using different coding types:

- I slices contain only intra-predicted macroblocks (see 1.2)
- P slices contain intra-predicted macroblocks and inter-predicted macroblocks with at most one motion-compensated prediction signal per prediction block
- B slices contain the same encodings as above, and can also be predicted using two motion-compensated prediction signals per prediction block
- SP slices are designed to support switching between similar videos (for example a single video encoded at two different bitrates). The obvious solution would be to add an I slice at the switching point; SP slices allow a similar mechanism without the intervention of a place-consuming I slice.
- SI slices are used in a similar way, but use intra prediction. They can be used for example to switch from one sequence to a completely different sequence (in this case, SP frames will be highly uneffective as there is no motion prediction).

1.2 Spatial prediction

1.2.1 Luma prediction

There are sets of prediction modes for spatial luma prediction (intra coding): 4×4 mode and 16×16 mode.

1.2.1.1 4×4 mode

There are 9 prediction modes for intra 4×4 mode. They all use the same notations for prediction. These notations are defined on table 1.1.

Q	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				
M								
N								
O								
P								

Table 1.1: Pixel Notation

The nine prediction modes are defined as presented in table 1.2 and figure 1.2.

Note: The following expressions are all expressed by divisions, for sake of undestanding. These divisions are integer divisions, and are normally implemented via bit shifting ($/2^n \Leftrightarrow >> n$)).

0	Vertical
1	Horizontal
2	DC
3	Diagonal Down Left
4	Diagonal Down Right
5	Vertical Right
6	Horizontal Down
7	Vertical Left
8	Horizontal Up

Table 1.2: 4×4 prediction modes

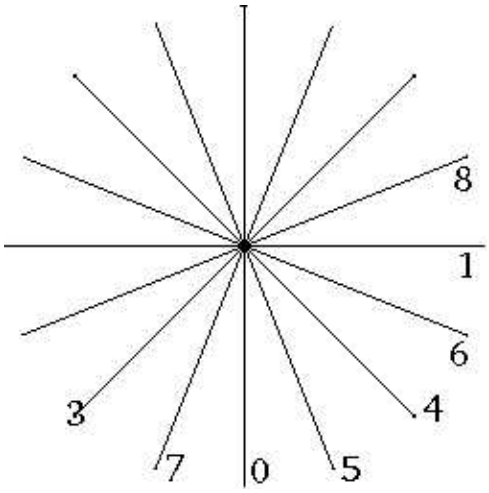

Figure 1.2: Prediction mode directions

Mode 0: Vertical prediction mode This mode is only available if A, B, C and D are available. Then we have:

$$\begin{array}{ccccccccc} a & = & e & = & i & = & m & = & A \\ b & = & f & = & j & = & n & = & B \\ c & = & g & = & k & = & o & = & C \\ d & = & h & = & l & = & p & = & D \end{array}$$

Mode 1: Horizontal prediction mode This mode is only available if I, J, K and L are available. Then we have:

$$\begin{array}{ccccccccc} a & = & b & = & c & = & d & = & I \\ e & = & f & = & g & = & h & = & J \\ i & = & j & = & k & = & l & = & K \\ m & = & n & = & o & = & p & = & L \end{array}$$

Mode 2: DC prediction If A, B, C, D, I, J, K and L are available, then all the value of the macroblock are set equal to P with

$$P = \frac{A + B + C + D + I + J + K + L + 4}{8}$$

(this division is implemented by left-shifting the bits of P by 3).

If only A, B, C and D exist, then

$$P = \frac{A + B + C + D + 2}{4}$$

Similarly, if only I, J, K, and L exist, then

$$P = \frac{I + J + K + L + 2}{4}$$

If A, B, C, D, I, J, K and L are not available, then $P = 128$.

Mode 3: Diagonal Down Left prediction This mode can be used if A, B, C, D, E, F, G and H are available. Then:

$$\begin{aligned} a &= \frac{A + 2B + C + 2}{4} \\ b = e &= \frac{B + 2C + D + 2}{4} \\ c = f = i &= \frac{C + 2D + E + 2}{4} \\ d = g = j = m &= \frac{D + 2E + F + 2}{4} \\ h = k = n &= \frac{E + 2F + G + 2}{4} \\ l = o &= \frac{F + 2G + H + 2}{4} \\ p &= \frac{G + 3H + 2}{4} \end{aligned}$$

Mode 4: Diagonal Down Right prediction This mode can be used if A, B, C, D, I, J, K, L and Q are available. Then:

$$\begin{aligned} a = f = k = p &= \frac{A + 2Q + I + 2}{4} \\ b = g = l &= \frac{Q + 2A + B + 2}{4} \\ c = h &= \frac{A + 2B + C + 2}{4} \\ d &= \frac{B + 2C + D + 2}{4} \\ e = j = o &= \frac{Q + 2I + J + 2}{4} \\ i = n &= \frac{I + 2J + K}{4} \\ m &= \frac{J + 2K + L}{4} \end{aligned}$$

Mode 5: Vertical Right prediction This mode can be used if A, B, C, D, I, J, K, L and Q are available. Then:

$$\begin{aligned}
a = j &= \frac{Q + A + 1}{2} \\
b = e = n &= \frac{I + 2Q + A + 2}{4} \\
c = l &= \frac{B + C + 1}{2} \\
d &= \frac{C + D + 1}{2} \\
f = o &= \frac{Q + 2A + B + 2}{4} \\
g = p &= \frac{A + 2B + C + 2}{4} \\
h &= \frac{B + 2C + D + 2}{4} \\
i &= \frac{J + 2I + Q + 2}{4} \\
k &= \frac{A + B + 1}{2} \\
m &= \frac{K + 2J + I + 2}{4}
\end{aligned}$$

Mode 6: Horizontal Down prediction This mode can be used if A, B, C, D, I, J, K, L and Q are available. Then:

$$\begin{aligned}
a = g &= \frac{Q + I + 1}{2} \\
b = h &= \frac{I + 2Q + A + 2}{4} \\
c &= \frac{B + 2A + Q + 2}{4} \\
d &= \frac{C + 2B + A + 2}{4} \\
e = k &= \frac{A + B + 1}{2} \\
f = l &= \frac{Q + 2A + B + 2}{4} \\
i = o &= \frac{J + K + 1}{2} \\
j = p &= \frac{I + 2J + K + 2}{4} \\
m &= \frac{K + L + 1}{2} \\
n &= \frac{J + 2K + L + 2}{4}
\end{aligned}$$

Mode 7: Vertical Left prediction This mode can be used if A, B, C, D, E, F, G and H are available. Then:

$$\begin{aligned} a &= \frac{A+B+1}{2} \\ b=i &= \frac{B+C+1}{2} \\ c=j &= \frac{C+D+1}{2} \\ d=k &= \frac{D+E+1}{2} \\ e &= \frac{A+2B+C+2}{4} \\ f=m &= \frac{B+2C+D+2}{4} \\ g=n &= \frac{C+2D+E+2}{4} \\ h=o &= \frac{D+2E+F+2}{4} \\ l &= \frac{E+F+1}{2} \\ p &= \frac{E+2F+G+2}{4} \end{aligned}$$

Mode 8: Horizontal Up prediction This mode can be used if I, J, K and L are available. Then:

$$\begin{aligned} a &= \frac{I+J+1}{2} \\ b &= \frac{I+J+K+2}{4} \\ c=e &= \frac{J+K+1}{2} \\ d=f &= \frac{J+K+L+2}{4} \\ g=h=i=j &= \frac{K+L+1}{2} \\ k=l=m=n=o=p &= L \end{aligned}$$

Note It is interesting to note that the choice between those modes is left to the implementation. No choice algorithm is described in [3]. A common shortcut would be to use DC prediction (mode 2) in all cases.

1.2.1.2 16×16 mode

There are 4 prediction modes for 16×16 intra coding. They are defined as presented in table 1.3.

0	Vertical
1	Horizontal
2	DC
3	Plane

Table 1.3: 16×16 prediction modes

For notation facilities, we denote $X_{i,j}$ the pixel at the i^{th} column and j^{th} row. i and j vary between -1 and -15. To set an example, in the previous notation, we would have $Q = X_{-1,-1}$, $A = X_{0,-1}$, $I = X_{-1,0}$ and $g = X_{2,1}$.

Mode 0: Vertical prediction The vertical prediction works in the same way as for 4×4 prediction. The pixels are derived from the pixels above the considered block, and copied vertically.

Thus we have:

$$X_{i,j} = X_{i,-1} \quad \forall i,j \in 0..15$$

Mode 1: Horizontal prediction The horizontal prediction works in the same way as for 4×4 prediction. The pixels are derived from the pixels to the left of the considered block, and copied horizontally.

Thus we have:

$$X_{i,j} = X_{-1,j} \quad \forall i,j \in 0..15$$

Mode 2: DC prediction If all the neighbours of the considered block are available, we have:

$$X_{i,j} = \frac{\sum_{x=0}^{15} X_{x,-1} + \sum_{y=0}^{15} X_{-1,y} + 16}{32} \quad \forall i,j \in 0..15$$

Else, if only the left neighbours are available, we have:

$$X_{i,j} = \frac{\sum_{x=0}^{15} X_{-1,j} + 8}{16} \quad \forall i,j \in 0..15$$

Similarly, if only the top neighbours are available, we have:

$$X_{i,j} = \frac{\sum_{x=0}^{15} X_{i,-1} + 8}{16} \quad \forall i,j \in 0..15$$

And finally, if no neighbours are available, then

$$X_{i,j} = 128 \quad \forall i,j \in 0..15$$

Mode 3: Plane prediction In this mode, we define $X_{i,j}$ as following:

$$X_{i,j} = \text{Clip1}\left(\frac{a + b(i-7) + c(y-7) + 16}{32}\right)$$

where Clip1 saturates the value between 0 and 255 and

$$\begin{aligned} a &= 16(X_{-1,15} + X_{15,-1}) \\ b &= \frac{5H + 32}{64} \\ c &= \frac{5V + 32}{64} \end{aligned}$$

where

$$\begin{aligned} H &= \sum_{x=0}^{7} (x+1)(X_{8+x,-1} - X_{6-x,-1}) \\ V &= \sum_{y=0}^{7} (y+1)(X_{-1,8+x} - X_{-1,6-y}) \end{aligned}$$

1.2.2 Chroma prediction

The chroma prediction is applied to macroblocks. Both chroma blocks (C_b and C_r) should use the same prediction mode. There are 4 prediction modes for chroma, presented in table 1.4. We use the same notations than precedently (16×16 luma prediction). However, as we use chroma here, we only have a 8×8 block (as there are 4 times less samples of chroma than luma).

0	DC
1	Horizontal
2	Vertical
3	Plane

Table 1.4: Chroma prediction modes

Mode 0: DC prediction If $X_{i,-1}$ and $X_{-1,j}$ are available for $i, j \in 0..3$, then:

$$Xi,j = \frac{\sum_{x=0}^{3} X_{x,-1} + \sum_{y=0}^{3} X_{-1,y} + 4}{8} \quad \forall i,j \in 0..3$$

Else, if $X_{i,-1}$ is available for $i \in 0..3$, but $X_{-1,j}$ is not available:

$$Xi,j = \frac{\sum_{x=0}^{3} X_{x,-1} + 2}{4} \quad \forall i,j \in 0..3$$

Similarly, if $X_{i,-1}$ is not available for $i \in 0..3$, but $X_{-1,j}$ is available:

$$Xi,j = \frac{\sum_{y=0}^{3} X_{-1,y} + 2}{4} \quad \forall i,j \in 0..3$$

Else, if no neighbour is available:

$$X_{i,j} = 128 \quad \forall i,j \in 0..3$$

If $X_{i,-1}$ is available for $i \in 4..7$, then:

$$Xi,j = \frac{\sum_{x=4}^{7} X_{x,-1} + 2}{4} \quad \forall i \in 4..7, j \in 0..3$$

Else, if $X_{-1,j}$ is available for $j \in 0..3$:

$$Xi,j = \frac{\sum_{y=0}^{3} X_{-1,y} + 2}{4} \quad \forall i \in 4..7, j \in 0..3$$

Else:

$$X_{i,j} = 128 \quad \forall i \in 4..7, j \in 0..3$$

This behaviour is adaptable for $i \in 0..3, j \in 4..7$.
$i, j \in 4..7$ work similarly to $i, j \in 0..3$.

Mode 1: Horizontal prediction This mode can be used if $X_{i,-1}$ is available for $i \in 0..7$. Then:

$$X_{i,j} = X_{i,-1}$$

Mode 2: Vertical prediction This mode can be used if $X_{-1,j}$ is available for $j \in 0..7$. Then:

$$X_{i,j} = X_{-1,j}$$

Mode 3: Plane prediction This mode can be used if $X_{i,-1}$ is available for $i \in 0..7$ and $X_{-1,j}$ is available for $j \in -1..7$. Then:

$$X_{i,j} = \text{Clip1}\left(\frac{a + b(i-3) + c(j-3) + 16}{32}\right) \quad \forall i, j \in 0..7$$

where

$$\begin{aligned} a &= 16(X_{-1,7} + X_{7,-1}) \\ b &= \frac{17H + 16}{32} \\ c &= \frac{17V + 16}{32} \end{aligned}$$

where

$$\begin{aligned} H &= \sum_{x=0}^{3}(x+1)(X(4+x,-1) - X(2-x,-1)) \\ V &= \sum_{y=0}^{3}(y+1)(X(-1,4+y) - X(-1,2-y)) \end{aligned}$$

1.3 Temporal prediction

1.3.1 Introduction

The temporal prediction relies on the fact that there is generally few difference between two consecutive images of a given video sample. A given image can then be encoded as a difference to a reference picture. The advantage of this is that the differences will be small, so the energy of the signal will lessen, and then need less bits to encode. Expressing the differences is done by assign up to 16 *motion vectors* to a macroblock.

1.3.2 Partition of a macroblock

To each macroblock a type is assigned. This type correspond to a partition of the macroblock. A 16×16 luma macroblock can be divided in

- one 16×16 subblock
- two 16×8 subblocks
- two 8×8 subblocks
- four 8×8 subblocks

Each of these subblocks can be divided again following the same schema.
The chroma macroblocks are divised similarly.

1.3.3 Motion estimation

For each macroblock part (as defined in previous section), one or two reference pictures are defined. These images are referenced as an index in a chosen subset of possible reference pictures. If only one reference picture is used (P macroblocks), the index of the picture in the first group of picture is set to 0.

The motion vector are then defined with the structures `mvd_l0[mbPartIdx][subMbPartIdx][compIdx]` and `mvd_l0[mbPartIdx][subMbPartIdx][compIdx]` where

- `mvd_l0` defines the vector referencing to the first picture
- `mvd_l1` defines the vector referencing to the second picture
- `mbPartIdx` is the index of the considered partition in the macroblock
- `subMbPartIdx` is the index of the considered subpartition in the partition
- `compIdx` is set to 0 for horizontal component and 1 for vertical component of the vector

It is worth noting that the search for reference picture is left to the implementation of the standard. No algorithm is provided by the standard itself.

The motion prediction can use values to a precision of one quarter of the distance between luma samples. If both vector components are integer, the corresponding sample can be used directly. Else, fractional sample values must be interpolated, as explained in 1.3.4.

1.3.4 Fractional sample interpolation

Figure 1.3 shows the notation we use to explain the fractional sample interpolation.

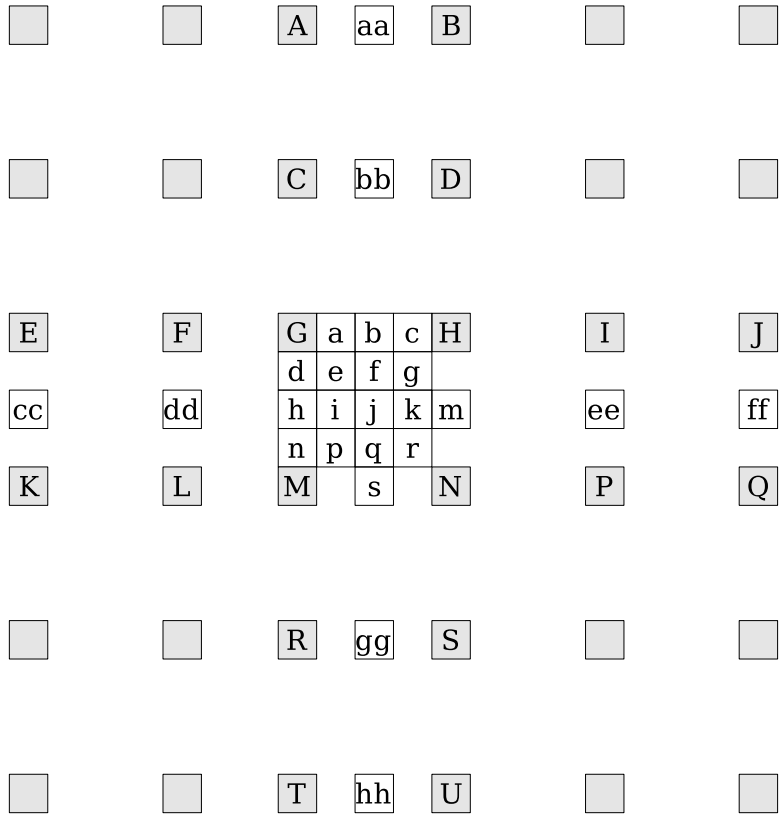

Figure 1.3: Fractional sample interpolation

We want to interpolate samples $a-k$ and $n-r$. b and h are the first samples calculated. Of course, all the final values are clipped to the range $0-255$.

$$\begin{aligned} b &= \frac{b_1+16}{32} \\ h &= \frac{h_1+16}{32} \end{aligned}$$

where

$$\begin{aligned} b_1 &= E-5F+20G+20H+5I+J \\ h_1 &= A-5C+20G+20M-5R+T \end{aligned}$$

cc, dd, ee, m_1 and ff are calculated similarly, and we obtain

$$\begin{aligned} j_1 &= cc-5dd+20h1+20m_1-5ee+ff \\ j &= \frac{j_1+512}{1024} \end{aligned}$$

The samples at the quarter positions are computed as following:

$$
\begin{aligned}
a &= \frac{G+b+1}{2} \\
c &= \frac{b+H+1}{2} \\
d &= \frac{G+h+1}{2} \\
f &= \frac{b+j+1}{2} \\
n &= \frac{h+M+1}{2} \\
i &= \frac{h+j+1}{2} \\
k &= \frac{j+m+1}{2} \\
e &= \frac{b+h+1}{2} \\
g &= \frac{b+m+1}{2} \\
p &= \frac{h+s+1}{2} \\
r &= \frac{m+s+1}{2}
\end{aligned}
$$

1.4 Deblocking filter

The block artefacts are the most annoying artefacts on a MPEG-encoded video in general. To partially avoid those artefacts, H.264/AVC introduces a deblocking filter in the loop. This filter is explained in [2]. It is to note that, as the filter is in the loop, the filtered images are used as references for prediction. Previous standards had an optional post-filtering approach - so the non-filtered images were used.

The filter is adaptive, that is to say the filter behavior changes depending on the situation. There are three levels of adaptability:

- On the slice level, characteristics of filtering can be adjusted
- On the block-edge level, decision is made about the strength of the filter
- On the sample level, some thresholds (defined later) can also be redefined to turn filtering on or off.

1.4.1 Edge level

The filter is applied to 4×4 luminance blocks or, more precisely, to the edges of these blocks. To every edge, we assign a Boundary-Strength coefficient Bs. Bs is determined as shown on table 1.5.

Block modes and conditions	Bs
One of the blocks is intra and the edge is a macroblock edge	4
One of the blocks is intra	3
One of the blocks has coded residuals	2
Difference of block motion ≥ 1 luma sample distance	1
Motion compensation from different reference frames	1
Else	0

Table 1.5: Bs parameter definition

A value of 4 means that a special mode of the filter is used. A value of 0 means that this edge is not filtered. For values between 1 and 3, the standard filter is applied.

1.4.2 Sample level

The aim of this filtering is to distinguish between blocking edges and edges of the image. The "true" edges should be filtered as less as possible, and the blocking edges should be filtered to reduce their visibility.

If we denote p_3, p_2, p_1, p_0, q_0, q_1, q_2 and q_3 a line of samples values, and the actual edge between p_0 and q_0, then the filtering will take place if and only if

$$\begin{aligned} \mid p_0 - q_0 \mid &< \alpha(\text{Index}_A) \\ \mid p_1 - p_0 \mid &< \beta(\text{Index}_B) \\ \mid q_1 - q_0 \mid &< \beta(\text{Index}_B) \end{aligned}$$

where

$$\begin{aligned} \text{Index}_A &= \min(\max(0, QP + \text{Offset}_A), 51) \\ \text{Index}_B &= \min(\max(0, QP + \text{Offset}_B), 51) \end{aligned}$$

and

$$\begin{aligned} \alpha(x) &= 0.8.(2^{x/6} - 1) \\ \beta(x) &= 0.5x - 7 \end{aligned}$$

1.4.3 Slice level

Offset_A and Offset_B can be defined as slice level to adjust the value of α and β. Transmitting negative offsets can help improving the quality of fine-grained details on the image, while transmitting positive offsets can help improving the quality for cases where the blocks are still visible after the standard filtering.

1.4.4 Filtering

1.4.4.1 Filtering strength

In each mode of the filter, there are again two modes: a normal mode and a strong mode. The strong mode is enabled if these conditions are true:

$$\mid p_2 - p_0 \mid < \beta(\text{Index}_B) \tag{1.1}$$
$$\mid q_2 - q_0 \mid < \beta(\text{Index}_B) \tag{1.2}$$

Case 1: $Bs = 1$ **to** 3 The filtered values for luminance p'_0 and q'_0 are calculated as

$$\begin{aligned} p'_0 &= p_0 + \Delta_0 \\ q'_0 &= q_0 - \Delta_0 \end{aligned}$$

where

$$\Delta_0 = \min(\max(-c_0, \Delta_{0i}, c_0))$$

where

$$\Delta_{0i} = (4(q_0 - p_0) + (p_1 - q_1) + 4) << 3$$

c_0 is set equal to c_1 and incremented for each of conditions (1.1) and (1.2) that holds true. c_1 is defined in a table page 152 of [4].

If condition (1.1) is true, the filtered value p'_1 is

$$p'_1 = p_1 + \Delta_{p1}$$

and, if condition(1.2) is true, we have similarly

$$q'_1 = q_1 + \Delta_{q_1}$$

where

$$\begin{aligned}\Delta_{p1} &= \min(\max(-c_1, \Delta_{p1i}), c_1)\\ \Delta_{q1} &= \min(\max(-c_1, \Delta_{q1i}), c_1)\end{aligned}$$

where

$$\begin{aligned}\Delta_{p1i} &= (p_2 + ((p_0 + q_0 + 1) >> 1) - 2p_1) >> 1\\ \Delta_{q1i} &= (q_2 + ((p_0 + q_0 + 1) >> 1) - 2q_1) >> 1\end{aligned}$$

For the chrominance filtering, only p_0' and q_0' are computed, with $c_0 = c_1 + 1$ - so there is no need to evaluate conditions (1.1) and (1.2).

Case 2: $Bs = 4$ If condition (1.1) is true and

$$| p_0 - q_0 | < (\alpha(\text{Index}_A) >> 2) + 2 \tag{1.3}$$

is true, then we apply this filter on luminance:

$$\begin{aligned}p_0' &= (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) >> 3\\ p_1' &= (p_2 + p_1 + p_0 + q_0 + 2) >> 2\\ p_2' &= (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) >> 3\end{aligned}$$

Otherwise, or for chrominance filtering, only p_0 is modified:

$$p_0' = (2p_1 + p_0 + q_1 + 2) >> 2$$

Similarly, if (1.2) and (1.3) hold true, we modify luminance with:

$$\begin{aligned}q_0' &= (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) >> 3\\ q_1' &= (q_2 + q_1 + q_0 + p_0 + 2) >> 2\\ q_2' &= (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) >> 3\end{aligned}$$

or, otherwise, with

$$q_0' = (2q_1 + q_0 + p_1 + 2) >> 2$$

1.5 Encoding

Human eyes notice much more low spatial frequency than high spatial frequency information. So we can assume that some high spatial frequency information can be discarded - this fact is used by MPEG video compression. A frequency domain representation of a given picture is then needed.

In "classical" MPEG4 as defined in [3], this is the role of DCT (Direct Cosine Transform). The information within a macroblock is converted by the DCT algorithm into the frequency domain:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{i=0}^{7} f(i,j)\cos\frac{\pi u(2i+1)}{16}\cos\frac{\pi v(2j+1)}{16}$$

where

$$C(x) = \begin{cases}\frac{1}{\sqrt{2}} & \text{if } x = 0\\ 1 & \text{otherwise}\end{cases}$$

After the transformation, the top left value of the block represents the DC level (think of this as the average brightness) of the block. The value immediately to the right of this represents low frequency horizontal information. The value in the top right represents high frequency horizontal information. Similarly, the bottom left value represents high frequency vertical information.

The major issue with this transform is that it implies irrational numbers. To circumvent this, [1] explains how an integer transform can be used instead. Instead of using a DCT on 8×8 blocks, we can use the following transform $X = Hx$ where

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

on 4×4 blocks. This solves the problem of irrationnality, transforms floating-point operations into integer operations, and is exactly inversible.

The DCT (or its replacement) in itself represents exactly the same quantity of information as the non-transformed block. However, it is much easier to compress this information as the majority of coefficient is near-zero. To compress the information, we use quantization, as defined also in [1]:

$$X_q(i,j) = \text{sign}\{X(i,j)\} \left[(\mid X(i,j) \mid A(Q_M, i, j) + f 2^{17+Q_E} >> (17 + Q_E) \right]$$

where

- $X_q(i,j)$ is the quantized coefficient for pixel (i,j) of the considered block.
- $X(i,j)$ is the non-quantized coefficient (after the DCT) for pixel (i,j) of the considered block.
- $Q_M = Q \mod 6$ and $Q_E = Q/6$, where Q is the quantization factor.
- $A(Q_M, i, j) = M(Q_M, r)$ where

$$\begin{cases} r = 0 & \text{for } (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\} \\ r = 1 & \text{for } (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\} \\ r = 2 & \text{otherwise} \end{cases}$$

- M is defined as:

$$M = \begin{pmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{pmatrix}$$

- f is a parameter chosen by the encoder, and is typically in the range $[0, 1/2]$

The decoding is done via inverse transforms, namely

$$X_r(i,j) X_q(i,j) B(Q_M, i, j) << Q_E$$

for reconstruction after quantization, where $B(Q_M, i, j) = S(Q_M, r)$ where r is defined as preceding, and S is defined as

$$S = \begin{pmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{pmatrix}$$

and

$$x_r = \tilde{H}_{inv} X_r$$

for inverse "DCT-like" transform, where

$$\tilde{H}_{inv} = \begin{pmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{pmatrix}$$

$\tilde{H}_{inv}$ is a scaled inverse of H, i.e.

$$\tilde{H}_{inv}\text{diag}\left\{\frac{1}{4}\frac{1}{5}\frac{1}{4}\frac{1}{5}\right\}H = I$$

1.6 Entropy coding

This is the last step of the encoding process. The blocks after transformation and quantization contain many 0s. The bigger coefficients are usually located at the top-left of the block, so the coefficients are reordered via a zigzag scan like shown on figure 1.4.

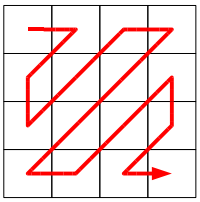

Figure 1.4: Zigzag scan

[5] gives the example of a typical zigzag scan of quantized transform coefficients, that illustrates well the "many-0s" property: `7 6 -2 0 -1 0 0 1 0 0 0 0 0 0 0 0`. Instead of directly coding this sequence, a number of statistical data is encoded:

- Number of "trailing 1s" ($T1s$): number of coefficients with absolute value equalt to 1 at the end of the scan (in the example, $T1s = 2$).
- Number of nonzero coefficients (N) (in the example, $N = 5$)
- Value of coefficients: coding of the values of the coefficient, in reverse order (in the example, the coefficients to be coded are, in this order, -2, 6 and 7).
- Sign information: this is the information concerning the coefficients with absolute value equal to 1.
- Total Zeroes: specifies the number of zeros between the last non-zero coefficient of the scan and its start (in the example, $TotalZeroes = 3$).
- RunBefore: specifies how the zeroes are distributed between the non-zero coefficients. We code the 0s before the last coefficient, in our exemple, 2. The number of 0s before the second-last coefficient is 1. As we have coded all our zeroes, we do not need other information.

The tables for coding the informations are coded in reference to VLC (variable length coding) tables, whose elements are switched during the coding process to provide a best suit to encoded data.

Chapter 2

Basic task splitting

2.1 Data structures

- Picture: structure to handle a picture, containing slices
- Slice: structure to handle a slice, containing macroblocks
- Macroblock: structure to handle a macroblock (16×16), containing chroma and luma
- Block: structure to handle a block (4×4), containing chroma and luma
- Sample: a group of 4 pixels (used for filtering)

2.2 Spatial prediction

2.2.1 Luma 4×4 prediction

- `decide_prediction_mode_4x4(Block * b, Block * topright, Block * top, Block * topleft, Block * left, Block bottomleft)` - sets a pointer to prediction function
- `predict_4x4_luma(Block * b, int[9] top, int[9] left)` - calculates the prediction into `mb` depending on the neighbouring blocks. Is actually a pointer to different prediction modes. There is a slight redundancy for the coefficient Q which is available in `top` and in `left`. Note : must be careful for implementation, as indice -1 is difficult to have in C...
  - `predict_4x4_luma_0(Block * b, int[9] top, int[9] left)` - prediction for mode 0 (vertical)
  - `predict_4x4_luma_1(Block * b, int[9] top, int[9] left)` - prediction for mode 1 (horizontal)
  - ...
  - `predict_4x4_luma_8(Block * b, int[9] top, int[9] left)` - prediction for mode 8 (horizontal up)

2.2.2 Luma 16×16 prediction

- `decide_prediction_mode_16x16(Macroblock * mb, Macroblock * topright, Macroblock * top, Macroblock * topleft, Macroblock * left, Macroblock bottomleft)` - sets a pointer to prediction function
- `predict_16x16_luma(Macroblock * mb, int[17] top, int[17] left)` - calculates the prediction into `mb` depending on `top` and `left`. Is actually a pointer to different prediction modes.
  - `predict_16x16_luma_0(Macroblock * mb, int[17] top, int[17] left)` - prediction for mode 0 (vertical)

  - `predict_16x16_luma_1(Macroblock * mb, int[17] top, int[17] left)` - prediction for mode 1 (horizontal)
  - `predict_16x16_luma_2(Macroblock * mb, int[17] top, int[17] left)` - prediction for mode 2 (DC)
  - `predict_16x16_luma_3(Macroblock * mb, int[17] top, int[17] left)` - prediction for mode 3 (plan)

2.2.3 Chroma prediction

- `decide_prediction_mode_chroma(decide_prediction_mode_16x16(Macroblock * mb, Macroblock * top, Macroblock * topleft, MacroBlock * left)`
- `predict_chroma(Macroblock * mb, int[9] top, int[9] left)` - calculates the prediction into `mb` depending on `top` and `left`. Is actually a pointer to different prediction modes.
  - `predict_chroma_0(Macroblock * mb, int[9] top, int[9] left)` - prediction for mode 0 (DC)
  - `predict_chroma_1(Macroblock * mb, int[9] top, int[9] left)` - prediction for mode 1 (horizontal)
  - `predict_chroma_2(Macroblock * mb, int[9] top, int[9] left)` - prediction for mode 2 (vertical)
  - `predict_chroma_3(Macroblock * mb, int[9] top, int[9] left)` - prediction for mode 3 (plan)

2.3 Temporal prediction

- `int decide_partition(Macroblock *mb)` - decides the partition and subpartition of the macroblock
- `Image ** get_reference_list0(Image * i)` - gets the first list of images useable as references for `i`
- `Image ** get_reference_list1(Image * i)` - gets the second list of images useable as references for `i`
- `int get_refIdxl0(Macroblock *mb, int index)` - decides the reference picture among the first list
- `int get_refIdxl1(Macroblock *mb, int index)` - decides the reference picture among the second list
- `int get_motion_estimation_h(Macroblock *mb, int partIdx, int subPartIdx)` - gets horizontal component of the motion vector
- `int get_motion_estimation_v(Macroblock *mb, int partIdx, int subPartIdx)` - gets vertical component of the motion vector
- interpolation (?)

2.4 Deblocking filter

- `int boundary_strength(Block * b1, Block * b2, int direction)` - calculates the boundary strength of the filter for the edge between block `b1` and `b2`. `direction` is set to 0 if the considered edge is horizontal, 1 if the considered edge is vertical.
- `bool sample_filterable(Sample * s1, Sample * b2)` - applies algorithm defined in 1.4.2.
- `filter_sample_luma(Sample * s1, Sample * s2)` - applies algorithm defined in 1.4.4 for luma components.
- `filter_sample_chroma(Sample * s1, Sample * s2)` - applies algorithm defined in 1.4.4 for chroma components.

2.5 Encoding

- `dct(Block * b1)` - applies dct to `b1`
- `quantize(Block * b1)` - applies quantization function to `b1`
- `idct(Block * b1)` - applies inverse dct to `b1`
- `dequantize(Block * b1)` - applies dequantization function to `b1`

2.6 Entropy coding

- `int * zigzag_scan(Block *b)`
- `int get_trailing_1s(int * block)`
- `int * get_coeff_values (int * block)`
- `int * get_sign_information (int * block)`
- `int get_total_zeroes(int * block)`
- `int * get_run_before(int * block)`

Bibliography

[1] Henrique S. Malvar, Antti Hallapuro, Marta Karczevicz, and Louis Kerofsky. Low-complexity transform and quantization in h.264/avc. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), July 2003.

[2] Peter Most, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz. Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), July 2003.

[3] Hiroshi Watanabe and Yukiko Ogura. Information technology – coding of audio-visual objects – part 2: Visual (iso/iec 14496-2. Technical report, ISO/IEC MPEG, 2001.

[4] Thomas Wiegand, Gary Sullivan, and Ajay Luthra. Draft itu-t recommendation and final draft international standard of joint video specification (itu-t rec. h.264 | iso/iec 14496-10 avc. Technical report, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, May 2003.

[5] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), July 2003.