VP9 Bitstream - superframe and uncompressed header

DRAFT

Rev 1.0, 2015.12.08

1. Introduction

This document is a description of part of the VP9 bitstream format. It covers the high-level format of frames, superframes, as well as the full format of the frame uncompressed header and superframe index.

1.1 Frames, Tiles and Blocks

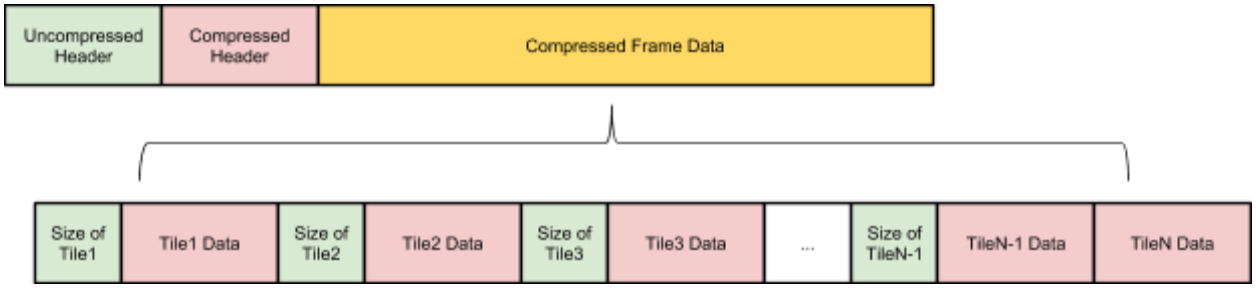
VP9 is based on decomposition of video image frames into rectangular pixel blocks of different sizes, prediction of such blocks using previously reconstructed blocks, and transform coding of the residual signal.

There are only two frame types in VP9. Intra frames or key frames are decoded without reference to any other frame in a sequence. Inter frames are encoded with reference to previously encoded frames or reference buffers.

Video image blocks are grouped in tiles. Based on the layout of the tiles within image frames, tiles can be categorized into column tiles and row tiles. Column tiles are partitioned from image frames vertically into columns and row tiles are partitioned from image frames horizontally into rows. Column tiles are independently coded and decodable units of the video frame. Each column tile can be decoded completely independently. Row tiles are inter-dependent. There has to be at least one tile per frame. Tiles are then broken into 64x64 super blocks that are coded in raster order within the video frame.

1.2 Overview of Compressed Frame Format

Every frame begins with an uncompressed header, followed by a compressed header. Beside the headers, the main body of a compressed video frame contains the compressed per-block data for one or more tiles.

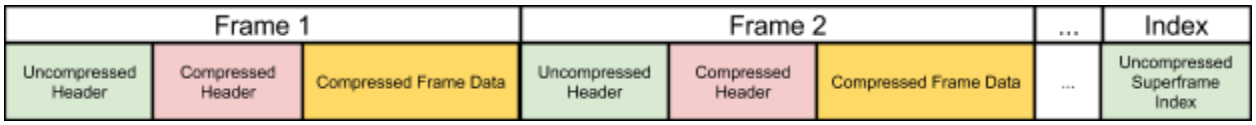


Uncompressed header contains bitstream profile, frame type (intra or inter), colorspace descriptor, YUV chroma subsampling, YUV range, frame size, motion compensation interpolation filter type, frame buffers to be refreshed, loop filter parameters, quantization

parameters, segmentation and tiling information. There is also frame context information and binary flags indicating the use of error resilient mode, parallel decoding mode, high precision motion vector mode, and intra only mode. See section 4 for details.

Compressed header contains frame transform mode, probabilities to decode transform size for each block within the frame, probabilities to decode transform coefficients, probabilities to decode modes and motion vectors, and so on.

VP9 supports consolidating multiple compressed video frames into one single chunk, called “superframe”. The superframe index is stored in the last up to 34 bytes of a chunk. The enclosed frames can be located by parsing superframe index. See section 3 for details.



2. Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses. Note that the methods for identifying and handling errors and other such situations are not specified in this document.

The functions presented here are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

- **read_bits(n)** reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) is specified to return a value equal to 0 and to not advance the bitstream pointer.

The following descriptors specify the parsing process of each syntax element:

- **f(n)**: fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read_bits(n).

- **u(n)**: unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with most significant bit written first.

- **s(n)**: signed integer using n bits for the value and 1 bit for signness. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified below:

s(n) {	Descriptor
value	u(n)
return **sign** ? -value : value	u(1)
}	

- **u_bytes(n)**: unsigned integer using n bytes with least significant byte first. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified below:

u_bytes(n) {	Descriptor
value = 0	
for (i = 0; i < n; ++i)	
value |= **byte** << (i * 8)	u(8)
return value	
}	

3. Superframe

Superframe is a way that vp9 uses to consolidate multiple compressed video frames into one single chunk. The superframe index is stored in the last up to 34 bytes of a chunk. To determine whether a chunk of data has a superframe index, check the last byte in the chunk for the marker 3 bits (0b110xxxxx):

If the marker bits are set apply a mask to the byte:
mask 2 bits (0bxxx11xxx) and add 1 to get the size in bytes for each frame size (1-4)
mask 3 bits (0bxxxxx111) add add 1 to get the # of frame sizes encoded in the index (1-8)

Calculate the total size of the index as follows:
2 + number of frame sizes encoded * number of bytes to hold each frame size

2 represents the one byte holding a marker byte that's duplicated at the start of the index and at

the end of the index.

Go to the first byte of the superframe index, and check that it matches the last byte of the superframe index.

No valid frame can have 0b110xxxxx in the last byte. In the encoder we insure this by adding a byte 0 to the data in this case.

The actual frames are concatenated one frame after each other consecutively. To find the second piece of compressed data in a chunk, read the index to get the frame size, and offset from the start of data to that byte using the size pulled from the data.

3.1 Syntax

superframe_index() {	Descriptor
SUPERFRAME_MARKER /* equal to 0b110 */	f(3)
frame_size_length_minus_one	u(2)
num_frames_minus_one	u(3)
for (i = 0; i <= num_frames_minus_one; ++i)	
frame_sizes[i]	u_bytes(v)
SUPERFRAME_MARKER /* equal to 0b110 */	f(3)
frame_size_length_minus_one	u(2)
num_frames_minus_one	u(3)
}	

3.2 Semantics

frame_size_length_minus_one plus one specifies the size in bytes for each frame.

num_frames_minus_one plus one specifies number of frames contained in this superframe.

frame_sizes[i] specifies the size of ith frame. Every frame_sizes[i] has frame_size_length_minus_one + 1 bytes.

4. Uncompressed Header

Uncompressed header contains bitstream profile, frame type (intra or inter), colorspace descriptor, YUV chroma subsampling, YUV range, frame size, motion compensation interpolation filter type, frame buffers to be refreshed, loop filter parameters, quantization parameters, segmentation and tiling information. There is also frame context information and binary flags indicating the use of error resilient mode, parallel decoding mode, high precision motion vector mode, and intra only mode.

4.1 Syntax

uncompressed_header() {	Descriptor
FRAME_MARKER /* equal to 0b10 */	f(2)
version	u(1)
high	u(1)
profile = (high << 1) + version	
if (profile == 3)	
RESERVED_ZERO /* equal to 0 */	f(1)
show_existing_frame	u(1)
if (show_existing_frame) {	
index_of_frame_to_show	u(3)
return	
}	
frame_type	u(1)
show_frame	u(1)
error_resilient_mode	u(1)
if (frame_type == KEY_FRAME /* 0 */) {	
SYNC_CODE /* equal to 0x498342 */	u(24)
bitdepth_colorspace_sampling(profile)	
refresh_frame_flags = 0xFF	

frame_size()	
} else {	
if (!show_frame)	
intra_only	u(1)
else	
intra_only = false	
if (!error_resilent_mode)	
reset_frame_context	u(2)
else	
reset_frame_context = 0	
if (intra_only) {	
SYNC_CODE /* equal to 0x498342 */	u(24)
if (profile > 0)	
bitdepth_colorspace_sampling(profile)	
refresh_frame_flags	u(REF_FRAMES = 8)
frame_size()	
} else {	
refresh_frame_flags	u(REF_FRAMES = 8)
for (i = 0; i < REFS_PER_FRAME; ++i) {	
frame_refs[i]	u(3)
ref_frame_sign_biases[i]	u(1)
}	
frame_size_from_refs()	
high_precision_mv	u(1)
interp_filter()	
}	
}	

if (!error_resilient_mode) {	
refresh_frame_context	u(1)
frame_parallel_decoding_mode	u(1)
} else {	
refresh_frame_context = 0	
frame_parallel_decoding_mode = 1	
}	
frame_context_index	u(2)
loop_filter()	
quantization()	
segmentation()	
tiles(width)	
compressed_header_size_in_bytes	u(16)
}	

4.2 Semantics

profile VP9 supports four profiles:

Profile	Bit Depth	SRGB Colorspace Support	Chroma Subsampling
0	8	No	YUV 4:2:0
1	8	Yes	YUV 4:2:2, YUV 4:4:0 or YUV 4:4:4[1]
2	10 or 12	No	YUV 4:2:0
3	10 or 12	Yes	YUV 4:2:2, YUV 4:4:0 or YUV 4:4:4[1]

[1]If colorspace is SRGB, then Chroma subsampling is YUV 4:4:4.

show_existing_frame if set to 1, indicates the frame indexed by index_of_frame_to_show is to be displayed, otherwise proceeds for further processing.

index_of_frame_to_show indicates the frame to be displayed. It is only available if show_existing_frame is 1.

frame_type indicates whether the frame type is KEY_FRAME (0) or INTER_FRAME (1).

show_frame indicates whether the frame is displayable (1), or not (0).

error_resilient_mode indicates whether error resilient mode is enabled (1) or disabled (0).

intra_only is only available for INTER_FRAME, and only available when show_frame = 0. It indicates that there are only intra blocks, i.e. no inter blocks in this frame.

reset_frame_context indicates whether the frame context should be reset to default values.

- 0 or 1 implies don't reset.
- 2 resets just the context specified in the frame header.
- 3 reset all contexts.

refresh_frame_flags indicates which reference frame slots will be updated with currently encoded frame, e.g. 0b00100010 indicates that slot_1 and slot_5 will reference current frame. See section 4.4 for details on frame buffer.

frame_refs[] indicates reference frame indexes for LAST_FRAME, GOLDEN_FRAME and ALTREF_FRAME. See section 4.4 for details on frame buffer.

ref_frame_sign_biases[] specify the intended direction of the motion vector (in time), backward (0) or forward (1).

high_precision_mv indicates whether high precision motion vector mode is enabled (1) or disabled (0).

refresh_frame_context indicates whether to refresh frame context.

frame_parallel_decoding_mode indicates whether parallel decoding mode is enabled (1) or disabled (0).

frame_context_index indicates the frame context to use.

compressed_header_size_in_bytes indicates the size of compressed header in bytes.

4.3 Bit depth, Color space and chroma subsampling

4.3.1 Syntax

bitdepth_colorspace_sampling(profile) {	
if (profile >= 2) {	
bit_depth = **bit_depth_flag** ? 12 : 10;	u(1)
} else {	
bit_depth = 8	
}	
colorspace	u(3)
if (colorspace != SRGB) {	
yuv_range_flag	u(1)
if (profile == 1 || profile == 3) {	
subsampling_x	u(1)
subsampling_y	u(1)
RESERVED_ZERO /* equal to 0 */	u(1)
} else {	
subsampling_x = subsampling_y = 1	
}	
} else {	
REQUIRE profile be 1 or 3	
subsampling_x = subsampling_y = 0	
RESERVED_ZERO /* equal to 0 */	u(1)
}	

4.3.2 Semantics

bit_depth_flag indicates bit depth in profile 2 and 3, 1 for 12 bits / pixel and 0 for 10 bits / pixel. In profile 0 or 1, it is always 8 bits / pixel.

colorspace is an integer that specifies the color space of the stream, enumerated in the following table:

COLOR SPACE	Value	Profile
Unknown	0	ALL
BT.601	1	ALL
BT.709	2	ALL
SMPTE-170	3	ALL
SMPTE-240	4	ALL
BT.2020[1]	5	ALL
Reserved	6	
sRGB (RGB)	7	1,3

[1]Note that VP9 passes the color space information in the bitstream including BT.2020, however, VP9 does not specify if it is in the form of “constant luminance” or “non-constant luminance”. As such, application should rely on the signaling outside of the VP9 bitstream. If there is no such signaling, the application may assume non-constant luminance for BT.2020.

yuv_range_flag indicates the black level and range of the luma and chroma signals.

YUV RANGE	Value	Details
Studio Swing	0	Y [16,235], UV [16,240][1]
Full Swing	1	YUV [0, 255]

[1]Note that VP9 encoder/decoder does not enforce the range of YUV values even when the YUV range is signalled as Studio Swing, rather it clamps all values to be between [0, 255], however, the application shall perform correct clamping and color conversion operations according to the specified range.

subsampling_x, subsampling_y indicates chroma subsampling format.

format	x	y
YUV 4:2:0[1]	1	1
YUV 4:2:2	1	0
YUV 4:4:0	0	1

```
YUV 4:4:4 | 0          | 0
----------+-----------+--------
```

[1]In Chroma subsampling format 4:2:0 and 4:2:2, VP9 assumes that chrome samples are collocated with luma samples if there is no explicit signaling outside of the VP9 bitstream. When there is explicit signaling at the container level, the signaled information override VP9's default assumption.

4.4 Frame Buffers

```
----------------------+----------------------
Constant              | Value
======================+======================
REF_FRAMES            | 8
REFS_PER_FRAME        | 3
----------------------+----------------------
```

There are REF_FRAMES slots for reference frames (ref_slots[0..7]). Each reference slot contains one of the previously decoded frames. Each key frame resets all reference slots with itself.

refresh_frame_flags from uncompressed header indicates which reference frame slots will be updated with the currently encoded frame, e.g. 0b00100010 indicates that slot_1 and slot_5 will reference the current frame.

```
function update_ref_slots(frame, refresh_frame_flags) {
  for (i = 0; i < REF_FRAMES; ++i)
    if (refresh_frame_flags & (1 << i))
      ref_slots[i] = frame
}
```

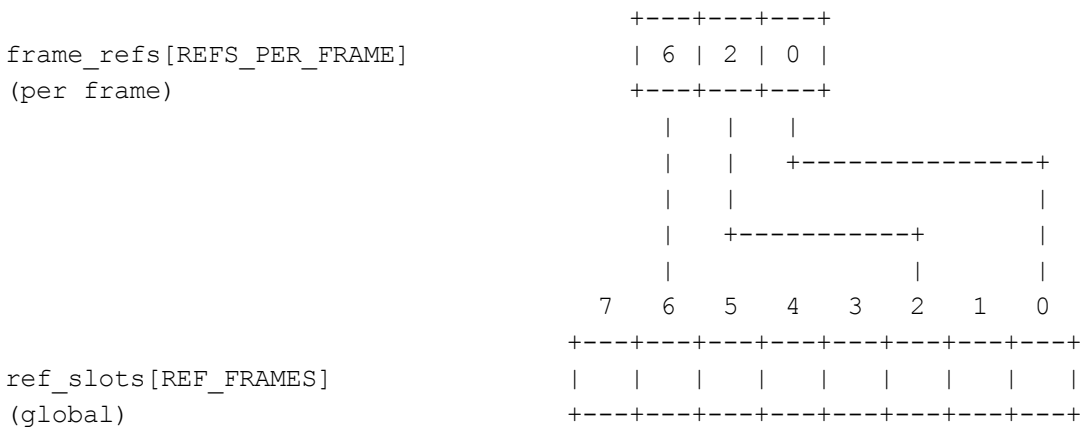
Also from the uncompressed header

```
-------------+------+---------------------------------
frame_refs[0] | u(3) | Reference index for LAST_FRAME
frame_refs[1] | u(3) | Reference index for GOLDEN_FRAME
frame_refs[2] | u(3) | Reference index for ALTREF_FRAME
-------------+------+---------------------------------
```

Each inter frame can reference REFS_PER_FRAME slots with previously decoded frames.

Each ref points to the slot with a frame. Each slot contains a frame (REF_FRAMES)

```
                                    2   1   0
```




```
function get_ref_frame(ref_index) {
  return ref_slots[frame_refs[ref_index]]
}
```

4.5 Frame Size

4.5.1 Syntax

frame_size() {	
width_minus_one	u(16)
height_minus_one	u(16)
has_scaling	u(1)
if (has_scaling) {	
render_width_minus_one	u(16)
render_height_minus_one	u(16)
}	
}	

frame_size_from_refs() {	
for (i = 0; i < REFS_PER_FRAME; ++i) {	
size_in_refs	u(1)
if (size_in_refs) {	

buf = get_ref_frame(i)	
frame.width = buf.width	
frame.height = buf.height	
break	
}	
}	
if (!size_in_refs) {	
width_minus_one	u(16)
height_minus_one	u(16)
}	
has_scaling	u(1)
if (has_scaling) {	
display_width_minus_one	u(16)
display_height_minus_one	u(16)
}	
}	

4.5.2 Semantics

width_minus_one plus one is frame width in pixels.

height_minus_one plus one is frame height in pixels.

has_scaling indicates whether render width and height are the same as frame width and height (0) or not (1).

render_width_minus_one plus one is render frame width in pixels. It is only available if has_scaling is 1.

render_height_minus_one plus one is render frame height in pixels. It is only available if has_scaling is 1.

4.6 Interpolation Filters

4.6.1 Syntax

interp_filter() {	
filter_switchable	u(1)
if (!filter_switchable)	
filter_index	u(2)
}	

4.6.2 Semantics

filter_switchable, if set to 1, indicates that the filter selection is signaled at the block level. Otherwise filter_index signals the filter selection for all blocks in the frame.

filter_index indicates the filter for the current frame. See the table below for the corresponding filter.

INTERP_FILTER	filter_index
EIGHTTAP_SMOOTH	00
EIGHTTAP	01
EIGHTTAP_SHARP	10
BILINEAR	11

4.7 Loop Filter

The baseline frame-level parameters controlling the loop filter are defined in the frame header along with a mechanism for adjustment based on a block's prediction mode and/or reference frame. loop_filter_level and sharpness_level are two controlling factors in applying the loop filtering operations. However, these values may also be overridden on a per-block basis using segmentation.

Besides the loop_filter_level and sharpness_level, there are a number of thresholds and limits

used in the above filters. These values may or may not be changed based on the loop_filter_level and / or sharpness_level strength value coded in the bitstream. In addition, the values can be adjusted for any block based on prediction mode, reference frame. They can also be adjusted by the segment based adjustments.

Constant	Value	Description
MAX_REF_LF_DELTAS	4	Number of loop filter deltas per frame type
MAX_MODE_LF_DELTAS	2	Number of loop filter deltas per prediction mode

4.7.1 Syntax

loop_filter() {	Descriptor
filter_level	u(6)
sharpness_level	u(3)
mode_ref_delta_enabled	u(1)
if (mode_ref_delta_enabled) {	
mode_ref_delta_update	u(1)
if (mode_ref_delta_update) {	
for(i = 0; i < MAX_REF_LF_DELTAS; ++i) {	
if (**update_ref_delta**)	u(1)
ref_deltas[i]	s(6)
}	
for(i = 0; i < MAX_MODE_LF_DELTAS; ++i) {	
if (**update_mode_delta**)	u(1)
mode_deltas[i]	s(6)
}	
}	
}	
}	

4.7.2 Semantics

filter_level indicates loop filter strength, can be adjusted for the segment.

sharpness_level indicates sharpness level.

mode_ref_delta_enabled indicates whether mode delta and ref delta are enabled.

mode_ref_delta_update indicates whether mode delta and ref delta values should be updated.

ref_deltas[i] is loop filter strength adjustment based on frame type (INTRA_FRAME, LAST_FRAME, GOLDEN_FRAME, ALTREF_FRAME).

mode_deltas[i] is loop filter strength adjustments based on mode (zero, new mv).

4.8 Quantization

All residue signals are specified via a quantized transform applied to the Y, U or V blocks. Before inverting the transform, each decoded coefficient is multiplied by one of four quants, the choice of which depends on the plane (Y, U or V) and coefficient position (DC/AC coefficient). Quants are determined using lookup tables ac_qlookup[] and dc_qlookup[] (There is one set of values per bit depth. See Appendix A for the definitions). Tables are the same for all planes. Base frame level lookup table index or qindex (AC for Y) and its three adjustments (DC for Y, AC for U or V, DC for U or V) are obtained from the uncompressed frame header. Base qindex can be overridden using segmentation feature.

4.8.1 Syntax

quantization() {	Descriptor
base_qindex	u(QINDEX_BITS = 8)
if (**y_dc_delta_q_set**)	u(1)
y_dc_delta_q	s(4)
else	
y_dc_delta_q = 0	
y_ac_delta = 0	

if (**uv_dc_delta_q_set**)	u(1)
uv_dc_delta_q	s(4)
else	
uv_dc_delta_q = 0	
if (**uv_ac_delta_q_set**)	u(1)
uv_ac_delta_q	s(4)
else	
uv_ac_delta_q = 0	
}	

4.8.2 Semantics

base_qindex indicates the base frame qindex, i.e. index in quant lookup table. Value range 0..255.

yc_dc_delta_q indicates explicit qindex adjustment for y dc-coefficient. Value range -15..15.

uv_dc_delta_q indicates the qindex adjustment for the uv dc-coefficient. Value range -15..15.

uv_ac_delta_q indicates the qindex adjustment for the uv ac-coefficient. Value range -15..15.

4.9 Segmentation

VP9 provides a means of segmenting the image and then applying various signals or adjustments at the segment level. Generally speaking, the segmentation mechanism provides a flexible set of tools that can be used, in an application specific way, to target improvements in perceptual quality for a given compression ratio. Segmentation can be particularly efficient and useful when the segmentation of a video sequence persists or changes little, for many frames.

Up to 8 segments may be specified for any given frame. For each of these segments it is possible to specify:

- A quantizer (Q) -- absolute value or delta.
- A loop filter strength -- absolute value or delta.
- A prediction reference frame.

- A block skip mode that implies both the use of a (0,0) motion vector and that no residual will be coded.

Each of these data values for each segment may be individually updated at the frame level. Where a value is not updated in a given frame, the value from the previous frame persists. The exceptions to this are key frames, intra only frames or other frames where independence from past frame values is required (for example to enable error resilience). In such cases all values are reset to a default.

It is possible to indicate segment affiliation for any prediction block of size 8x8 pixels or greater. Updates to this “segment map” are explicitly coded using either a temporal coding or direct coding strategy (chosen at the frame level).

If no explicit update is coded for a block’s segment affiliation, then it persists from frame to frame with the same provisos detailed above for the segment data values in regard to key frames, intra only frames and frames where independence from past frames is required. In such cases the segment affiliation for each block defaults to 0 unless explicitly updated.

Internally, segment affiliation is stored at the resolution of 8x8 blocks (a segment map). This can lead to conflicts when, for example, a transform size of 32x32 is selected for a 64x64 region. If the different component 8x8 blocks that comprise a larger region have different segment affiliations, then the segment affiliation for the larger region is defined as being the lowest segment i.d. of any of the contributing 8x8 regions. The appropriate, Q delta, loop filter delta, reference frame and skip mode features from this “lowest” segment are then applied to the whole region. However the underlying segment affiliation values at the 8x8 level are not altered.

Where an explicit update to segment affiliation is specified for a prediction unit that is larger than 8x8, the appropriate entries in the segment map, for all the 8x8 blocks that lie within the prediction unit, are reset to the new value.

Every block may optionally override some of the default behaviors of the decoder. Specifically, VP9 uses segment-based adjustments to support changing quantizer level and loop filter level for a block. When the segment-based adjustment feature is enabled for a frame, each block within the frame is coded with a segment_id. This effectively segments all the macroblocks in the current frame into a number of different segments. Blocks within the same segment behave exactly the same for quantizer and loop filter level adjustments.

Constant	Value
MAX_SEGMENTS	8
SEG_TREE_PROBS	MAX_SEGMENTS - 1
PREDICTION_PROBS	3
SEG_LVL_MAX	4

------------------+--------------------

4.9.1 Syntax

segmentation() {	Descriptor
enabled	u(1)
if (!enabled) return	
update_map	u(1)
if (update_map) {	
for (i = 0; i < SEG_TREE_PROBS; ++i) {	
if (**tree_probs_set**)	u(1)
tree_probs[i]	u(8)
else	
tree_probs[i] = MAX_PROB	
}	
for (i = 0; i < PREDICTION_PROBS; ++i)	
pred_probs[i] = MAX_PROB	
temporal_update	u(1)
if (temporal_update) {	
for (i = 0; i < PREDICTION_PROBS; ++i) {	
if (**pred_probs_set**)	u(1)
pred_probs[i]	u(8)
}	
}	
update_data	u(1)
if (udpate_data) {	
abs_delta	u(1)
for (i = 0; i < MAX_SEGMENTS; ++i) {	
for (j = 0; j < SEG_LVL_MAX; ++j) {	

feature_enabled[i][j]	u(1)
if (feature_enabled[i][j]) {	
if (seg_feature_data_signed[j])	
feature_data[i][j]	s(v)
else	
feature_data[i][j]	u(v)
}	
}	
}	
}	
}	

4.9.2 Semantics

enabled indicates whether segmentation is enabled (1) or disabled (0).

update_map indicates whether the segmentation map should be updated (1) or not (0).

tree_probs[] segment id for each block is coded using a tree, where each id is coded using a series of 0 and 1 indicating the path from root to the leaf node. tree_probs has the probabilities of left child vs right child at each node in the tree.

temporal_update is a binary flag to indicate if we use temporal prediction to code segment id, i.e. a value of 1 indicates that the segment id of the block from last frame is used as a prediction for the same block of this frame.

pred_probs[] describes how likely the predicted id value is being used in this frame.

update_data indicates whether segmentation data should be updated (1) or not (0).

abs_delta indicates interpretation of “data” values, whether it is delta (0) or absolute (1).

feature_enabled[i][j] indicates whether the feature data for ith segment and jth type is enabled.

There are four types of feature:

Index	Description
0	Use alternate Quantizer
1	Use alternate loop filter value
2	Optional Segment reference frame
3	Optional segment (0, 0) + skip mode

feature_data[i][j] feature data for ith segment and jth type. It has seg_feature_data_bits[j] for the data and it may contain an additional bit for the sign, whether the sign bit is present is determined by seg_feature_data_signed[j].

```
seg_feature_data_bits = { 8, 6, 2, 0 }
seg_feature_data_signed = { 1, 1, 0, 0 }
```

For example, with j = 1, the field is in the form of s(6) and has 6 + 1 bits.

4.10 Tiles

Video image blocks are grouped in tiles. Based on the layout of the tiles within image frames, tiles can be categorized into column tiles and row tiles. Column tiles are partitioned from image frames vertically into columns and row tiles are partitioned from image frames horizontally into rows.

Column tiles are independently coded and decodable subunits of the video frame. When enabled a frame can be split into, for example, 2 or 4 column-based tiles. Each tile shares the same frame entropy model, but all contexts and pixel values (for intra prediction) that cross tile-boundaries take the same value as those at the left, top or right edge of the frame. Each tile can thus be decoded and encoded completely independently, which is expected to enable significant speedups in multi-threaded encoders / decoders, without introducing any additional latency. Loop-filtering across tile-edges may still be applied, assuming a decoder implementation model where the loop-filtering operation lags the decoder's reconstruction of the individual tiles within the frame so as not to use any pixel that is not already reconstructed. Further, backward entropy adaptation - a lightweight operation - can still be conducted for the whole frame after entropy decoding for all tiles has finished.

Row tiles are inter-dependent.

MIN_TILE_WIDTH_B64	4	Min number of super blocks per tile
MAX_TILE_WIDTH_B64	64	Max number of super blocks per tile

```
function num_bits(n) {
  bits = 0
  while (n > 0) {
    n >>= 1
    ++bits
  }
  return bits
}
```

4.10.1 Syntax

tiles(width) {	Descriptor
superblocks = (width + 63) / 64	
min_log2_tile_cols = num_bits((superblocks - 1) / MAX_TILE_WIDTH_B64)	
max_log2_tile_cols = num_bits(superblocks / MIN_TILE_WIDTH_B64 / 2)	
max_ones = max_log2_tile_cols - min_log2_tile_cols	
log2_tile_cols = min_log2_tile_cols	
while (max_ones--) {	
one	u(1)
if (!one) break	
++log2_tile_cols	
}	
tile_cols = 1 << log2_tile_cols	
log2_tile_rows	u(1)
if (log2_tile_rows)	
log2_tile_rows += **extra_bit**	u(1)
tile_rows = 1 << log2_tile_rows	
}	

4.10.2 Semantics

tile_cols is number of column tiles.

tile_rows is number of row tiles.

Appendix A - dc_qlookup and ac_qlookup tables

```
QINDEX_RANGE = 256

dc_qlookup_8[QINDEX_RANGE] = {
  4,       8,    8,    9,   10,   11,   12,   12,
  13,     14,   15,   16,   17,   18,   19,   19,
  20,     21,   22,   23,   24,   25,   26,   26,
  27,     28,   29,   30,   31,   32,   32,   33,
  34,     35,   36,   37,   38,   38,   39,   40,
  41,     42,   43,   43,   44,   45,   46,   47,
  48,     48,   49,   50,   51,   52,   53,   53,
  54,     55,   56,   57,   57,   58,   59,   60,
  61,     62,   62,   63,   64,   65,   66,   66,
  67,     68,   69,   70,   70,   71,   72,   73,
  74,     74,   75,   76,   77,   78,   78,   79,
  80,     81,   81,   82,   83,   84,   85,   85,
  87,     88,   90,   92,   93,   95,   96,   98,
  99,    101,  102,  104,  105,  107,  108,  110,
  111,   113,  114,  116,  117,  118,  120,  121,
  123,   125,  127,  129,  131,  134,  136,  138,
  140,   142,  144,  146,  148,  150,  152,  154,
  156,   158,  161,  164,  166,  169,  172,  174,
  177,   180,  182,  185,  187,  190,  192,  195,
  199,   202,  205,  208,  211,  214,  217,  220,
  223,   226,  230,  233,  237,  240,  243,  247,
  250,   253,  257,  261,  265,  269,  272,  276,
  280,   284,  288,  292,  296,  300,  304,  309,
  313,   317,  322,  326,  330,  335,  340,  344,
  349,   354,  359,  364,  369,  374,  379,  384,
  389,   395,  400,  406,  411,  417,  423,  429,
  435,   441,  447,  454,  461,  467,  475,  482,
  489,   497,  505,  513,  522,  530,  539,  549,
  559,   569,  579,  590,  602,  614,  626,  640,
  654,   668,  684,  700,  717,  736,  755,  775,
  796,   819,  843,  869,  896,  925,  955,  988,
  1022, 1058, 1098, 1139, 1184, 1232, 1282, 1336,
}

dc_qlookup_10[QINDEX_RANGE] = {
  4,      9,    10,    13,    15,    17,    20,    22,
  25,    28,    31,    34,    37,    40,    43,    47,
  50,    53,    57,    60,    64,    68,    71,    75,
  78,    82,    86,    90,    93,    97,   101,   105,
  109,   113,   116,   120,   124,   128,   132,   136,
  140,   143,   147,   151,   155,   159,   163,   166,
```

```
  170,   174,   178,   182,   185,   189,   193,   197,
  200,   204,   208,   212,   215,   219,   223,   226,
  230,   233,   237,   241,   244,   248,   251,   255,
  259,   262,   266,   269,   273,   276,   280,   283,
  287,   290,   293,   297,   300,   304,   307,   310,
  314,   317,   321,   324,   327,   331,   334,   337,
  343,   350,   356,   362,   369,   375,   381,   387,
  394,   400,   406,   412,   418,   424,   430,   436,
  442,   448,   454,   460,   466,   472,   478,   484,
  490,   499,   507,   516,   525,   533,   542,   550,
  559,   567,   576,   584,   592,   601,   609,   617,
  625,   634,   644,   655,   666,   676,   687,   698,
  708,   718,   729,   739,   749,   759,   770,   782,
  795,   807,   819,   831,   844,   856,   868,   880,
  891,   906,   920,   933,   947,   961,   975,   988,
  1001,  1015,  1030,  1045,  1061,  1076,  1090,  1105,
  1120,  1137,  1153,  1170,  1186,  1202,  1218,  1236,
  1253,  1271,  1288,  1306,  1323,  1342,  1361,  1379,
  1398,  1416,  1436,  1456,  1476,  1496,  1516,  1537,
  1559,  1580,  1601,  1624,  1647,  1670,  1692,  1717,
  1741,  1766,  1791,  1817,  1844,  1871,  1900,  1929,
  1958,  1990,  2021,  2054,  2088,  2123,  2159,  2197,
  2236,  2276,  2319,  2363,  2410,  2458,  2508,  2561,
  2616,  2675,  2737,  2802,  2871,  2944,  3020,  3102,
  3188,  3280,  3375,  3478,  3586,  3702,  3823,  3953,
  4089,  4236,  4394,  4559,  4737,  4929,  5130,  5347,
}

dc_qlookup_12[QINDEX_RANGE] = {
  4,    12,    18,    25,    33,    41,    50,    60,
  70,    80,    91,   103,   115,   127,   140,   153,
  166,   180,   194,   208,   222,   237,   251,   266,
  281,   296,   312,   327,   343,   358,   374,   390,
  405,   421,   437,   453,   469,   484,   500,   516,
  532,   548,   564,   580,   596,   611,   627,   643,
  659,   674,   690,   706,   721,   737,   752,   768,
  783,   798,   814,   829,   844,   859,   874,   889,
  904,   919,   934,   949,   964,   978,   993,  1008,
  1022,  1037,  1051,  1065,  1080,  1094,  1108,  1122,
  1136,  1151,  1165,  1179,  1192,  1206,  1220,  1234,
  1248,  1261,  1275,  1288,  1302,  1315,  1329,  1342,
  1368,  1393,  1419,  1444,  1469,  1494,  1519,  1544,
  1569,  1594,  1618,  1643,  1668,  1692,  1717,  1741,
  1765,  1789,  1814,  1838,  1862,  1885,  1909,  1933,
  1957,  1992,  2027,  2061,  2096,  2130,  2165,  2199,
  2233,  2267,  2300,  2334,  2367,  2400,  2434,  2467,
  2499,  2532,  2575,  2618,  2661,  2704,  2746,  2788,
  2830,  2872,  2913,  2954,  2995,  3036,  3076,  3127,
```

```
  3177,  3226,  3275,  3324,  3373,  3421,  3469,  3517,
  3565,  3621,  3677,  3733,  3788,  3843,  3897,  3951,
  4005,  4058,  4119,  4181,  4241,  4301,  4361,  4420,
  4479,  4546,  4612,  4677,  4742,  4807,  4871,  4942,
  5013,  5083,  5153,  5222,  5291,  5367,  5442,  5517,
  5591,  5665,  5745,  5825,  5905,  5984,  6063,  6149,
  6234,  6319,  6404,  6495,  6587,  6678,  6769,  6867,
  6966,  7064,  7163,  7269,  7376,  7483,  7599,  7715,
  7832,  7958,  8085,  8214,  8352,  8492,  8635,  8788,
  8945,  9104,  9275,  9450,  9639,  9832, 10031, 10245,
  10465, 10702, 10946, 11210, 11482, 11776, 12081, 12409,
  12750, 13118, 13501, 13913, 14343, 14807, 15290, 15812,
  16356, 16943, 17575, 18237, 18949, 19718, 20521, 21387,
}

ac_qlookup_8[QINDEX_RANGE] = {
  4,       8,    9,   10,   11,   12,   13,   14,
  15,     16,   17,   18,   19,   20,   21,   22,
  23,     24,   25,   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,   52,   53,   54,
  55,     56,   57,   58,   59,   60,   61,   62,
  63,     64,   65,   66,   67,   68,   69,   70,
  71,     72,   73,   74,   75,   76,   77,   78,
  79,     80,   81,   82,   83,   84,   85,   86,
  87,     88,   89,   90,   91,   92,   93,   94,
  95,     96,   97,   98,   99,  100,  101,  102,
  104,   106,  108,  110,  112,  114,  116,  118,
  120,   122,  124,  126,  128,  130,  132,  134,
  136,   138,  140,  142,  144,  146,  148,  150,
  152,   155,  158,  161,  164,  167,  170,  173,
  176,   179,  182,  185,  188,  191,  194,  197,
  200,   203,  207,  211,  215,  219,  223,  227,
  231,   235,  239,  243,  247,  251,  255,  260,
  265,   270,  275,  280,  285,  290,  295,  300,
  305,   311,  317,  323,  329,  335,  341,  347,
  353,   359,  366,  373,  380,  387,  394,  401,
  408,   416,  424,  432,  440,  448,  456,  465,
  474,   483,  492,  501,  510,  520,  530,  540,
  550,   560,  571,  582,  593,  604,  615,  627,
  639,   651,  663,  676,  689,  702,  715,  729,
  743,   757,  771,  786,  801,  816,  832,  848,
  864,   881,  898,  915,  933,  951,  969,  988,
  1007, 1026, 1046, 1066, 1087, 1108, 1129, 1151,
  1173, 1196, 1219, 1243, 1267, 1292, 1317, 1343,
  1369, 1396, 1423, 1451, 1479, 1508, 1537, 1567,
  1597, 1628, 1660, 1692, 1725, 1759, 1793, 1828,
```

```
}

ac_qlookup_10[QINDEX_RANGE] = {
  4,     9,    11,    13,    16,    18,    21,    24,
  27,    30,    33,    37,    40,    44,    48,    51,
  55,    59,    63,    67,    71,    75,    79,    83,
  88,    92,    96,   100,   105,   109,   114,   118,
  122,   127,   131,   136,   140,   145,   149,   154,
  158,   163,   168,   172,   177,   181,   186,   190,
  195,   199,   204,   208,   213,   217,   222,   226,
  231,   235,   240,   244,   249,   253,   258,   262,
  267,   271,   275,   280,   284,   289,   293,   297,
  302,   306,   311,   315,   319,   324,   328,   332,
  337,   341,   345,   349,   354,   358,   362,   367,
  371,   375,   379,   384,   388,   392,   396,   401,
  409,   417,   425,   433,   441,   449,   458,   466,
  474,   482,   490,   498,   506,   514,   523,   531,
  539,   547,   555,   563,   571,   579,   588,   596,
  604,   616,   628,   640,   652,   664,   676,   688,
  700,   713,   725,   737,   749,   761,   773,   785,
  797,   809,   825,   841,   857,   873,   889,   905,
  922,   938,   954,   970,   986,  1002,  1018,  1038,
  1058,  1078,  1098,  1118,  1138,  1158,  1178,  1198,
  1218,  1242,  1266,  1290,  1314,  1338,  1362,  1386,
  1411,  1435,  1463,  1491,  1519,  1547,  1575,  1603,
  1631,  1663,  1695,  1727,  1759,  1791,  1823,  1859,
  1895,  1931,  1967,  2003,  2039,  2079,  2119,  2159,
  2199,  2239,  2283,  2327,  2371,  2415,  2459,  2507,
  2555,  2603,  2651,  2703,  2755,  2807,  2859,  2915,
  2971,  3027,  3083,  3143,  3203,  3263,  3327,  3391,
  3455,  3523,  3591,  3659,  3731,  3803,  3876,  3952,
  4028,  4104,  4184,  4264,  4348,  4432,  4516,  4604,
  4692,  4784,  4876,  4972,  5068,  5168,  5268,  5372,
  5476,  5584,  5692,  5804,  5916,  6032,  6148,  6268,
  6388,  6512,  6640,  6768,  6900,  7036,  7172,  7312,
}

ac_qlookup_12[QINDEX_RANGE] = {
  4,    13,    19,    27,    35,    44,    54,    64,
  75,    87,    99,   112,   126,   139,   154,   168,
  183,   199,   214,   230,   247,   263,   280,   297,
  314,   331,   349,   366,   384,   402,   420,   438,
  456,   475,   493,   511,   530,   548,   567,   586,
  604,   623,   642,   660,   679,   698,   716,   735,
  753,   772,   791,   809,   828,   846,   865,   884,
  902,   920,   939,   957,   976,   994,  1012,  1030,
  1049,  1067,  1085,  1103,  1121,  1139,  1157,  1175,
  1193,  1211,  1229,  1246,  1264,  1282,  1299,  1317,
```

```
  1335,  1352,  1370,  1387,  1405,  1422,  1440,  1457,
  1474,  1491,  1509,  1526,  1543,  1560,  1577,  1595,
  1627,  1660,  1693,  1725,  1758,  1791,  1824,  1856,
  1889,  1922,  1954,  1987,  2020,  2052,  2085,  2118,
  2150,  2183,  2216,  2248,  2281,  2313,  2346,  2378,
  2411,  2459,  2508,  2556,  2605,  2653,  2701,  2750,
  2798,  2847,  2895,  2943,  2992,  3040,  3088,  3137,
  3185,  3234,  3298,  3362,  3426,  3491,  3555,  3619,
  3684,  3748,  3812,  3876,  3941,  4005,  4069,  4149,
  4230,  4310,  4390,  4470,  4550,  4631,  4711,  4791,
  4871,  4967,  5064,  5160,  5256,  5352,  5448,  5544,
  5641,  5737,  5849,  5961,  6073,  6185,  6297,  6410,
  6522,  6650,  6778,  6906,  7034,  7162,  7290,  7435,
  7579,  7723,  7867,  8011,  8155,  8315,  8475,  8635,
  8795,  8956,  9132,  9308,  9484,  9660,  9836, 10028,
  10220, 10412, 10604, 10812, 11020, 11228, 11437, 11661,
  11885, 12109, 12333, 12573, 12813, 13053, 13309, 13565,
  13821, 14093, 14365, 14637, 14925, 15213, 15502, 15806,
  16110, 16414, 16734, 17054, 17390, 17726, 18062, 18414,
  18766, 19134, 19502, 19886, 20270, 20670, 21070, 21486,
  21902, 22334, 22766, 23214, 23662, 24126, 24590, 25070,
  25551, 26047, 26559, 27071, 27599, 28143, 28687, 29247,
}
```