Video coding

Concepts and notations.

A video signal consists of a time sequence of images. Typical frame rates are 24, 25, 30, 50 and 60 images per seconds.

Each image is either sent *progressively* (the whole image at once) or *interlaced* (half of the image at a time using double the frame rate, first all even lines and then all odd lines.

The whole image is called a *frame* and a half image is called a *field*.

Colour format

The colour information is usually stored as one luma and two chroma signals (YCbCr)

The chroma signals are usually sampled more coarsely than the luma signal.

4:4:4 No subsampling of the chroma signals.

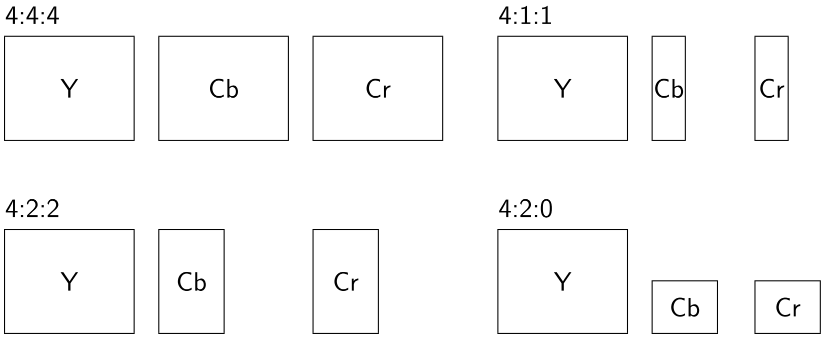
4:2:2 Chroma signals subsampled a factor 2 horizontally.

4:1:1 Chroma signals subsampled a factor 4 horizontally.

4:2:0 Chroma signals subsampled a factor 2 both horizontally and vertically.

The first three were already used for analog video signals, and the numbers then referred to the relative bandwith of the different signals.

Subsampling



4:2:0 is the most common format when distributing video to the end consumer.

Motion JPEG (2000)

Code each frame in the sequence using JPEG or JPEG 2000.

Does not use any dependence between images.

One application where Motion JPEG 2000 is used is digital cinema, where the video is stored at high resolution (up to 4096×2160) and relatively moderate compression ratio is used.

DCI - Digital Cinema Initiatives

Three levels of resolution

1. Max resolution 4096×2160, 24 frames per second.
2. Max resolution 2048×1080, 48 frames per second.
3. Max resolution 2048×1080, 24 frames per second.

The pixels are quadratic.

12 bits quantization per colour component. No subsampling of the chroma signals. Thus, level 1 has an uncoded data rate of 7.6 Gbit/s.

The image date is coded using Motion JPEG 2000. Maximum rate is 250 Mbit/s after compression.

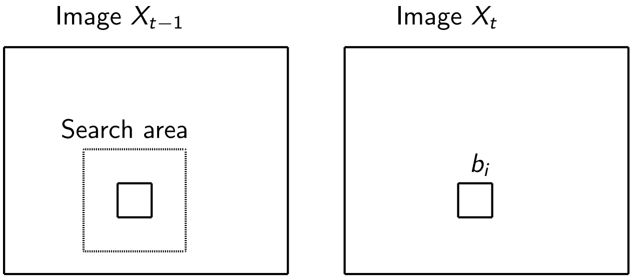
Hybrid coding

Consecutive images in a sequence are usually very similar, which can be exploited when coding the video.

Most current video coding methods use *hybrid coding*, where an image is first predicted in the time dimension and then the prediction error is transform coded in the image plane.

To compensate for camera movements, zooming and object motion in the images, block based *motion compensation* is used.

Motion compensation

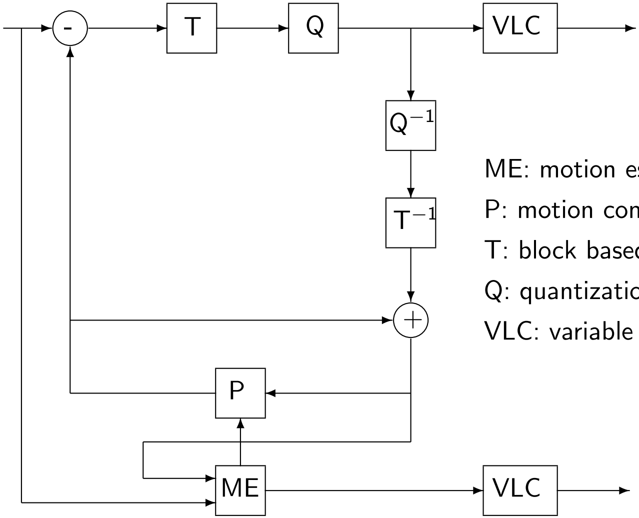


We want to code the image X_t using prediction from the previous image X_{t-1}.

Motion estimation: For each block b_i in the image X_t we look for a block $\tilde{b}_i$ in the previous image X_{t-1} that is as similar to b_i as possible. The search is performed in a limited area around b_i:s position. The result is a *motion vector* that needs to be coded and sent to the receiver.

The prediction errors (differences between b_i and $\tilde{b}_i$) for each block i are coded using transform coding.

Hybrid coder using motion compensation



ME: motion estimation

P: motion compensated prediction

T: block based transform

Q: quantization

VLC: variable length coding

Motion compensation

The receiver can decode an image $\hat{X}_t$, using the previous decoded image $\hat{X}_{t-1}$, the received motion vectors and the decoded difference block.

In order to avoid error propagation, the encoder should also do motion compensated prediction using a decoded version $\hat{X}_{t-1}$ of the previous image instead of X_{t-1}.

At regular intervals an image that is coded independently of surrounding frames are sent.

Block sizes

How should the blocksize in the motion compensation be chosen?

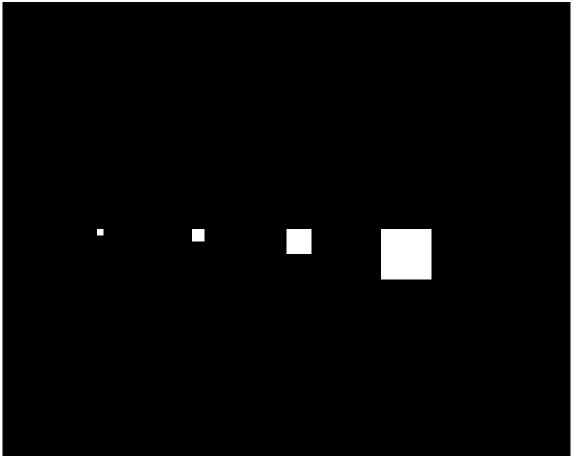
The smaller blocks that are used, the better the prediction. However, this will give more data in the form of motion vectors.

Most older coding standards use a block size of 16×16 pixels for motion compensation. In the HEVC standard the largest block size for motion compensation is 64×64 pixels.

Modern standards allow adaptive blocksizes for motion compensation.

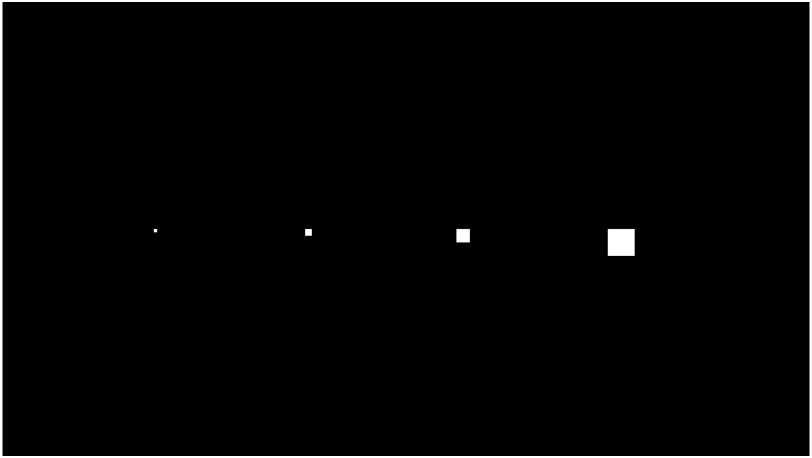
The blocks used for motion compensation are usually referred to as *macroblocks* (HEVC: *Coding Tree Unit*).

Resolution 720×576



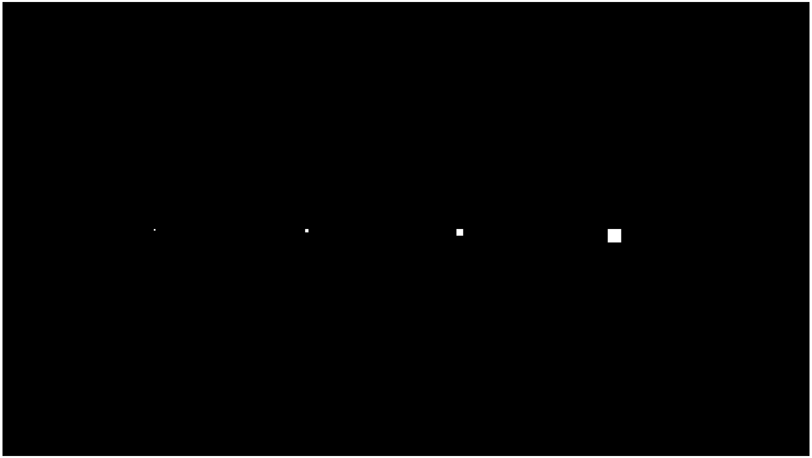
Block sizes 8×8, 16×16, 32×32 and 64×64.

Resolution 1920×1080



Block sizes 8×8, 16×16, 32×32 and 64×64.

Resolution 3840×2160



Block sizes 8×8, 16×16, 32×32 and 64×64.

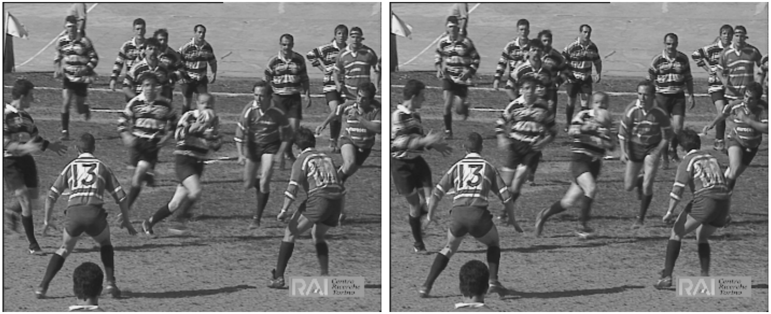
Motion estimation

The motion estimation is often one of the most time consuming parts of the coder, since large search areas might be needed to find good predictions.

Hardware support might be needed to get realtime performance. For instance, GPU implementations could allow parallelization.

The search procedure can be speeded up by for instance using a logarithmic search instead of a full search. This will come at a small reduction in compression, since we're not guaranteed to find the best motion vector.

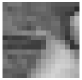
Example



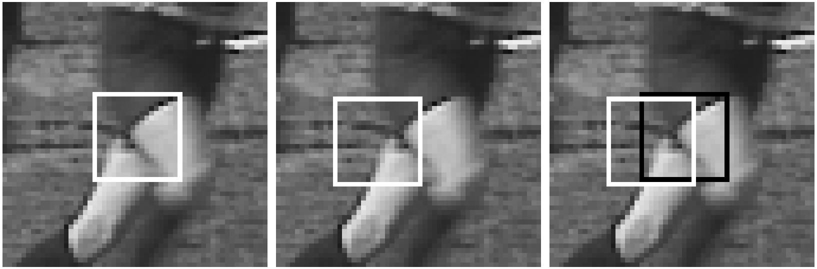
Two consecutive frames from a video sequence.

The camera is panning to the right, which means that the whole image seems to be moving to the left. The player with the ball is moving to the right.

A single block

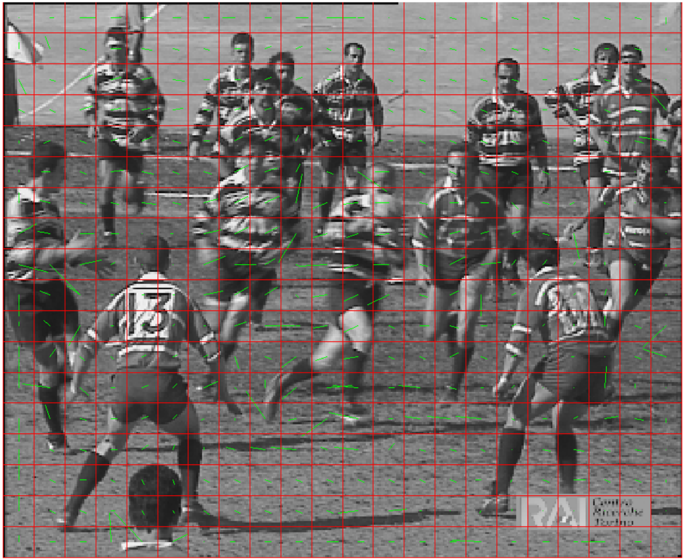
Block to be predicted: 

Search area in the previous frame, centered around the same position (±20 pixels) and position for the best match.

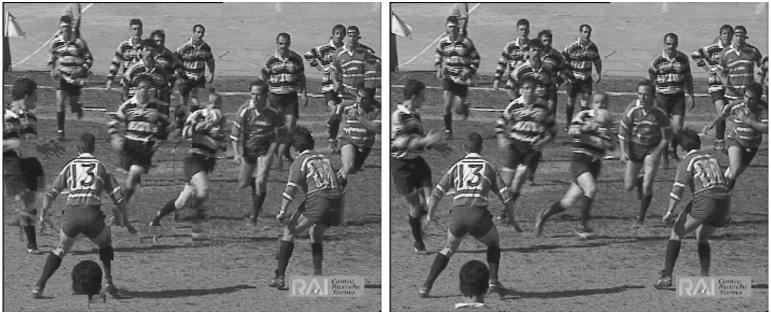


The motion vector is the difference in position between the center and the best match: (-7,1).

Example, motion vectors

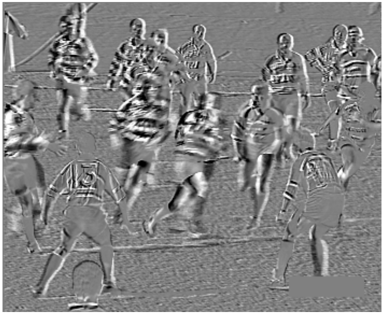


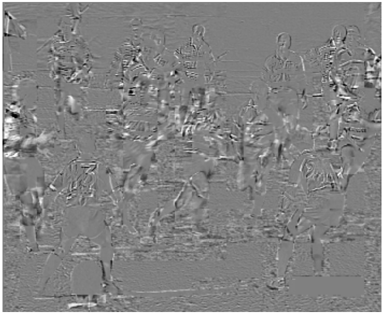
Example



Motion compensated prediction of frame 2 and the original frame 2.

Example





Prediction error if no motion compensation was used (all motion vectors set to zero) and prediction error when motion compensation is used.

The motion compensation gives a prediction error image that is easier to code, ie gives a lower rate at the same distortion or lower distortion at the same rate.

Bidirectional prediction

All video coding standards since MPEG-1 have supported bidirectional prediction, where you can do motion compensated prediction from a previous frame, a future frame or both.

Doing this gives you a better chance of finding a good prediction. For instance, if objects are moving around in the scene or the camera is panning, the area you are predicting might not be visible in the previous frame.

When doing bidirectional prediction, you have to make sure that two frames are not predicted from each other. The simplest way to make sure if this is to not allow bidirectional frames to be used for prediction of other frames.

If the coder uses bidirectional coding, the frames need to be transmitted in a different order than they are displayed, since the decoder needs to have access to future frames in order to be able to decode.

Frame types

- I Intra. The frame is coded independent of surrounding frames.
- P Predicted. The frame is coded using motion compensation from an earlier frame.
- B Bidirectional. The frame is coded using motion compensation from an earlier and/or a later frame.

B frames typically give higher compression ratio than P frames.

Usually we can also choose coding method for each macroblock. In an I frame all macroblocks need to be coded as I blocks, in a P frame the macroblocks can be coded as I or P blocks and in a B frame the macroblocks can be coded as I, P or B blocks.

Frame reordering

Suppose we code every 12:th frame as an I frame and that we have two B frames between each pair of I/P frames, so that the display order of coded frames is

$$I_0\ B_1\ B_2\ P_3\ B_4\ B_5\ P_6\ B_7\ B_8\ P_9\ B_{10}\ B_{11}\ I_{12}\ B_{13}\ B_{14}\ P_{15}\quad \dots$$

P_3 is predicted from I_0, P_6 is predicted from P_3 et c.

B_1 and B_2 are predicted from I_0 and P_3, B_4 and B_5 are predicted from P_3 and P_6 et c.

The coder must code and transmit the frames in the order

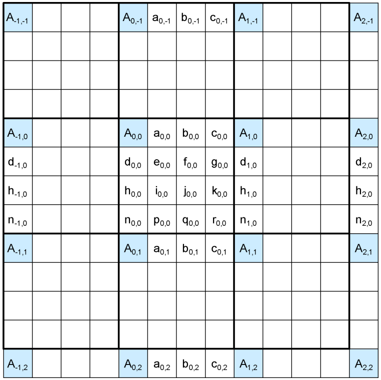
$$I_0\ P_3\ B_1\ B_2\ P_6\ B_4\ B_5\ P_9\ B_7\ B_8\ I_{12}\ B_{10}\ B_{11}\ P_{15}\ B_{13}\ B_{14}\quad \dots$$

in order for the decoder to be able to decode correctly.

Subpixel accuracy

To achieve a better motion compensated prediction it is very common to have subpixel accuracy in the motion vectors.

Separable linear filters are used to interpolate values between the actual pixels.


$A_{-1,-1}$ $A_{0,-1}$ $a_{0,-1}$ $b_{0,-1}$ $c_{0,-1}$ $A_{1,-1}$ $A_{2,-1}$
$A_{-1,0}$ $A_{0,0}$ $a_{0,0}$ $b_{0,0}$ $c_{0,0}$ $A_{1,0}$ $A_{2,0}$
$d_{-1,0}$ $d_{0,0}$ $e_{0,0}$ $f_{0,0}$ $g_{0,0}$ $d_{1,0}$ $d_{2,0}$
$h_{-1,0}$ $h_{0,0}$ $i_{0,0}$ $j_{0,0}$ $k_{0,0}$ $h_{1,0}$ $h_{2,0}$
$n_{-1,0}$ $n_{0,0}$ $p_{0,0}$ $q_{0,0}$ $r_{0,0}$ $n_{1,0}$ $n_{2,0}$
$A_{-1,1}$ $A_{0,1}$ $a_{0,1}$ $b_{0,1}$ $c_{0,1}$ $A_{1,1}$ $A_{2,1}$
$A_{-1,2}$ $A_{0,2}$ $a_{0,2}$ $b_{0,2}$ $c_{0,2}$ $A_{1,2}$ $A_{2,2}$

Intra prediction

Starting with H.264, prediction is being done for I blocks too. In previous standards simple DC prediction from one transform block to the next was used, but now more advanced prediction is done before the transform.

The block to be coded is predicted from surrounding, already coded pixels (known to the decoder). Usually some form of linear interpolation is used to find a a prediction block and the prediction error is then transform coded.

Post processing

At low rates, he decoded frames will suffer from typical transform coding artefICTs, like blocking artifacts and ringing artifacts close to sharp edges. This will affect the performance of coder, because of the introduction of false edges in the frames.

Starting in H.264, a post processing step was added in the coder, where these coding artifacts are filtered out.

Scalable coding

Scalable coding is a way of including video of several quality levels into the same bit stream. This gives us the ability to just transmit and/or decode a part of the bit stream and still get a complete video sequence, just with lower quality.

The bitstream contains a base layer of lower quality video and then several enhancement layers that will increase the quality.

This can be done in several different ways:

- Spatial scalability. The base layer contains a low resolution version of the video and the enhancement layer(s) add higher resolution(s).
- SNR scalability. The base layer contains a sequence of full resolution and low quality. The enhancement layer(s) adds more details to the frames.
- Temporal scalability. The base layer contains a sequence of full resolution but low frame rate. The enhancement layer(s) adds more frames.

Standards

The two large standardization organisations that develop video coding standards are ITU-T (International Telecommunication Union) and MPEG (Moving Picture Experts Group). MPEG is a cooperation between ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission).
ITU-T and MPEG have worked together on several standards.

- 1990: H.261
- 1991: MPEG-1
- 1994: MPEG-2/H.262
- 1995: H.263
- 1998: MPEG-4
- 2003: MPEG-4 AVC/H.264
- 2013: HEVC/MPEG-H/H.265
- 2020: VVC/MPEG-I/H.266

Non MPEG/ITU coding methods

There are of course other coding methods, besides the MPEG and ITU standards.

(Groups of) companies like Microsoft (Windows Media), Google (VP8, VP9) and AOMedia (AV1) have developed their own coding methods.

AOMedia is an alliance of several information technology, hardware and content provider companies. It was formed in partly in response to licensing difficulties with current standards, with the goal of creating a royalty-free alternative to H.264 and HEVC.

These coders are also hybrid coders, very similar to MPEG and ITU standards.

H.261

Low rate coder suitable for videoconferencing and video telephony. Typical rates 64-128 kbit/s (ISDN). The standard can handle rates up to 2 Mbit/s.

Based on motion compensation of macroblocks of 16×16 pixels and a DCT on blocks of 8×8 pixels.

Frame size 352×288 (CIF), or 176×144 (QCIF).

Colour format 4:2:0. Each macroblock thus contains 4 luma transform blocks and 2 chroma transform blocks.

Low framerate, typically 10-15 frames/s

Only I and P frames (called *INTRA mode* and *INTER mode* in the H.261 standard).

H.261, cont.

Motion vectors can be maximum ± 15. The difference to the motion vector of the previous block is coded using variable length coding, short codewords for small differences.

32 different quantizers to choose between, uniform with varying step sizes. It is possible to choose quantizer for a whole group of 11×3 macroblocks, in order to save bits. For each macroblock we also send information about which of the 6 transform blocks that contain any non-zero components.

The quantized blocks are zigzag scanned and runlength coded. The most common pairs (runlength, non-zero component) are coded using a tabulated variable length code, the other pairs are coded using a 20 bit fixed length code.

Gives acceptable image quality at 128 kbit/s for scenes with small motion.

H.263

Expanded variant of H.261.

- Possibility of longer motion vectors.
- Motion compensation using halfpixel precision (interpolation)
- More resolutions possible, eg 4CIF (704×576).
- Arithmetic coding
- PB frames (simplified version of B frames)

Compared to H.261 it gives the same quality at about half the rate.

MPEG-1

Similar to H.261, MPEG-1 uses motion compensation on macroblocks of 16×16 pixels and DCT on blocks of 8×8 pixels.

Random access is desired, ie the possibility to jump anywhere in a video sequence and start decoding, so I frames are used at regular intervals.

MPEG-1 is the standard where B frames where introduced, where the prediction can use both earlier and future I or P frames. Other B frames are never used for prediction. If the coder uses B frames, the frames need to be transmitted in a different order than they are displayed, since the decoder needs to have access to future frames in order to be able to decode.

MPEG-1, cont.

The motion compensation allows arbitrarily large motion vectors and halfpixel precision.

The quantization is similar to the one in JPEG, using quantization matrices. The standard matrix for I blocks look like

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

MPEG-1, cont.

Standard quantization matrix for P and B blocks:

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

MPEG-1, cont.

The quantization matrices are scaled with a factor that can change value between each macroblock, which is used for rate control.

Luma and chroma blocks have separate quantization matrices. It is possible to send other quantization matrices in the coded bitstream.

The quantized coefficients are zigzag scanned and the zeros are runlength encoded. The pairs (runlength, non-zero component) are coded using fixed variable length codes.

MPEG-1 is for instance used in VideoCD. Resolution 352×288 (25 frames/s) or 352×240 (30 frames/s). Rates 1-1.5 Mbit/s.

MPEG-2 (H.262)

Almost identical to MPEG-1. A MPEG-2 decoder should be able to decode MPEG-1 streams.

Supports higher resolution and higher rates than MPEG-1.

Supports coding fields separately (MPEG-1 only codes complete frames)

Typical formats for MPEG-2

- 720×576, 25 frames/s or 720×480, 30 frames/s (DVD, DVB)
- 1280×720, 1920×1080 (HDTV, Blu-ray)

Profiles and levels

A *profile* defines a subset of possible algorithms that can be used when coding.

A *level* sets limits on numerical parameters (eg resolution, frame rate, length of motion vectors, data rate).

In MPEG-2 there are 5 profiles (Simple, Main, SNR Scalable, Spatially Scalabe, High) and 4 levels (Low, Main, High 1440, High).

Profiles/levels

Some examples:

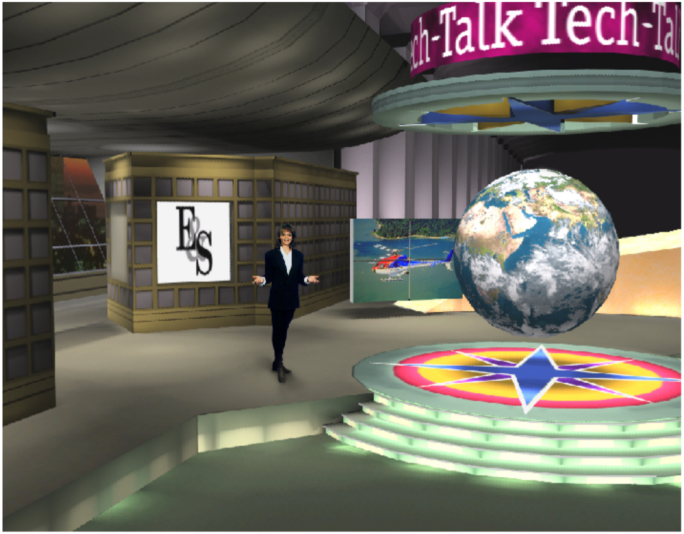
- Main profile, low level:
  Max resolution 352×288, 30 frames/s. Max rate 3 Mbit/s. Colour format 4:2:0.
- Main profile, main level:
  Max resolution 720×576, 30 frames/s. Max rate 10 Mbit/s. Colour format 4:2:0.
- High profile, high level:
  Max resolution 1920×1152, 60 frames/s. Max rate 100 Mbit/s. Colour format 4:2:2.

MPEG-4

The MPEG-4 standard is large standard that covers lots of multimedia coding methods (still images, video, wireframes, graphics, general audio, speech, synthetic audio, et c).

A scene is described, containing a number of image and audio sources. Each source is coded using a suitable coding method. In the decoder all sources are put together and rendered as a scene.

Example: Scene in MPEG-4



MPEG-4, cont.

Even though the standard covers lots of coding methods, the only parts that are commonly used are the general video and audio coding methods.

The first video coding standard defined by MPEG-4 is very similar to previous MPEG standards, with some extensions that can reduce the rate, such as arithmetic coding and quarterpixel motion vector resolution.

MPEG-4, cont.

- Still images
  Still images in MPEG-4 can be coded using a subband coder (wavelet coder) using zero-trees.
- Sprites
  A *sprite* in MPEG-4 is a still image in the background of a videosequence, often much larger than the video itself, so that the camera can pan over it. By using sprites, the background can be transmitted just once, so we don't have to send it for each frame.
- Synthetic objects
  Human faces can be described using a threedimensional wireframe model and corresponding texture. The wireframe can then be animated at a very low rate (basically we only have to send information about how the wireframe moves). The texture only needs to transmitted once. This is called *model based coding*.

MPEG-4, audio coding

Several different audio coding methods are supported.

- General waveform coding of audio (AAC).
- Speech coding.
- Text-to-speech. Support for synthesizing speech from text. Can be syncronized with the animation of wireframe models.
- Music description language. Describes what instruments are used and what notes they are playing (compare to MIDI).

H.264/MPEG-4 AVC

An extension to MPEG-4 is H.264 (also known as MPEG-4 Advanced Video Coding and MPEG-4 part 10) and was developed in cooperation between ITU-T and MPEG.

H.264 is one of the coding methods used on Blu-ray discs (MPEG-2 and VC-1 are also supported) and for transmission of HDTV material according to the DVB standards (MPEG-2 is also supported).

The first variant of H.264 came in 2003. Several extensions have been added later, such as 3D and multiview coding.

Comparison to other MPEG standards

Simlar to earlier MPEG standards, H.264 is a hybrid coder, where motion compensated prediction from earlier (and later) frames is used and where the prediction error is coded using transform coding.

The coder uses macroblocks of 16×16 pixels.

The macroblocks can be coded as I, P or B blocks (ie without prediction, with prediction from an earlier frame or with prediction from both earlier and later frames).

The whole frame does not need to be coded in the same way. Each frame can be split into parts (slices) and each slice can be of type I, P or B. Also on the macroblock level we can have different types of macroblocks in a slice. For an I slice all macroblocks have to be of type I, for a P slice the macroblocks can be of type I or P and for a B slice the macroblocks can be of type I, P or B.

H.264, cont.

Apart from doing motion compensation on whole macroblocks (16×16 pixels) we can also do it on smaller blocks (16×8, 8×16, 8×8, 4×8, 8×4 and 4×4). The coder thus has the option of splitting each macroblock into smaller parts if it is not possible to do a good prediction on the big block.

Unlike the other MPEG standards that use a DCT of size 8×8, H.264 uses an integer transform of size 4×4 according to

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

Note that the integer transform is not normalized, but this is compensated for in the quantization.

H.264, cont.

Each macroblock of 16×16 pixels is split into 16 transform blocks for the luma and 4 transform blocks for each chroma part (assuming 4:2:0 format).

The DC levels of the transform blocks are then additionally transformed using a DWHT (4×4 for the luma, 2×2 for the chromas).

The transform components are the quantized uniformly and source coded. There are several source coding methods that can be used.

H.264, cont.

In the extensions of H.264 support for larger transform blocks have been introduced. (8×8, 8×4 and 4×8). The 8-point transform looks like

$$\begin{pmatrix} 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 \\ 19 & 15 & 9 & 3 & -3 & -9 & -15 & -19 \\ 17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 \\ 9 & 3 & -19 & -15 & 15 & 19 & -3 & -9 \\ 13 & -13 & -13 & 13 & 13 & -13 & -13 & 13 \\ 15 & -19 & -3 & 9 & -9 & 3 & 19 & -15 \\ 7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 \\ 3 & -9 & 15 & -19 & 19 & -15 & 9 & -3 \end{pmatrix}$$

As can be seen, this transform is not normalized either, but this is compensated for in the quantization.

H.264, cont.

In H.264 it is allowed to do prediction from B slices, which is not allowed in the earlier MPEG standards. In order to avoid causality problems, the coder must make sure that two B slices are not predicted from each other.

The number of reference frames for the motion compensation can be up to 16 (unlike other MPEG standards where at most two reference frames can be used, one earlier I/P frame and one later). This gives an even better chance for the coder to find a good prediction for each macroblock.

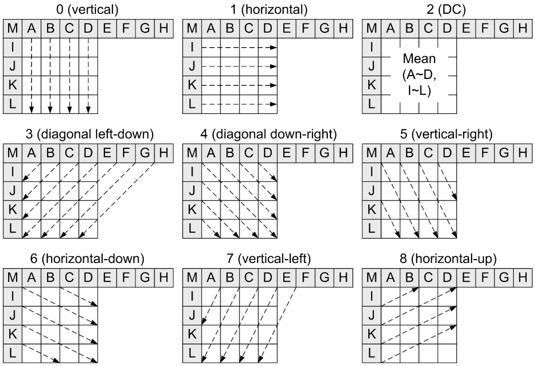
In H.264 there is also support for weighted prediction, ie using a prediction coefficient and not just use pixel differences.

H.264, cont.

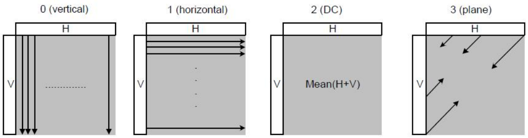
Even for I blocks prediction is used. This prediction uses pixels in surrounding, already coded blocks. The prediction is calculated as a linear interpolation from the surrounding pixel values. Either one prediction for the whole macroblock is used, which can be done in 4 different ways, or the luma macroblock is split into 16 small blocks of 4×4 pixels. The prediction for each of the small blocks can be done in 9 different ways. For the chroma blocks only the simple prediction of the entire block can be done (4 different ways, the same prediction for both Cb and Cr).

H.264, cont.

Intra prediction modes for 4×4 luma blocks.



Intra prediction modes for chroma blocks and 16×16 luma blocks.



H.264, cont.

There are two different source coding methods to use in H.264.

VLC Quantized and runlength encoded transform components are coded using tabulated codes (CAVLC). Other data (motion vectors, header data, et c.) are coded using fixed length codes or Exp-Golomb codes.

CABAC Context Adaptive Binary Arithmetic Coding. All data is coded using conditioning (contexts) and all probability models are adapted continuously.

Profiles and levels

H.264 has a number of profiles and levels. Similar to all MPEG standards, a profile determines what types of algorithms that can be used and the level sets limits on numerical parameters (like resolution, framerate or data rate). Some examples of profiles:

- BP Basic Profile. Only I and P slices, no B slices. Only 4×4 transforms. Only VLC as source coding method. Only progressive coding (frames).
- MP Main Profile. Also allows B slices, interlace (fields) and CABAC.
- HiP High Profile. Also allows 8×8 transforms.

(There are also other smaller differences between the profiles, but the listed differences are the most important).

High Profile is used in DVB and on Blu-ray discs.

Complexity

Since there are so many ways of coding each macroblock, a H.264 coder is typically much slower than coders for the simpler MPEG standards.

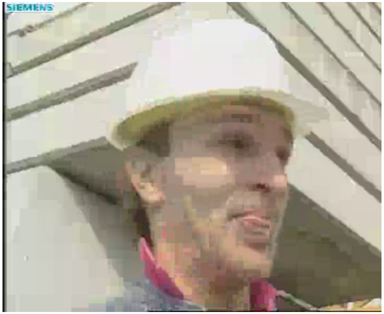
For example, an I block can be predicted in 592 different ways ($16 \cdot 9 + 4$ ways to predict the luma, 4 ways to predict the chromas, $(16 \cdot 9 + 4) \cdot 4 = 592$).

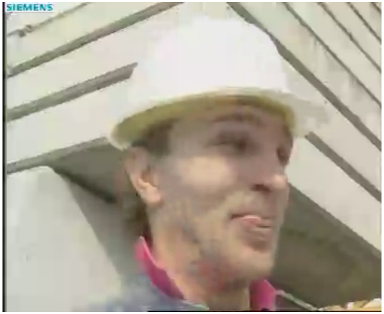
Similarly, for each P or B macroblock we can choose between many different block sizes for motion compensation and several reference frames.

In order to do fast encoding, we can not try all coding options to find the best one. The coder must try to quickly reject prediction modes that will probably not give a good result. We will lose some in coding performance, but the coder will be faster.

Deblocking

Especially when coding at low rates we get many block artifacts from the transform coding. These artifacts, apart from looking bad, will make the motion compensation not work well. To cure this problem, lowpass filtering on the block edges is done in H.264. Resultat with and without filtering below.





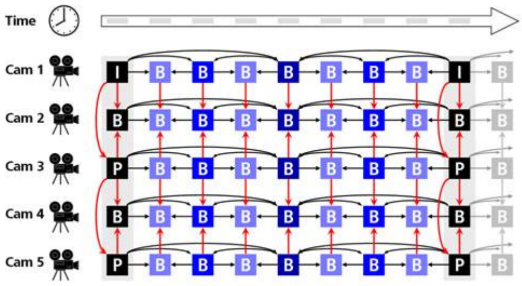
Multiview, 3D

Lately it has become popular to have several cameras that film the same scene from different angles (or, in the case of computer generated material, the video is rendered from different angles). This can be used for 3D video or multiview video, where the viewer can choose between several viewing angles.

In the same way that consecutive frames in a video sequence are very similar, images from cameras close to each other will be very similar. A coding method for multiview or 3D can thus do predictive coding between cameras and not just in the time domain.

3D/Multiview coding

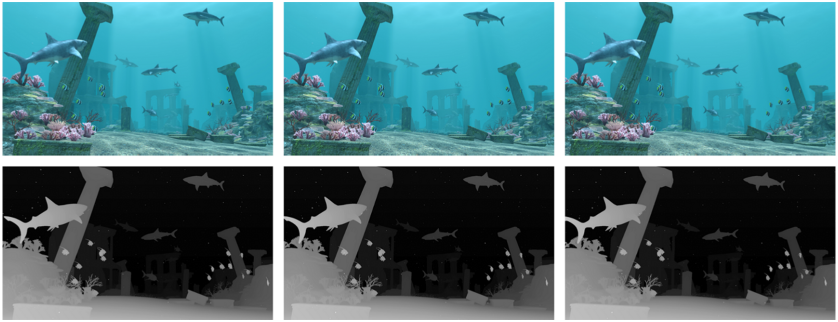
Prediction both in time and between cameras.



This type of 3D coding is referred to as 2D plus delta.

2D plus depth

An alternative way of coding 3D video is to use 2D plus depth. This uses one stream of video data and one stream of depth images. The depth images are used to generate two different viewpoints (one for each eye).



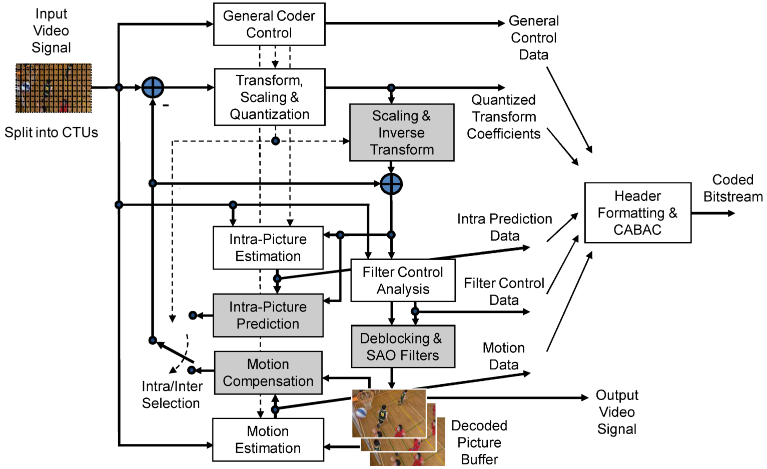
High Efficiency Video Coding

The focus in the work with HEVC has been on developing a coder for high resolution video. Displays with resolutions 4K UHD (3840×2160) and 8K UHD (7680×4320) are available on the consumer market and more and more video material at these resolutions is being produced.

Another goal is to make sure that the dedoder can utilize parallel hardware architectures.

The work on HEVC started in January 2010 and the first version of the standard was adopted in January 2013.

HEVC encoder structure

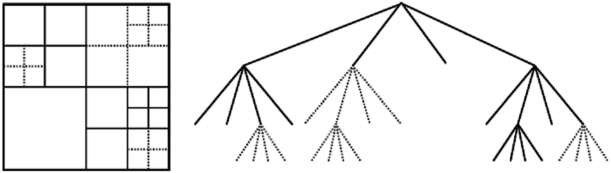


Block structure

The core coding unit is the *Coding Tree Unit (CTU)*, consisting of a square block of pixels of size 64×64, 32×32 or 16×16. This corresponds to the *macroblocks* used in earlier standards.

The colour format is 4:2:0.

The CTU is partitioned (quadtree structure) into a number of *Coding Units (CU)*. The smallest allowed size of a CU is 8×8.



Frame structures

Each frame to be coded can be split into *slices* and *tiles*.

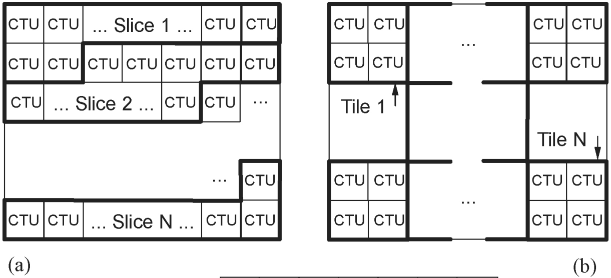
A slice consist of a number of CTU:s in raster scan order that can be correctly decoded without the use of data from other slices. This means that prediction is not performed across slice boundaries.

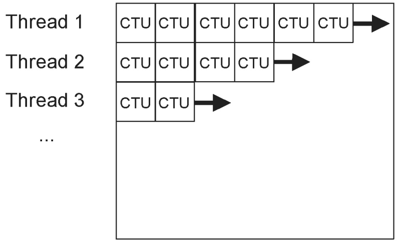
A tile is a rectangular area of the frame that can be correctly decoded without the use of data from other tiles.

A slice can contain multiple tiles. A tile can contain multiple slices.

Slices and tiles can be processed in parallel.

Slices and tiles





(c)

Slice types

Each slice is coded as an I slice, a P slice or a B slice.

- I All CU:s are coded using intra prediction.
- P CU:s are coded either using intra prediction or inter prediction from an earlier decoded picture (one motion vector).
- B CU:s are coded either using intra prediction or inter prediction from one earlier decoded picture and/or one later decoded picture (one or two motion vectors).

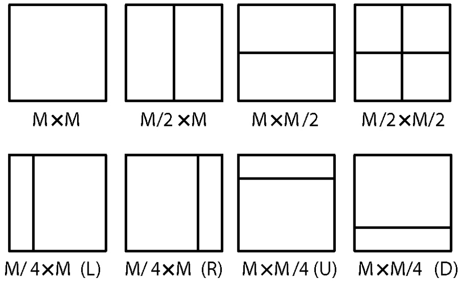
Prediction

The decision to code a picture area using intra or inter prediction is made at the CU level. The CU is partitioned into *Prediction Units (PU)*. The standard supports PU sizes of 64×64 down to 4×4.

For intra prediction the PU size is the same as the CU size for all sizes except the smallest allowed CU size. For this case, it is allowed to split the CU into four PU:s.

For inter prediction, the CU can be split into one, two or four PU:s. A split into four PU:s is only allowed when the CU size is the minimum allowed size.

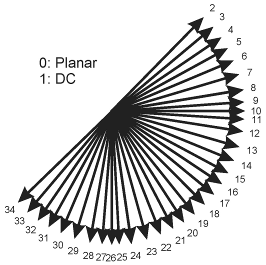
PU sizes

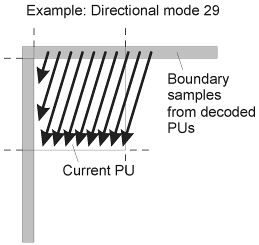


Possible ways to split a CU into PU:s. For intra prediction, only $M \times M$ and $M/2 \times M/2$ can be used. For inter prediction, the lower four partitions are only allowed when $M \geq 16$.

Intra prediction

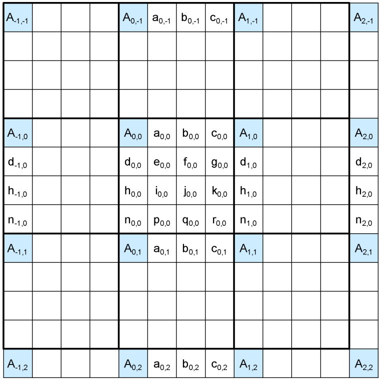
The intra prediction uses previously decoded boundary samples from neighbouring blocks to form the prediction signal. Interpolation along 33 different directions can be used. In addition, planar and DC prediction is possible. There is thus 35 ways to predict each block.




Example: Directional mode 29
Boundary samples from decoded PUs
Current PU

Inter prediction

Each inter PU can have one or two motion vectors and reference picture indices. The motion vectors uses quarter pixel accuracy. Sub-pixel values are interpolated using separable 8-tap filters for half-pixel positions and then separable 7-tap filters for quarter pixel positions.



$A_{-1,-1}$ $A_{0,-1}$ $a_{0,-1}$ $b_{0,-1}$ $c_{0,-1}$ $A_{1,-1}$ $A_{2,-1}$
$A_{-1,0}$ $A_{0,0}$ $a_{0,0}$ $b_{0,0}$ $c_{0,0}$ $A_{1,0}$ $A_{2,0}$
$d_{-1,0}$ $d_{0,0}$ $e_{0,0}$ $f_{0,0}$ $g_{0,0}$ $d_{1,0}$ $d_{2,0}$
$h_{-1,0}$ $h_{0,0}$ $i_{0,0}$ $j_{0,0}$ $k_{0,0}$ $h_{1,0}$ $h_{2,0}$
$n_{-1,0}$ $n_{0,0}$ $p_{0,0}$ $q_{0,0}$ $r_{0,0}$ $n_{1,0}$ $n_{2,0}$
$A_{-1,1}$ $A_{0,1}$ $a_{0,1}$ $b_{0,1}$ $c_{0,1}$ $A_{1,1}$ $A_{2,1}$
$A_{-1,2}$ $A_{0,2}$ $a_{0,2}$ $b_{0,2}$ $c_{0,2}$ $A_{1,2}$ $A_{2,2}$

Transform

The prediction residual (from intra or inter prediction) for each CU is quadtree partitioned into *Transform Units (TU)*. A TU can have size 32×32, 16×16, 8×8 or 4×4.

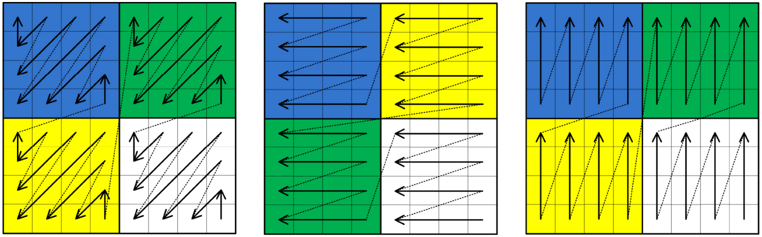
For intra prediction the PU and TU sizes are always the same.

The size of a TU can be larger than the corresponding PU for inter prediction.

The transforms used are integer approximations of the discrete cosine transform (DCT). For intra predicted blocks of size 4×4 an integer version of the discrete sine transform (DST) is also used.

Scanning order of transform coefficients

The transform coefficients are always scanned in sub-blocks of size 4×4, even when the transform blocks are larger.
There are three methods: diagonal up-right, horizontal and vertical.



Diagonal Horizontal Vertical

Scanning order of transform coefficients

For inter prediction only diagonal up-right scanning is used.

For intra prediction, the method used will depend on the direction of the prediction. The vertical scan is used when the prediction directions is close to horizontal. The horizontal scan is used when the prediction directions is close to vertical. For other directions the diagonal up-right method is used.

Quantization and coding

The quantization used is uniform quantization. The coarseness of the quantization is controlled by a quantization parameter QP that can take values from 0 to 51. An increase of QP by 6 corresponds to a doubling of the stepsize, giving an exponential mapping from QP to stepsize, which in turn gives an approximately linear mapping from QP to the rate. Quantization scaling matrices are also supported (giving different stepsizes for different transform components).

The only entropy coding method supported is CABAC (Context Adaptive Binary Arithmetic Coding). This is the same coding method used in H.264.

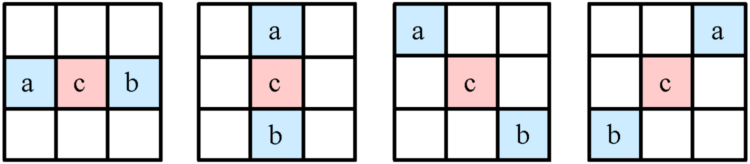
Post processing

After an image is decoded it is filtered to reduce blocking artifacts and other errors inside the blocks.

Deblocking filters are used on the block edges, to reduce the blocking artifacts. This is similar to H.264.

Sample Adaptive Offset (SAO) is a type of non-linear filtering that reduces artifacts in smooth areas (banding) and around edges (ringing). It uses look-up tables of sample offsets that have to be transmitted. A classification of the decoded pixels are made and for each class an offset value is transmitted.

Edge offset classification

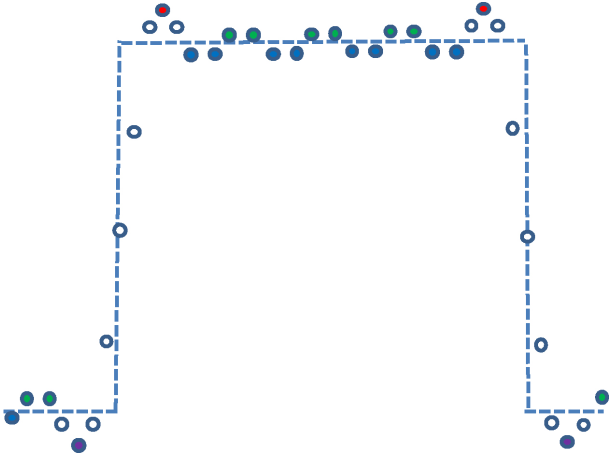


Classification of decoded pixels into 4 categories.

category	condition
1	$c < a \;\&\&\; c < b$
2	$c < a \;\&\&\; c == b \;\|\; c == a \;\&\&\; c < b$
3	$c > a \;\&\&\; c == b \;\|\; c == a \;\&\&\; c > b$
4	$c > a \;\&\&\; c > b$
0	None of the above

The extra information for a block is the direction used for classification and an offset value for categories 1-4. The decoder can make the same classification and add the offset values to the pixels.

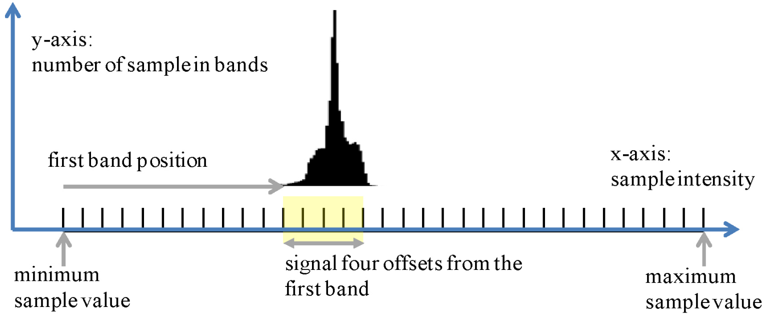
Edge offset classification



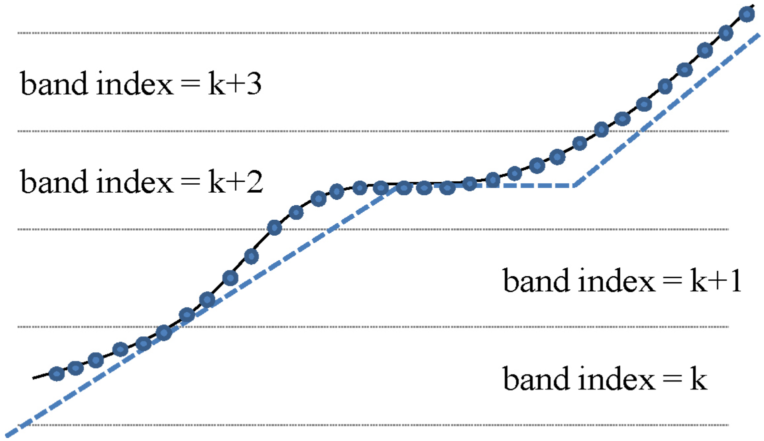
Category 1 (purple), category 2 (blue), category 3 (green), category 4 (red) and category 0 (white).

Block offset classification

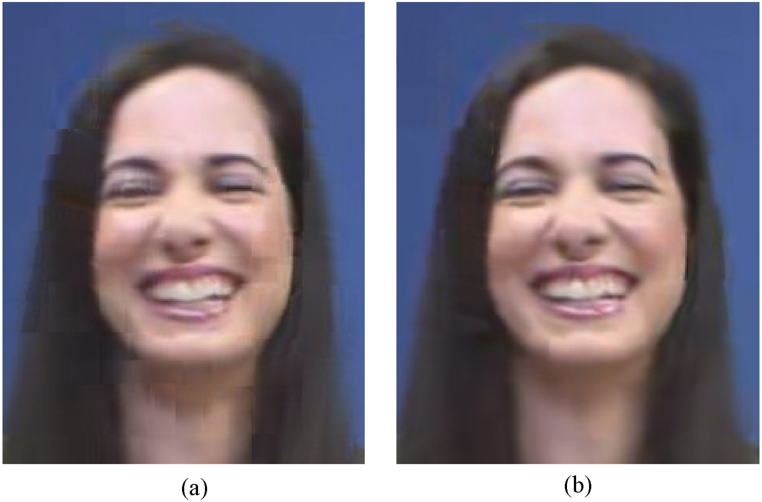
Classification of decoded pixel values into 32 evenly spaced bands (ie a histogram). Only four consecutive bands are used. The extra information is the position of the first band and an offset value for each of the four bands.
The decoder can make the same classification and then add offsets to all pixels in the four bands.



Band offset classification

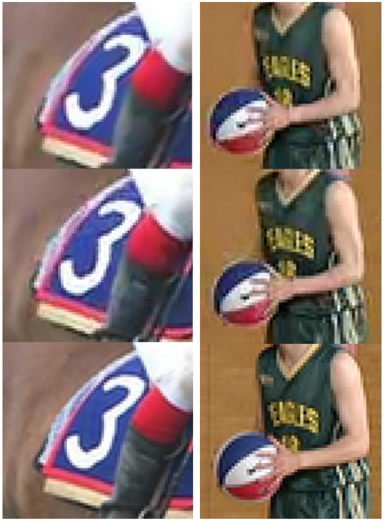


Deblocking



Decoded image without (a) and with (b) deblocking filtering.

SAO


(a) (b)

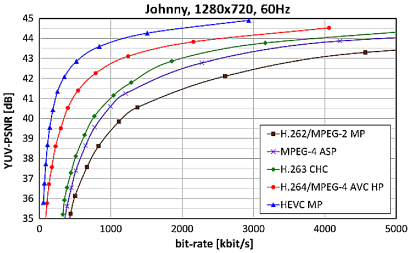
Top to bottom: With SAO, without SAO, original.

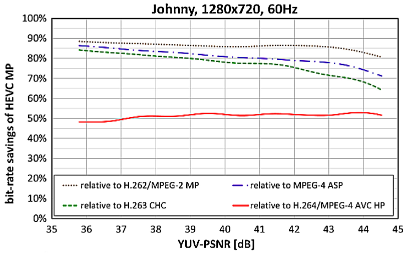
Special coding modes

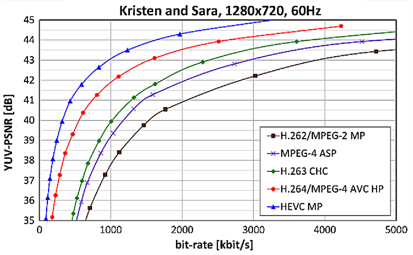
There are three special coding modes that can be used on the CU or TU level.

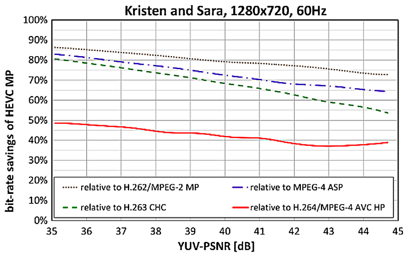
- I_PCM mode: Prediction, transform, quantization and entropy coding are bypassed and the samples are directly represented by a pre-defined number of bits.
- Lossless mode: Transform, quantization and other processing that affect the decoded picture (deblocking filtering, SAO) are bypassed. The residual from the prediction (intra or inter) is fed directly into the entropy coder.
- Transform skip mode: The transform is bypassed. This might give a better result for computer generated images or graphics mixed with the video (eg Screen Content Coding). This can only be used for TB:s of size 4×4.

Coding comparison

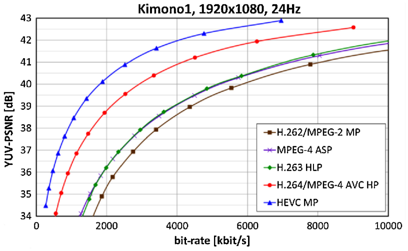


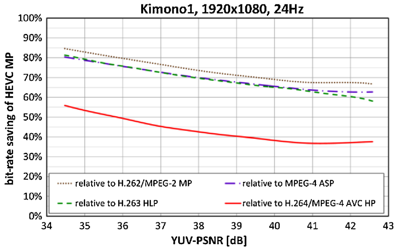


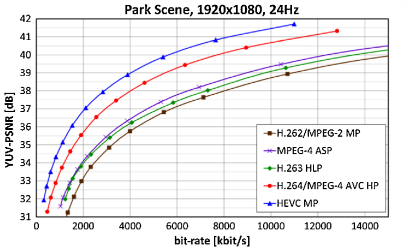


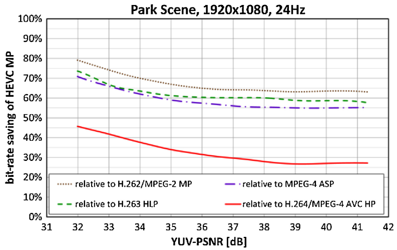


Coding comparison









Extensions of HEVC

Several extensions of HEVC standard have been made, for instance:

- Scalable coding (support for decoding just a part of the bit stream and get an image of lower quality)
- 3D/stereo/multi-view
- Extended range formats (increased bit depth. 4:2:2, 4:4:4 chroma subsampling)
- Support for *Screen Content Coding*, ie source data that contains graphical and text parts.