



# AV1 Bitstream & Decoding Process Specification

*Last modified: 2018-11-12 13:09 PT*

## Authors

Peter de Rivaz, Argon Design Ltd  
Jack Haughton, Argon Design Ltd

## Codec Working Group Chair

Adrian Grange, Google Inc

## Design

Lou Quillio, Google Inc

## Version 1.0.0

This version 1.0.0 of the AV1 Bitstream Specification corresponds to the Git tag [v1.0.0](#) in the [AOMediaCodec/av1-spec](#) project. Its content has been validated as consistent with the reference decoder provided by [libaom v1.0.0](#).

To comment on this document, use the [mailing list](#) or the [issue tracker](#).

Copyright 2018, The Alliance for Open Media

Licensing information is available at <http://aomedia.org/license/>

The MATERIALS ARE PROVIDED “AS IS.” The Alliance for Open Media, its members, and its contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to the materials. The entire risk as to implementing or otherwise using the materials is assumed by the implementer and user. IN NO EVENT WILL THE ALLIANCE FOR OPEN MEDIA, ITS MEMBERS, OR CONTRIBUTORS BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS DELIVERABLE OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER MEMBER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Abstract

This document defines the bitstream formats and decoding process for the [Alliance for Open Media](#) AV1 video codec.

## Contents

1. [Scope](#)
2. [Terms and definitions](#)
3. [Symbols and abbreviated terms](#)
4. [Conventions](#)
  - 4.1. [General](#)
  - 4.2. [Arithmetic operators](#)
  - 4.3. [Logical operators](#)
  - 4.4. [Relational operators](#)
  - 4.5. [Bitwise operators](#)
  - 4.6. [Assignment](#)
  - 4.7. [Mathematical functions](#)
  - 4.8. [Method of describing bitstream syntax](#)
  - 4.9. [Functions](#)
  - 4.10. [Descriptors](#)
    - 4.10.1. [General](#)
    - 4.10.2. [f\(n\)](#)
    - 4.10.3. [uvlc\(\)](#)
    - 4.10.4. [le\(n\)](#)
    - 4.10.5. [leb128\(\)](#)
    - 4.10.6. [su\(n\)](#)
    - 4.10.7. [ns\(n\)](#)
    - 4.10.8. [L\(n\)](#)
    - 4.10.9. [S\(\)](#)
    - 4.10.10. [NS\(n\)](#)
5. [Syntax structures](#)
  - 5.1. [General](#)
  - 5.2. [Low overhead bitstream format](#)
  - 5.3. [OBU syntax](#)
    - 5.3.1. [General OBU syntax](#)
    - 5.3.2. [OBU header syntax](#)
    - 5.3.3. [OBU extension header syntax](#)
    - 5.3.4. [Trailing bits syntax](#)
    - 5.3.5. [Byte alignment syntax](#)
  - 5.4. [Reserved OBU syntax](#)
  - 5.5. [Sequence header OBU syntax](#)
    - 5.5.1. [General sequence header OBU syntax](#)
    - 5.5.2. [Color config syntax](#)
    - 5.5.3. [Timing info syntax](#)
    - 5.5.4. [Decoder model info syntax](#)

- 5.5.5. Operating parameters info syntax
- 5.6. Temporal delimiter obu syntax
- 5.7. Padding OBU syntax
- 5.8. Metadata OBU syntax
  - 5.8.1. General metadata OBU syntax
  - 5.8.2. Metadata ITUT T35 syntax
  - 5.8.3. Metadata high dynamic range content light level syntax
  - 5.8.4. Metadata high dynamic range mastering display color volume syntax
  - 5.8.5. Metadata scalability syntax
  - 5.8.6. Scalability structure syntax
  - 5.8.7. Metadata timecode syntax
- 5.9. Frame header OBU syntax
  - 5.9.1. General frame header OBU syntax
  - 5.9.2. Uncompressed header syntax
  - 5.9.3. Get relative distance function
  - 5.9.4. Reference frame marking function
  - 5.9.5. Frame size syntax
  - 5.9.6. Render size syntax
  - 5.9.7. Frame size with refs syntax
  - 5.9.8. Superres params syntax
  - 5.9.9. Compute image size function
  - 5.9.10. Interpolation filter syntax
  - 5.9.11. Loop filter params syntax
  - 5.9.12. Quantization params syntax
  - 5.9.13. Delta quantizer syntax
  - 5.9.14. Segmentation params syntax
  - 5.9.15. Tile info syntax
  - 5.9.16. Tile size calculation function
  - 5.9.17. Quantizer index delta parameters syntax
  - 5.9.18. Loop filter delta parameters syntax
  - 5.9.19. CDEF params syntax
  - 5.9.20. Loop restoration params syntax
  - 5.9.21. TX mode syntax
  - 5.9.22. Skip mode params syntax
  - 5.9.23. Frame reference mode syntax
  - 5.9.24. Global motion params syntax
  - 5.9.25. Global param syntax
  - 5.9.26. Decode signed subexp with ref syntax
  - 5.9.27. Decode unsigned subexp with ref syntax
  - 5.9.28. Decode subexp syntax
  - 5.9.29. Inverse recenter function
  - 5.9.30. Film grain params syntax
  - 5.9.31. Temporal point info syntax
- 5.10. Frame OBU syntax
- 5.11. Tile group OBU syntax

- 5.11.1. General tile group OBU syntax
- 5.11.2. Decode tile syntax
- 5.11.3. Clear block decoded flags function
- 5.11.4. Decode partition syntax
- 5.11.5. Decode block syntax
- 5.11.6. Mode info syntax
- 5.11.7. Intra frame mode info syntax
- 5.11.8. Intra segment ID syntax
- 5.11.9. Read segment ID syntax
- 5.11.10. Skip mode syntax
- 5.11.11. Skip syntax
- 5.11.12. Quantizer index delta syntax
- 5.11.13. Loop filter delta syntax
- 5.11.14. Segmentation feature active function
- 5.11.15. TX size syntax
- 5.11.16. Block TX size syntax
- 5.11.17. Var TX size syntax
- 5.11.18. Inter frame mode info syntax
- 5.11.19. Inter segment ID syntax
- 5.11.20. Is inter syntax
- 5.11.21. Get segment ID function
- 5.11.22. Intra block mode info syntax
- 5.11.23. Inter block mode info syntax
- 5.11.24. Filter intra mode info syntax
- 5.11.25. Ref frames syntax
- 5.11.26. Assign MV syntax
- 5.11.27. Read motion mode syntax
- 5.11.28. Read inter intra syntax
- 5.11.29. Read compound type syntax
- 5.11.30. Get mode function
- 5.11.31. MV syntax
- 5.11.32. MV component syntax
- 5.11.33. Compute prediction syntax
- 5.11.34. Residual syntax
- 5.11.35. Transform block syntax
- 5.11.36. Transform tree syntax
- 5.11.37. Get TX size function
- 5.11.38. Get plane residual size function
- 5.11.39. Coefficients syntax
- 5.11.40. Compute transform type function
- 5.11.41. Get scan function
- 5.11.42. Intra angle info luma syntax
- 5.11.43. Intra angle info chroma syntax
- 5.11.44. Is directional mode function
- 5.11.45. Read CFL alphas syntax

- 5.11.46. Palette mode info syntax
- 5.11.47. Transform type syntax
- 5.11.48. Get transform set function
- 5.11.49. Palette tokens syntax
- 5.11.50. Palette color context function
- 5.11.51. Is inside function
- 5.11.52. Is inside filter region function
- 5.11.53. Clamp MV row function
- 5.11.54. Clamp MV col function
- 5.11.55. Clear CDEF function
- 5.11.56. Read CDEF syntax
- 5.11.57. Read loop restoration syntax
- 5.11.58. Read loop restoration unit syntax
- 5.12. Tile list OBU syntax
  - 5.12.1. General tile list OBU syntax
  - 5.12.2. Tile list entry syntax
- 6. Syntax structures semantics
  - 6.1. General
  - 6.2. OBU semantics
    - 6.2.1. General OBU semantics
    - 6.2.2. OBU header semantics
    - 6.2.3. OBU extension header semantics
    - 6.2.4. Trailing bits semantics
    - 6.2.5. Byte alignment semantics
  - 6.3. Reserved OBU semantics
  - 6.4. Sequence header OBU semantics
    - 6.4.1. General sequence header OBU semantics
    - 6.4.2. Color config semantics
    - 6.4.3. Timing info semantics
    - 6.4.4. Decoder model info semantics
    - 6.4.5. Operating parameters info semantics
  - 6.5. Temporal delimiter OBU semantics
  - 6.6. Padding OBU semantics
  - 6.7. Metadata OBU semantics
    - 6.7.1. General metadata OBU semantics
    - 6.7.2. Metadata ITUT T35 semantics
    - 6.7.3. Metadata high dynamic range content light level semantics
    - 6.7.4. Metadata high dynamic range mastering display color volume semantics
    - 6.7.5. Metadata scalability semantics
    - 6.7.6. Scalability structure semantics
      - 6.7.6.1. General
      - 6.7.6.2. L1T2 (Informative)
      - 6.7.6.3. L1T3 (Informative)
      - 6.7.6.4. L2T1 / L2T1h (Informative)
      - 6.7.6.5. L2T2 / L2T2h (Informative)

- 6.7.6.6. L2T3 / L2T3h (Informative)
  - 6.7.6.7. S2T1 / S2T1h (Informative)
  - 6.7.6.8. S2T2 / S2T2h (Informative)
  - 6.7.6.9. S2T3 / S2T3h (Informative)
  - 6.7.7. Metadata timecode semantics
- 6.8. Frame header OBU semantics
  - 6.8.1. General frame header OBU semantics
  - 6.8.2. Uncompressed header semantics
  - 6.8.3. Reference frame marking semantics
  - 6.8.4. Frame size semantics
  - 6.8.5. Render size semantics
  - 6.8.6. Frame size with refs semantics
  - 6.8.7. Superres params semantics
  - 6.8.8. Compute image size semantics
  - 6.8.9. Interpolation filter semantics
  - 6.8.10. Loop filter semantics
  - 6.8.11. Quantization params semantics
  - 6.8.12. Delta quantizer semantics
  - 6.8.13. Segmentation params semantics
  - 6.8.14. Tile info semantics
  - 6.8.15. Quantizer index delta parameters semantics
  - 6.8.16. Loop filter delta parameters semantics
  - 6.8.17. Global motion params semantics
  - 6.8.18. Global param semantics
  - 6.8.19. Decode subexp semantics
  - 6.8.20. Film grain params semantics
  - 6.8.21. TX mode semantics
  - 6.8.22. Skip mode params semantics
  - 6.8.23. Frame reference mode semantics
  - 6.8.24. Temporal point info semantics
- 6.9. Frame OBU semantics
- 6.10. Tile group OBU semantics
  - 6.10.1. General tile group OBU semantics
  - 6.10.2. Decode tile semantics
  - 6.10.3. Clear block decoded flags semantics
  - 6.10.4. Decode partition semantics
  - 6.10.5. Decode block semantics
  - 6.10.6. Intra frame mode info semantics
  - 6.10.7. Intra segment ID semantics
  - 6.10.8. Read segment ID semantics
  - 6.10.9. Inter segment ID semantics
  - 6.10.10. Skip mode semantics
  - 6.10.11. Skip semantics
  - 6.10.12. Quantizer index delta semantics
  - 6.10.13. Loop filter delta semantics

- 6.10.14. CDEF params semantics
- 6.10.15. Loop restoration params semantics
- 6.10.16. TX size semantics
- 6.10.17. Block TX size semantics
- 6.10.18. Var TX size semantics
- 6.10.19. Transform type semantics
- 6.10.20. Is inter semantics
- 6.10.21. Intra block mode info semantics
- 6.10.22. Inter block mode info semantics
- 6.10.23. Filter intra mode info semantics
- 6.10.24. Ref frames semantics
- 6.10.25. Assign mv semantics
- 6.10.26. Read motion mode semantics
- 6.10.27. Read inter intra semantics
- 6.10.28. Read compound type semantics
- 6.10.29. MV semantics
- 6.10.30. MV component semantics
- 6.10.31. Compute prediction semantics
- 6.10.32. Residual semantics
- 6.10.33. Transform block semantics
- 6.10.34. Coefficients semantics
- 6.10.35. Intra angle info semantics
- 6.10.36. Read CFL alphas semantics
- 6.10.37. Palette mode info semantics
- 6.10.38. Palette tokens semantics
- 6.10.39. Palette color context semantics
- 6.10.40. Read CDEF semantics
- 6.10.41. Read loop restoration unit semantics
- 6.11. Tile list OBU semantics
  - 6.11.1. General tile list OBU semantics
  - 6.11.2. Tile list entry semantics
- 7. Decoding process
  - 7.1. Overview
  - 7.2. General decoding process
  - 7.3. Large scale tile decoding process
    - 7.3.1. General
    - 7.3.2. Decode camera tile process
  - 7.4. Decode frame wrapup process
  - 7.5. Ordering of OBUs
  - 7.6. Random access decoding
    - 7.6.1. General
    - 7.6.2. Definitions
    - 7.6.3. Conformance requirements
    - 7.6.4. Encoder consequences
    - 7.6.5. Decoder consequences

- 7.7. Frame end update CDF process
- 7.8. Set frame refs process
- 7.9. Motion field estimation process
  - 7.9.1. General
  - 7.9.2. Projection process
  - 7.9.3. Get MV projection process
  - 7.9.4. Get block position process
- 7.10. Motion vector prediction processes
  - 7.10.1. General
  - 7.10.2. Find MV stack process
    - 7.10.2.1. Setup global MV process
    - 7.10.2.2. Scan row process
    - 7.10.2.3. Scan col process
    - 7.10.2.4. Scan point process
    - 7.10.2.5. Temporal scan process
    - 7.10.2.6. Temporal sample process
    - 7.10.2.7. Add reference motion vector process
    - 7.10.2.8. Search stack process
    - 7.10.2.9. Compound search stack process
    - 7.10.2.10. Lower precision process
    - 7.10.2.11. Sorting process
    - 7.10.2.12. Extra search process
    - 7.10.2.13. Add extra MV candidate process
    - 7.10.2.14. Context and clamping process
  - 7.10.3. Has overlappable candidates process
  - 7.10.4. Find warp samples process
    - 7.10.4.1. General
    - 7.10.4.2. Add sample process
- 7.11. Prediction processes
  - 7.11.1. General
  - 7.11.2. Intra prediction process
    - 7.11.2.1. General
    - 7.11.2.2. Basic intra prediction process
    - 7.11.2.3. Recursive intra prediction process
    - 7.11.2.4. Directional intra prediction process
    - 7.11.2.5. DC intra prediction process
    - 7.11.2.6. Smooth intra prediction process
    - 7.11.2.7. Filter corner process
    - 7.11.2.8. Intra filter type process
    - 7.11.2.9. Intra edge filter strength selection process
    - 7.11.2.10. Intra edge upsample selection process
    - 7.11.2.11. Intra edge upsample process
    - 7.11.2.12. Intra edge filter process
  - 7.11.3. Inter prediction process
    - 7.11.3.1. General



- 7.11.3.2. Rounding variables derivation process
    - 7.11.3.3. Motion vector scaling process
    - 7.11.3.4. Block inter prediction process
    - 7.11.3.5. Block warp process
    - 7.11.3.6. Setup shear process
    - 7.11.3.7. Resolve divisor process
    - 7.11.3.8. Warp estimation process
    - 7.11.3.9. Overlapped motion compensation process
    - 7.11.3.10. Overlap blending process
    - 7.11.3.11. Wedge mask process
    - 7.11.3.12. Difference weight mask process
    - 7.11.3.13. Intra mode variant mask process
    - 7.11.3.14. Mask blend process
    - 7.11.3.15. Distance weights process
  - 7.11.4. Palette prediction process
  - 7.11.5. Predict chroma from luma process
- 7.12. Reconstruction and dequantization
  - 7.12.1. General
  - 7.12.2. Dequantization functions
  - 7.12.3. Reconstruct process
- 7.13. Inverse transform process
  - 7.13.1. General
  - 7.13.2. 1D transforms
    - 7.13.2.1. Butterfly functions
    - 7.13.2.2. Inverse DCT array permutation process
    - 7.13.2.3. Inverse DCT process
    - 7.13.2.4. Inverse ADST input array permutation process
    - 7.13.2.5. Inverse ADST output array permutation process
    - 7.13.2.6. Inverse ADST4 process
    - 7.13.2.7. Inverse ADST8 process
    - 7.13.2.8. Inverse ADST16 process
    - 7.13.2.9. Inverse ADST process
    - 7.13.2.10. Inverse Walsh-Hadamard transform process
    - 7.13.2.11. Inverse identity transform 4 process
    - 7.13.2.12. Inverse identity transform 8 process
    - 7.13.2.13. Inverse identity transform 16 process
    - 7.13.2.14. Inverse identity transform 32 process
    - 7.13.2.15. Inverse identity transform process
  - 7.13.3. 2D inverse transform process
- 7.14. Loop filter process
  - 7.14.1. General
  - 7.14.2. Edge loop filter process
  - 7.14.3. Filter size process
  - 7.14.4. Adaptive filter strength process
  - 7.14.5. Adaptive filter strength selection process

- 7.14.6. Sample filtering process
  - 7.14.6.1. General
  - 7.14.6.2. Filter mask process
  - 7.14.6.3. Narrow filter process
  - 7.14.6.4. Wide filter process
- 7.15. CDEF process
  - 7.15.1. CDEF block process
  - 7.15.2. CDEF direction process
  - 7.15.3. CDEF filter process
- 7.16. Upscaling process
- 7.17. Loop restoration process
  - 7.17.1. Loop restore block process
  - 7.17.2. Self guided filter process
  - 7.17.3. Box filter process
  - 7.17.4. Wiener filter process
  - 7.17.5. Wiener coefficient process
  - 7.17.6. Get source sample process
- 7.18. Output process
  - 7.18.1. General
  - 7.18.2. Film grain synthesis process
    - 7.18.2.1. General
    - 7.18.2.2. Random number process
    - 7.18.2.3. Generate grain process
    - 7.18.2.4. Scaling lookup initialization process
    - 7.18.2.5. Add noise synthesis process
- 7.19. Motion field motion vector storage process
- 7.20. Reference frame update process
- 7.21. Reference frame loading process
- 8. Parsing process
  - 8.1. Parsing process for  $f(n)$
  - 8.2. Parsing process for symbol decoder
    - 8.2.1. General
    - 8.2.2. Initialization process for symbol decoder
    - 8.2.3. Boolean decoding process
    - 8.2.4. Exit process for symbol decoder
    - 8.2.5. Parsing process for read\_literal
    - 8.2.6. Symbol decoding process
  - 8.3. Parsing process for CDF encoded syntax elements
    - 8.3.1. General
    - 8.3.2. Cdf selection process
- 9. Additional tables
  - 9.1. General
  - 9.2. Scan tables
  - 9.3. Conversion tables
  - 9.4. Default CDF tables

- 9.5. Quantizer matrix tables
  - 9.5.1. General
  - 9.5.2. Derivation process (Informative)
  - 9.5.3. Tables
- 10. Annex A: Profiles and levels
  - 10.1. General
  - 10.2. Profiles
  - 10.3. Levels
  - 10.4. Decoder Conformance
- 11. Annex B: Length delimited bitstream format
  - 11.1. Overview
  - 11.2. Length delimited bitstream syntax
  - 11.3. Length delimited bitstream semantics
- 12. Annex C: Error resilience behavior (informative)
  - 12.1. General
  - 12.2. Definition of processable frames
  - 12.3. Recommendation for processable frames
  - 12.4. Encoder consequences of processable frames
  - 12.5. Decoder consequences of processable frames
- 13. Annex D: Large scale tile use case (informative)
- 14. Annex E: Decoder model
  - 14.1. General
  - 14.2. Decoder model definitions
  - 14.3. Operating modes
    - 14.3.1. Resource availability mode
    - 14.3.2. Decoding schedule mode
    - 14.3.3. When timing information is not present in the bitstream
  - 14.4. Frame timing definitions
    - 14.4.1. Start of DFG bits arrival
    - 14.4.2. End of DFG bits arrival
    - 14.4.3. Scheduled removal times
    - 14.4.4. Removal times in decoding schedule mode
    - 14.4.5. Removal times in resource availability mode
    - 14.4.6. Frame decode timing
    - 14.4.7. Frame presentation timing
  - 14.5. Decoder model
    - 14.5.1. Decoder model functions
    - 14.5.2. Decoder model process
  - 14.6. Bitstream conformance
    - 14.6.1. General
    - 14.6.2. Decoder buffer delay consistency across RAP (applies to decoding schedule mode)
    - 14.6.3. Smoothing buffer overflow
    - 14.6.4. Smoothing buffer underflow
    - 14.6.5. Minimum decode time (applies to decoding schedule mode)
    - 14.6.6. Minimum presentation Interval

14.6.7. [Decode deadline](#)

14.6.8. [Level imposed constraints](#)

14.6.9. [Decode Process constraints](#)

15. [Bibliography](#)

# 1. Scope

This document specifies the Alliance for Open Media AV1 bitstream formats and decoding process.

## 2. Terms and definitions

For the purposes of this document, the following terms and definitions apply:

**AC coefficient**

Any transform coefficient whose frequency indices are non-zero in at least one dimension.

**Altref**

(Alternative reference frame) A frame that can be used in inter coding.

**Base layer**

The layer with `spatial_id` and `temporal_id` values equal to 0.

**Bitstream**

The sequence of bits generated by encoding a sequence of frames.

**Bit string**

An ordered string with limited number of bits. The left most bit is the most significant bit (MSB), the right most bit is the least significant bit (LSB).

**Block**

A square or rectangular region of samples.

**Block scan**

A specified serial ordering of quantized coefficients.

**Byte**

An 8-bit bit string.

**Byte alignment**

One bit is byte aligned if the position of the bit is an integer multiple of eight from the position of the first bit in the bitstream.

**CDEF**

Constrained Directional Enhancement Filter designed to adaptively filter blocks based on identifying the direction.

**CDF**

Cumulative distribution function representing the probability times 32768 that a symbol has value less than or equal to a given level.

**Chroma**

A sample value matrix or a single sample value of one of the two color difference signals.

**Note:** Symbols of chroma are U and V.

**Coded frame**

The representation of one frame before the decoding process.

**Component**

One of the three sample value matrices (one luma matrix and two chroma matrices) or its single sample value.

**Compound prediction**

A type of inter prediction where sample values are computed by blending together predictions from two reference frames (the frames blended can be the same or different).

**DC coefficient**

A transform coefficient whose frequency indices are zero in both dimensions.

**Decoded frame**

The frame reconstructed out of the bitstream by the decoder.

**Decoder**

One embodiment of the decoding process.

**Decoding process**

The process that derives decoded frames from syntax elements.

**Dequantization**

The process in which transform coefficients are obtained by scaling the quantized coefficients.

**Encoder**

One embodiment of the encoding process.

**Encoding process**

A process not specified in this Specification that generates the bitstream that conforms to the description provided in this document.

**Enhancement layer**

A layer with either `spatial_id` greater than 0 or `temporal_id` greater than 0.

**Flag**

A binary variable - some variables and syntax elements (e.g. `obu_extension_flag`) are described using the word flag to highlight that the syntax element can only be equal to 0 or equal to 1.

**Frame**

The representation of video signals in the spatial domain, composed of one luma sample matrix (Y) and two chroma sample matrices (U and V).

**Frame context**

A set of probabilities used in the decoding process.

**Golden frame**

A frame that can be used in inter coding. Typically the golden frame is encoded with higher quality and is used as a reference for multiple inter frames.

**Inter coding**

Coding one block or frame using inter prediction.

**Inter frame**

A frame compressed by referencing previously decoded frames and which may use intra prediction or inter prediction.

**Inter prediction**

The process of deriving the prediction value for the current frame using previously decoded frames.

**Intra coding**

Coding one block or frame using intra prediction.

**Intra frame**

A frame compressed using only intra prediction which can be independently decoded.

**Intra prediction**

The process of deriving the prediction value for the current sample using previously decoded sample values in the same decoded frame.

**Inverse transform**

The process in which a transform coefficient matrix is transformed into a spatial sample value matrix.

**Key frame**

An Intra frame which resets the decoding process when it is shown.

**Layer**

A set of tile group OBUs with identical spatial\_id and identical temporal\_id values.

**Level**

A defined set of constraints on the values for the syntax elements and variables.

**Loop filter**

A filtering process applied to the reconstruction intended to reduce the visibility of block edges.

**Luma**

A sample value matrix or a single sample value representing the monochrome signal related to the primary colors.

**Note:** The symbol representing luma is Y.

**Mode info**

Syntax elements sent for a block containing an indication of how a block is to be predicted during the decoding process.

**Mode info block**

A luma sample value block of size 4x4 or larger and its two corresponding chroma sample value blocks (if present).

**Motion vector**

A two-dimensional vector used for inter prediction which refers the current frame to the reference frame, the value of which provides the coordinate offsets from a location in the current frame to a location in the reference frame.

**OBU**

All structures are packetized in “Open Bitstream Units” or OBUs. Each OBU has a header, which provides identifying information for the contained data (payload).

**Parse**

The procedure of getting the syntax element from the bitstream.

**Prediction**

The implementation of the prediction process consisting of either inter or intra prediction.

**Prediction process**

The process of estimating the decoded sample value or data element using a predictor.

**Prediction value**

The value, which is the combination of the previously decoded sample values or data elements, used in the decoding process of the next sample value or data element.

**Profile**

A subset of syntax, semantics and algorithms defined in a part.

**Quantization parameter**

A variable used for scaling the quantized coefficients in the decoding process.

**Quantized coefficient**

A transform coefficient before dequantization.

**Raster scan**

Maps a two dimensional rectangular raster into a one dimensional raster, in which the entry of the one dimensional raster starts from the first row of the two dimensional raster, and the scanning then goes through the second row and the third row, and so on. Each raster row is scanned in left to right order.

**Reconstruction**

Obtaining the addition of the decoded residual and the corresponding prediction values.

**Reference**

One of a set of tags, each of which is mapped to a reference frame.

**Reference frame**

A storage area for a previously decoded frame and associated information.



**Reserved**

A special syntax element value which may be used to extend this part in the future.

**Residual**

The differences between the reconstructed samples and the corresponding prediction values.

**Sample**

The basic elements that compose the frame.

**Sample value**

The value of a sample. This is an integer from 0 to 255 (inclusive) for 8-bit frames, from 0 to 1023 (inclusive) for 10-bit frames, and from 0 to 4095 (inclusive) for 12-bit frames.

**Segmentation map**

A 3-bit number containing the segment affiliation for each 4x4 block in the image. A segmentation map is stored for each reference frame to allow new frames to use a previously coded map.

**Sequence**

The highest level syntax structure of coding bitstream, including one or several consecutive coded frames.

**Superblock**

The top level of the block quadtree within a tile. All superblocks within a frame are the same size and are square. The superblocks may be 128x128 luma samples or 64x64 luma samples. A superblock may contain 1 or 2 or 4 mode info blocks, or may be bisected in each direction to create 4 sub-blocks, which may themselves be further subpartitioned, forming the block quadtree.

**Switch Frame**

An inter frame that can be used as a point to switch between sequences. Switch frames overwrite all the reference frames without forcing the use of intra coding. The intention is to allow a streaming use case where videos can be encoded in small chunks (say of 1 second duration), each starting with a switch frame. If the available bandwidth drops, the server can start sending chunks from a lower bitrate encoding instead. When this happens the inter prediction uses the existing higher quality reference frames to decode the switch frame. This approach allows a bitrate switch without the cost of a full key frame.

**Syntax element**

An element of data represented in the bitstream.

**Temporal delimiter OBU**

An indication that the following OBUs will have a different presentation/decoding time stamp from the one of the last frame prior to the temporal delimiter.

**Temporal unit**

A Temporal unit consists of all the OBUs that are associated with a specific, distinct time instant. It consists of a temporal delimiter OBU, and all the OBUs that follow, up to but not including the next temporal delimiter.

**Temporal group**

A set of frames whose temporal prediction structure is used periodically in a video sequence.

**Tile**

A rectangular region of the frame that can be decoded and encoded independently, although loop-filtering across tile edges is still applied.

**Transform block**

A rectangular transform coefficient matrix, used as input to the inverse transform process.

**Transform coefficient**

A scalar value, considered to be in a frequency domain, contained in a transform block.

**Uncompressed header**

High level description of the frame to be decoded that is encoded without the use of arithmetic encoding.

### 3. Symbols and abbreviated terms

#### DCT

Discrete Cosine Transform

#### ADST

Asymmetric Discrete Sine Transform

#### LSB

Least Significant Bit

#### MSB

Most Significant Bit

#### WHT

Walsh Hadamard Transform

The specification makes use of a number of constant integers. Constants that relate to the semantics of a particular syntax element are defined in [section 6](#).

Additional constants are defined below:

Symbol name	Value	Description
REFS_PER_FRAME	7	Number of reference frames that can be used for inter prediction
TOTAL_REFS_PER_FRAME	8	Number of reference frame types (including intra type)
BLOCK_SIZE_GROUPS	4	Number of contexts when decoding <code>y_mode</code>
BLOCK_SIZES	22	Number of different block sizes used
BLOCK_INVALID	22	Sentinel value to mark partition choices that are not allowed
MAX_SB_SIZE	128	Maximum size of a superblock in luma samples
MI_SIZE	4	Smallest size of a mode info block in luma samples
MI_SIZE_LOG2	2	Base 2 logarithm of smallest size of a mode info block
MAX_TILE_WIDTH	4096	Maximum width of a tile in units of luma samples
MAX_TILE_AREA	$4096 * 2304$	Maximum area of a tile in units of luma samples
MAX_TILE_ROWS	64	Maximum number of tile rows
MAX_TILE_COLS	64	Maximum number of tile columns

Symbol name	Value	Description
INTRABC_DELAY_PIXELS	256	Number of horizontal luma samples before intra block copy can be used
INTRABC_DELAY_SB64	4	Number of 64 by 64 blocks before intra block copy can be used
NUM_REF_FRAMES	8	Number of frames that can be stored for future reference
IS_INTER_CONTEXTS	4	Number of contexts for <code>is_inter</code>
REF_CONTEXTS	3	Number of contexts for <code>single_ref</code> , <code>comp_ref</code> , <code>comp_bwdref</code> , <code>uni_comp_ref</code> , <code>uni_comp_ref_p1</code> and <code>uni_comp_ref_p2</code>
MAX_SEGMENTS	8	Number of segments allowed in segmentation map
SEGMENT_ID_CONTEXTS	3	Number of contexts for <code>segment_id</code>
SEG_LVL_ALT_Q	0	Index for quantizer segment feature
SEG_LVL_ALT_LF_Y_V	1	Index for vertical luma loop filter segment feature
SEG_LVL_REF_FRAME	5	Index for reference frame segment feature
SEG_LVL_SKIP	6	Index for skip segment feature
SEG_LVL_GLOBALMV	7	Index for global mv feature
SEG_LVL_MAX	8	Number of segment features
PLANE_TYPES	2	Number of different plane types (luma or chroma)
TX_SIZE_CONTEXTS	3	Number of contexts for transform size
INTERP_FILTERS	3	Number of values for <code>interp_filter</code>
INTERP_FILTER_CONTEXTS	16	Number of contexts for <code>interp_filter</code>
SKIP_MODE_CONTEXTS	3	Number of contexts for decoding <code>skip_mode</code>
SKIP_CONTEXTS	3	Number of contexts for decoding skip
PARTITION_CONTEXTS	4	Number of contexts when decoding <code>partition</code>
TX_SIZES	5	Number of square transform sizes
TX_SIZES_ALL	19	Number of transform sizes (including non-square sizes)
TX_MODES	3	Number of values for <code>tx_mode</code>
DCT_DCT	0	Inverse transform rows with <code>DCT</code> and columns with <code>DCT</code>

Symbol name	Value	Description
ADST_DCT	1	Inverse transform rows with DCT and columns with ADST
DCT_ADST	2	Inverse transform rows with ADST and columns with DCT
ADST_ADST	3	Inverse transform rows with ADST and columns with ADST
FLIPADST_DCT	4	Inverse transform rows with FLIPADST and columns with DCT
DCT_FLIPADST	5	Inverse transform rows with DCT and columns with FLIPADST
FLIPADST_FLIPADST	6	Inverse transform rows with FLIPADST and columns with FLIPADST
ADST_FLIPADST	7	Inverse transform rows with ADST and columns with FLIPADST
FLIPADST_ADST	8	Inverse transform rows with FLIPADST and columns with ADST
IDTX	9	Inverse transform rows with identity and columns with identity
V_DCT	10	Inverse transform rows with identity and columns with DCT
H_DCT	11	Inverse transform rows with DCT and columns with identity
V_ADST	12	Inverse transform rows with identity and columns with ADST
H_ADST	13	Inverse transform rows with ADST and columns with identity
V_FLIPADST	14	Inverse transform rows with identity and columns with FLIPADST
H_FLIPADST	15	Inverse transform rows with FLIPADST and columns with identity
TX_TYPES	16	Number of inverse transform types
MB_MODE_COUNT	17	Number of values for YMode
INTRA_MODES	13	Number of values for y_mode

Symbol name	Value	Description
UV_INTRA_MODES_CFL_NOT_ALLOWED	13	Number of values for <code>uv_mode</code> when chroma from luma is not allowed
UV_INTRA_MODES_CFL_ALLOWED	14	Number of values for <code>uv_mode</code> when chroma from luma is allowed
COMPOUND_MODES	8	Number of values for <code>compound_mode</code>
COMPOUND_MODE_CONTEXTS	8	Number of contexts for <code>compound_mode</code>
COMP_NEWMV_CTXS	5	Number of new mv values used when constructing context for <code>compound_mode</code>
NEW_MV_CONTEXTS	6	Number of contexts for <code>new_mv</code>
ZERO_MV_CONTEXTS	2	Number of contexts for <code>zero_mv</code>
REF_MV_CONTEXTS	6	Number of contexts for <code>ref_mv</code>
DRL_MODE_CONTEXTS	3	Number of contexts for <code>drl_mode</code>
MV_CONTEXTS	2	Number of contexts for decoding motion vectors including one for intra block copy
MV_INTRABC_CONTEXT	1	Motion vector context used for intra block copy
MV_JOINTS	4	Number of values for <code>mv_joint</code>
MV_CLASSES	11	Number of values for <code>mv_class</code>
CLASS0_SIZE	2	Number of values for <code>mv_class0_bit</code>
MV_OFFSET_BITS	10	Maximum number of bits for decoding motion vectors
MAX_LOOP_FILTER	63	Maximum value used for loop filtering
REF_SCALE_SHIFT	14	Number of bits of precision when scaling reference frames
SUBPEL_BITS	4	Number of bits of precision when choosing an inter prediction filter kernel
SUBPEL_MASK	15	$(1 \ll \text{SUBPEL\_BITS}) - 1$
SCALE_SUBPEL_BITS	10	Number of bits of precision when computing inter prediction locations
MV_BORDER	128	Value used when clipping motion vectors
PALETTE_COLOR_CONTEXTS	5	Number of values for color contexts

Symbol name	Value	Description
PALETTE_MAX_COLOR_CONTEXT_HASH	8	Number of mappings between color context hash and color context
PALETTE_BLOCK_SIZE_CONTEXTS	7	Number of values for palette block size
PALETTE_Y_MODE_CONTEXTS	3	Number of values for palette Y plane mode contexts
PALETTE_UV_MODE_CONTEXTS	2	Number of values for palette U and V plane mode contexts
PALETTE_SIZES	7	Number of values for palette_size
PALETTE_COLORS	8	Number of values for palette_color
PALETTE_NUM_NEIGHBORS	3	Number of neighbors considered within palette computation
DELTA_Q_SMALL	3	Value indicating alternative encoding of quantizer index delta values
DELTA_LF_SMALL	3	Value indicating alternative encoding of loop filter delta values
QM_TOTAL_SIZE	3344	Number of values in the quantizer matrix
MAX_ANGLE_DELTA	3	Maximum magnitude of AngleDeltaY and AngleDeltaUV
DIRECTIONAL_MODES	8	Number of directional intra modes
ANGLE_STEP	3	Number of degrees of step per unit increase in AngleDeltaY or AngleDeltaUV.
TX_SET_TYPES_INTRA	3	Number of intra transform set types
TX_SET_TYPES_INTER	4	Number of inter transform set types
WARPEDMODEL_PREC_BITS	16	Internal precision of warped motion models
IDENTITY	0	Warp model is just an identity transform
TRANSLATION	1	Warp model is a pure translation
ROTZOOM	2	Warp model is a rotation + symmetric zoom + translation
AFFINE	3	Warp model is a general affine transform
GM_ABS_TRANS_BITS	12	Number of bits encoded for translational components of global motion models, if part of a ROTZOOM or AFFINE model

Symbol name	Value	Description
GM_ABS_TRANS_ONLY_BITS	9	Number of bits encoded for translational components of global motion models, if part of a TRANSLATION model
GM_ABS_ALPHA_BITS	12	Number of bits encoded for non-translational components of global motion models
DIV_LUT_PREC_BITS	14	Number of fractional bits of entries in divisor lookup table
DIV_LUT_BITS	8	Number of fractional bits for lookup in divisor lookup table
DIV_LUT_NUM	257	Number of entries in divisor lookup table
MOTION_MODES	3	Number of values for motion modes
SIMPLE	0	Use translation or global motion compensation
OBMC	1	Use overlapped block motion compensation
LOCALWARP	2	Use local warp motion compensation
LEAST_SQUARES_SAMPLES_MAX	8	Largest number of samples used when computing a local warp
LS_MV_MAX	256	Largest motion vector difference to include in local warp computation
WARPEDMODEL_TRANS_CLAMP	$1 \ll 23$	Clamping value used for translation components of warp
WARPEDMODEL_NONDIAGAFFINE_CLAMP	$1 \ll 13$	Clamping value used for matrix components of warp
WARPEDPIXEL_PREC_SHIFTS	$1 \ll 6$	Number of phases used in warped filtering
WARPEDDIFF_PREC_BITS	10	Number of extra bits of precision in warped filtering
GM_ALPHA_PREC_BITS	15	Number of fractional bits for sending non-translational warp model coefficients
GM_TRANS_PREC_BITS	6	Number of fractional bits for sending translational warp model coefficients
GM_TRANS_ONLY_PREC_BITS	3	Number of fractional bits used for pure translational warps
INTERINTRA_MODES	4	Number of inter intra modes
MASK_MASTER_SIZE	64	Size of MasterMask array



Symbol name	Value	Description
SEGMENT_ID_PREDICTED_CONTEXTS	3	Number of contexts for <code>segment_id_predicted</code>
IS_INTER_CONTEXTS	4	Number of contexts for <code>is_inter</code>
SKIP_CONTEXTS	3	Number of contexts for <code>skip</code>
FWD_REFS	4	Number of syntax elements for forward reference frames
BWD_REFS	3	Number of syntax elements for backward reference frames
SINGLE_REFS	7	Number of syntax elements for single reference frames
UNIDIR_COMP_REFS	4	Number of syntax elements for unidirectional compound reference frames
COMPOUND_TYPES	2	Number of values for <code>compound_type</code>
CFL_JOINT_SIGNS	8	Number of values for <code>cfl_alpha_signs</code>
CFL_ALPHABET_SIZE	16	Number of values for <code>cfl_alpha_u</code> and <code>cfl_alpha_v</code>
COMP_INTER_CONTEXTS	5	Number of contexts for <code>comp_mode</code>
COMP_REF_TYPE_CONTEXTS	5	Number of contexts for <code>comp_ref_type</code>
CFL_ALPHA_CONTEXTS	6	Number of contexts for <code>cfl_alpha_u</code> and <code>cfl_alpha_v</code>
INTRA_MODE_CONTEXTS	5	Number of contexts for <code>intra_frame_y_mode</code>
COMP_GROUP_IDX_CONTEXTS	6	Number of contexts for <code>comp_group_idx</code>
COMPOUND_IDX_CONTEXTS	6	Number of contexts for <code>compound_idx</code>
INTRA_EDGE_KERNELS	3	Number of filter kernels for the intra edge filter
INTRA_EDGE_TAPS	5	Number of kernel taps for the intra edge filter
FRAME_LF_COUNT	4	Number of loop filter strength values
MAX_VARTX_DEPTH	2	Maximum depth for variable transform trees
TXFM_PARTITION_CONTEXTS	21	Number of contexts for <code>txfm_split</code>
REF_CAT_LEVEL	640	Bonus weight for close motion vectors
MAX_REF_MV_STACK_SIZE	8	Maximum number of motion vectors in the stack
MFMV_STACK_SIZE	3	Stack size for motion field motion vectors

Symbol name	Value	Description
MAX_TX_DEPTH	2	Number of contexts for tx_depth when the maximum transform size is 8x8
WEDGE_TYPES	16	Number of directions for the wedge mask process
FILTER_BITS	7	Number of bits used in Wiener filter coefficients
WIENER_COEFFS	3	Number of Wiener filter coefficients to read
SGRPROJ_PARAMS_BITS	4	Number of bits needed to specify self guided filter set
SGRPROJ_PRJ_SUBEXP_K	4	Controls how self guided deltas are read
SGRPROJ_PRJ_BITS	7	Precision bits during self guided restoration
SGRPROJ_RST_BITS	4	Restoration precision bits generated higher than source before projection
SGRPROJ_MTABLE_BITS	20	Precision of mtable division table
SGRPROJ_RECIP_BITS	12	Precision of division by n table
SGRPROJ_SGR_BITS	8	Internal precision bits for core selfguided_restoration
EC_PROB_SHIFT	6	Number of bits to reduce CDF precision during arithmetic coding
EC_MIN_PROB	4	Minimum probability assigned to each symbol during arithmetic coding
SELECT_SCREEN_CONTENT_TOOLS	2	Value that indicates the allow_screen_content_tools syntax element is coded
SELECT_INTEGER_MV	2	Value that indicates the force_integer_mv syntax element is coded
RESTORATION_TILESIZE_MAX	256	Maximum size of a loop restoration tile
MAX_FRAME_DISTANCE	31	Maximum distance when computing weighted prediction
MAX_OFFSET_WIDTH	8	Maximum horizontal offset of a projected motion vector
MAX_OFFSET_HEIGHT	0	Maximum vertical offset of a projected motion vector
WARP_PARAM_REDUCE_BITS	6	Rounding bitwidth for the parameters to the shear process

Symbol name	Value	Description
NUM_BASE_LEVELS	2	Number of quantizer base levels
COEFF_BASE_RANGE	12	The quantizer range above NUM_BASE_LEVELS above which the Exp-Golomb coding process is activated
BR_CDF_SIZE	4	Number of contexts for <code>coeff_br</code>
SIG_COEF_CONTEXTS_EOB	4	Number of contexts for <code>coeff_base_eob</code>
SIG_COEF_CONTEXTS_2D	26	Context offset for <code>coeff_base</code> for horizontal-only or vertical-only transforms.
SIG_COEF_CONTEXTS	42	Number of contexts for <code>coeff_base</code>
SIG_REF_DIFF_OFFSET_NUM	5	Maximum number of context samples to be used in determining the context index for <code>coeff_base</code> and <code>coeff_base_eob</code> .
SUPERRES_NUM	8	Numerator for upscaling ratio
SUPERRES_DENOM_MIN	9	Smallest denominator for upscaling ratio
SUPERRES_DENOM_BITS	3	Number of bits sent to specify denominator of upscaling ratio
SUPERRES_FILTER_BITS	6	Number of bits of fractional precision for upscaling filter selection
SUPERRES_FILTER_SHIFTS	$1 \ll \text{SUPERRES\_FILTER\_BITS}$	Number of phases of upscaling filters
SUPERRES_FILTER_TAPS	8	Number of taps of upscaling filters
SUPERRES_FILTER_OFFSET	3	Sample offset for upscaling filters
SUPERRES_SCALE_BITS	14	Number of fractional bits for computing position in upscaling
SUPERRES_SCALE_MASK	$(1 \ll 14) - 1$	Mask for computing position in upscaling
SUPERRES_EXTRA_BITS	8	Difference in precision between SUPERRES_SCALE_BITS and SUPERRES_FILTER_BITS
TXB_SKIP_CONTEXTS	13	Number of contexts for <code>all_zero</code>
EOB_COEF_CONTEXTS	9	Number of contexts for <code>eob_extra</code>
DC_SIGN_CONTEXTS	3	Number of contexts for <code>dc_sign</code>
LEVEL_CONTEXTS	21	Number of contexts for <code>coeff_br</code>

Symbol name	Value	Description
TX_CLASS_2D	0	Transform class for transform types performing non-identity transforms in both directions
TX_CLASS_HORIZ	1	Transform class for transforms performing only a horizontal non-identity transform
TX_CLASS_VERT	2	Transform class for transforms performing only a vertical non-identity transform
REFMVS_LIMIT	$(1 < 12) - 1$	Largest reference MV component that can be saved
INTRA_FILTER_SCALE_BITS	4	Scaling shift for intra filtering process
INTRA_FILTER_MODES	5	Number of types of intra filtering
COEFF_CDF_Q_CTXS	4	Number of selectable context types for the coeff( ) syntax structure
PRIMARY_REF_NONE	7	Value of <code>primary_ref_frame</code> indicating that there is no primary reference frame

## 4. Conventions

### 4.1. General

The mathematical operators and their precedence rules used to describe this Specification are similar to those used in the C programming language. However, the operation of integer division with truncation is specifically defined.

In addition, a length 2 array used to hold a motion vector (indicated by the variable name ending with the letters `Mv` or `Mvs`) can be accessed using either array notation (e.g. `Mv[ 0 ]` and `Mv[ 1 ]`), or by just the name (e.g., `Mv`). The only operations defined when using the name are assignment and equality/inequality testing. Assignment of an array is represented using the notation `A = B` and is specified to mean the same as doing both the individual assignments `A[ 0 ] = B[ 0 ]` and `A[ 1 ] = B[ 1 ]`. Equality testing of 2 motion vectors is represented using the notation `A == B` and is specified to mean the same as `(A[ 0 ] == B[ 0 ] && A[ 1 ] == B[ 1 ])`. Inequality testing is defined as `A != B` and is specified to mean the same as `(A[ 0 ] != B[ 0 ] || A[ 1 ] != B[ 1 ])`.

When a variable is said to be representable by a signed integer with `x` bits, it means that the variable is greater than or equal to `-(1 << (x-1))`, and that the variable is less than or equal to `(1 << (x-1))-1`.

The key words “must”, “must not”, “required”, “shall”, “shall not”, “should”, “should not”, “recommended”, “may”, and “optional” in this document are to be interpreted as described in RFC 2119.

### 4.2. Arithmetic operators

+	Addition
–	Subtraction (as a binary operator) or negation (as a unary prefix operator)
*	Multiplication
/	Integer division with truncation of the result toward zero. For example, <code>7/4</code> and <code>-7/-4</code> are truncated to <code>1</code> and <code>-7/4</code> and <code>7/-4</code> are truncated to <code>-1</code> .
<code>a % b</code>	Remainder from division of <code>a</code> by <code>b</code> . Both <code>a</code> and <code>b</code> are positive integers.
÷	Floating point (arithmetical) division.
<code>ceil(x)</code>	The smallest integer that is greater or equal than <code>x</code> .
<code>floor(x)</code>	The largest integer that is smaller or equal than <code>x</code> .

### 4.3. Logical operators

<code>a &amp;&amp; b</code>	Logical AND operation between <code>a</code> and <code>b</code>
<code>a    b</code>	Logical OR operation between <code>a</code> and <code>b</code>
!	Logical NOT operation.

## 4.4. Relational operators

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

## 4.5. Bitwise operators

&	AND operation
	OR operation
^	XOR operation
~	Negation operation
a >> b	Shift <code>a</code> in 2's complement binary integer representation format to the right by <code>b</code> bit positions. This operator is only used with <code>b</code> being a non-negative integer. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of <code>a</code> prior to the shift operation.
a << b	Shift <code>a</code> in 2's complement binary integer representation format to the left by <code>b</code> bit positions. This operator is only used with <code>b</code> being a non-negative integer. Bits shifted into the LSBs as a result of the left shift have a value equal to <code>0</code> .

## 4.6. Assignment

=	Assignment operator
++	Increment, <code>x++</code> is equivalent to <code>x = x + 1</code> . When this operator is used for an array index, the variable value is obtained before the auto increment operation
--	Decrement, i.e. <code>x--</code> is equivalent to <code>x = x - 1</code> . When this operator is used for an array index, the variable value is obtained before the auto decrement operation
+=	Addition assignment operator, for example <code>x += 3</code> corresponds to <code>x = x + 3</code>
-=	Subtraction assignment operator, for example <code>x -= 3</code> corresponds to <code>x = x - 3</code>

## 4.7. Mathematical functions

The following mathematical functions (Abs, Clip3, Clip1, Min, Max, Round2 and Round2Signed) are defined as follows:

$$\text{Abs}(x) = \begin{cases} x; & x \geq 0 \\ -x; & x < 0 \end{cases}$$

$$\text{Clip1}(x) = \text{Clip3}(0, 2^{\text{BitDepth}} - 1, x)$$

$$\text{Clip3}(x, y, z) = \begin{cases} x; & z < x \\ y; & z > y \\ z; & \text{otherwise} \end{cases}$$

$$\text{Min}(x, y) = \begin{cases} x; & x \leq y \\ y; & x > y \end{cases}$$

$$\text{Max}(x, y) = \begin{cases} x; & x \geq y \\ y; & x < y \end{cases}$$

$$\text{Round2}(x, n) = \left\lfloor \frac{x + (2^{n-1})}{2^n} \right\rfloor$$

$$\text{Round2Signed}(x, n) = \begin{cases} \text{Round2}(x, n); & x \geq 0 \\ -\text{Round2}(-x, n); & x < 0 \end{cases}$$

The definition of Round2 uses standard mathematical power and division operations, not integer operations. An equivalent definition using integer operations is:

```
Round2( x, n ) {
  if ( n == 0 )
    return x
  return ( x + ( 1 << (n - 1) ) ) >> n
}
```

The FloorLog2(x) function is defined to be the floor of the base 2 logarithm of the input x.

The input x will always be an integer, and will always be greater than or equal to 1.

This function extracts the location of the most significant bit in x.

An equivalent definition (using the pseudo-code notation introduced in the following section) is:

```
FloorLog2( x ) {
    s = 0
    while ( x != 0 ) {
        x = x >> 1
        s++
    }
    return s - 1
}
```

The CeilLog2(x) function is defined to be the ceiling of the base 2 logarithm of the input x (when x is 0, it is defined to return 0).

The input x will always be an integer, and will always be greater than or equal to 0.

This function extracts the number of bits needed to code a value in the range 0 to x-1.

An equivalent definition (using the pseudo-code notation introduced in the following section) is:

```
CeilLog2( x ) {
    if ( x < 2 )
        return 0
    i = 1
    p = 2
    while ( p < x ) {
        i++
        p = p << 1
    }
    return i
}
```

## 4.8. Method of describing bitstream syntax

The description style of the syntax is similar to the C programming language. Syntax elements in the bitstream are represented in bold type. Each syntax element is described by its name (using only lower case letters with underscore characters) and a descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e. not bold) type. If the value of a syntax element is being computed (e.g. being written with a default value instead of being coded in the bitstream), it also appears in regular type (e.g. `tile_size_minus_1`).

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax



structure and all depending syntax structures. These variables may be used in the decoding process for later syntax structures. Variables starting with a lower case letter are only used within the process from which they are derived. (Single character variables are allowed.)

Constant values appear in all upper case letters with underscore characters (e.g. MI\_SIZE).

Constant lookup tables appear as words (with the first letter of each word in upper case, and remaining letters in lower case) separated with underscore characters (e.g. Block\_Width[...]).

Hexadecimal notation, indicated by prefixing the hexadecimal number by `0x`, may be used when the number of bits is an integer multiple of 4. For example, `0x1a` represents a bit string `0001 1010`.

Binary notation is indicated by prefixing the binary number by `0b`. For example, `0b00011010` represents a bit string `0001 1010`. Binary numbers may include underscore characters to enhance readability. If present, the underscore characters appear every 4 binary digits starting from the LSB. For example, `0b11010` may also be written as `0b1_1010`.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value not equal to 0.

The following table lists examples of the syntax specification format. When `syntax_element` appears (with bold face font), it specifies that this syntax element is parsed from the bitstream.

	Type
<code>/* A statement can be a syntax element with associated descriptor or can be an expression used to specify its existence, type, and value, as in the following examples */</code>	
<code><b>syntax_element</b></code>	f(1)
<code>/* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement. */</code>	
<code>{</code>	
<code>    statement</code>	
<code>    ...</code>	
<code>}</code>	
<code>/* A “while” structure specifies that the statement is to be evaluated repeatedly while the condition remains true. */</code>	
<code>while ( condition )</code>	
<code>    statement</code>	
<code>/* A “do .. while” structure executes the statement once, and then tests the condition. It repeatedly evaluates the statement while the condition remains true. */</code>	

do	
statement	
while ( condition )	
/* An “if .. else” structure tests the condition first. If it is true, the primary statement is evaluated. Otherwise, the alternative statement is evaluated. If the alternative statement is unnecessary to be evaluated, the “else” and corresponding alternative statement can be omitted. */	
if ( condition )	
primary statement	
else	
alternative statement	
/* A “for” structure evaluates the initial statement at the beginning then tests the condition. If it is true, the primary and subsequent statements are evaluated until the condition becomes false. */	
for ( initial statement; condition; subsequent statement )	
primary statement	
/* The return statement in a syntax structure specifies that the parsing of the syntax structure will be terminated without processing any additional information after this stage. When a value immediately follows a return statement, this value shall also be returned as the output of this syntax structure. */	
return x	

## 4.9. Functions

Bitstream functions used for syntax description are specified in this section.

Other functions are included in the syntax tables. The convention is that a section is called syntax if it causes syntax elements to be read from the bitstream, either directly or indirectly through subprocesses. The remaining sections are called functions.

The specification of these functions makes use of a bitstream position indicator. This bitstream position indicator locates the position of the bit that is going to be read next.

**get\_position( )**: Return the value of the bitstream position indicator.

**init\_symbol( sz )**: Initialize the arithmetic decode process for the Symbol decoder with a size of sz bytes as specified in [section 8.2.2](#).

**exit\_symbol( )**: Exit the arithmetic decode process as described in [section 8.2.4](#) (this includes reading trailing bits).

## 4.10. Descriptors

### 4.10.1. General

The following descriptors specify the parsing of syntax elements. Lower case descriptors specify syntax elements that are represented by an integer number of bits in the bitstream; upper case descriptors specify syntax elements that are represented by arithmetic coding.

### 4.10.2. f(n)

Unsigned n-bit number appearing directly in the bitstream. The bits are read from high to low order. The parsing process specified in [section 8.1](#) is invoked and the syntax element is set equal to the return value.

### 4.10.3. uvlc()

Variable length unsigned n-bit number appearing directly in the bitstream. The parsing process for this descriptor is specified below:

uvlc() {	Type
leadingZeros = 0	
while ( 1 ) {	
done	f(1)
if ( done )	
break	
leadingZeros++	
}	
if ( leadingZeros >= 32 ) {	
return ( 1 << 32 ) - 1	
}	
value	f(leadingZeros)
return value + ( 1 << leadingZeros ) - 1	
}	

### 4.10.4. le(n)

Unsigned little-endian n-byte number appearing directly in the bitstream. The parsing process for this descriptor is specified below:

le(n) {	Type
t = 0	
for ( i = 0; i < n; i++) {	
byte	f(8)
t += ( byte << ( i * 8 ) )	
}	
return t	
}	

**Note:** This syntax element will only be present when the bitstream position is byte aligned.

## 4.10.5. leb128()

Unsigned integer represented by a variable number of little-endian bytes.

**Note:** This syntax element will only be present when the bitstream position is byte aligned.

In this encoding, the most significant bit of each byte is equal to 1 to signal that more bytes should be read, or equal to 0 to signal the end of the encoding.

A variable Leb128Bytes is set equal to the number of bytes read during this process.

The parsing process for this descriptor is specified below:

leb128() {	Type
value = 0	
Leb128Bytes = 0	
for ( i = 0; i < 8; i++ ) {	
leb128_byte	f(8)
value  = ( (leb128_byte & 0x7f) << (i*7) )	
Leb128Bytes += 1	
if ( !(leb128_byte & 0x80) ) {	
break	
}	
return value	
}	

It is a requirement of bitstream conformance that the value returned from the leb128 parsing process is less than or equal to  $(1 \ll 32) - 1$ .

**leb128\_byte** contains 8 bits read from the bitstream. The bottom 7 bits are used to compute the variable value. The most significant bit is used to indicate that there are more bytes to be read.

It is a requirement of bitstream conformance that the most significant bit of leb128\_byte is equal to 0 if i is equal to 7. (This ensures that this syntax descriptor never uses more than 8 bytes.)

**Note:** There are multiple ways of encoding the same value depending on how many leading zero bits are encoded. There is no requirement that this syntax descriptor uses the most compressed representation. This can be useful for encoder implementations by allowing a fixed amount of space to be filled in later when the value becomes known.

## 4.10.6. su(n)

Signed integer converted from an n bits unsigned integer in the bitstream. (The unsigned integer corresponds to the bottom n bits of the signed integer.) The parsing process for this descriptor is specified below:

su(n) {	Type
value	f(n)
signMask = 1 << (n - 1)	
if ( value & signMask )	
value = value - 2 * signMask	
return value	
}	

### 4.10.7. ns(n)

Unsigned encoded integer with maximum number of values n (i.e. output in range 0..n-1).

This descriptor is similar to f(CeilLog2(n)), but reduces wastage incurred when encoding non-power of two value ranges by encoding 1 fewer bits for the lower part of the value range. For example, when n is equal to 5, the encodings are as follows (full binary encodings are also presented for comparison):

Value	Full binary encoding	ns(n) encoding
0	000	00
1	001	01
2	010	10
3	011	110
4	100	111

The parsing process for this descriptor is specified as:

ns( n ) {	Type
w = FloorLog2(n) + 1	
m = (1 << w) - n	
v	f(w - 1)
if ( v < m )	
return v	
extra_bit	f(1)
return (v << 1) - m + extra_bit	
}	

The abbreviation ns stands for non-symmetric. This encoding is non-symmetric because the values are not all coded with the same number of bits.

### 4.10.8. L(n)

Unsigned arithmetic encoded n-bit number encoded as n flags (a “literal”). The flags are read from high to low order. The syntax element is set equal to the return value of read\_literal( n ) (see [section 8.2.5](#) for a specification of this process).

### 4.10.9. S()

An arithmetic encoded symbol coded from a small alphabet of at most 16 entries.

The symbol is decoded based on a context sensitive CDF (see [section 8.3](#) for the specification of this process).

### 4.10.10. NS(n)

Unsigned arithmetic encoded integer with maximum number of values n (i.e. output in range 0..n-1).

This descriptor is the same as ns(n) except the underlying bits are coded arithmetically.

The parsing process for this descriptor is specified as:

NS( n ) {	Type
w = FloorLog2(n) + 1	
m = (1 << w) - n	
v	L(w - 1)
if ( v < m )	
return v	
extra_bit	L(1)
return (v << 1) - m + extra_bit	
}	

## 5. Syntax structures

### 5.1. General

This section presents the syntax structures in a tabular form. The meaning of each of the syntax elements is presented in [Section 6](#).

### 5.2. Low overhead bitstream format

This specification defines a low-overhead bitstream format as a sequence of the OBU syntactical elements defined in this section. When using this format, `obu_has_size_field` must be equal to 1. For applications requiring a format where it is easier to skip through frames or temporal units, a length-delimited bitstream format is defined in Annex B.

Derived specifications, such as container formats enabling storage of AV1 videos together with audio or subtitles, should indicate which of these formats they rely on. Other methods of packing OBUs into a bitstream format are also allowed.

### 5.3. OBU syntax

#### 5.3.1. General OBU syntax

open_bitstream_unit( sz ) {	Type
obu_header()	
if ( obu_has_size_field ) {	
obu_size	leb128()
} else {	
obu_size = sz - 1 - obu_extension_flag	
}	
startPosition = get_position( )	
if ( obu_type != OBU_SEQUENCE_HEADER &&	
obu_type != OBU_TEMPORAL_DELIMITER &&	
OperatingPointIdc != 0 &&	
obu_extension_flag == 1 )	
{	
inTemporalLayer = (OperatingPointIdc >> temporal_id ) & 1	
inSpatialLayer = (OperatingPointIdc >> ( spatial_id + 8 ) ) & 1	
if ( !inTemporalLayer    ! inSpatialLayer ) {	
drop_obu( )	
return	
}	
}	
if ( obu_type == OBU_SEQUENCE_HEADER )	
sequence_header_obu( )	
else if ( obu_type == OBU_TEMPORAL_DELIMITER )	
temporal_delimiter_obu( )	
else if ( obu_type == OBU_FRAME_HEADER )	
frame_header_obu( )	
else if ( obu_type == OBU_REDUNDANT_FRAME_HEADER )	
frame_header_obu( )	

else if ( obu_type == OBU_TILE_GROUP )	
tile_group_obu( obu_size )	
else if ( obu_type == OBU_METADATA )	
metadata_obu( )	
else if ( obu_type == OBU_FRAME )	
frame_obu( obu_size )	
else if ( obu_type == OBU_TILE_LIST )	
tile_list_obu( )	
else if ( obu_type == OBU_PADDING )	
padding_obu( )	
else	
reserved_obu( )	
currentPosition = get_position( )	
payloadBits = currentPosition - startPosition	
if ( obu_size > 0 && obu_type != OBU_TILE_GROUP &&	
obu_type != OBU_FRAME ) {	
trailing_bits( obu_size * 8 - payloadBits )	
}	
}	

### 5.3.2. OBU header syntax

obu_header() {	Type
obu_forbidden_bit	f(1)
obu_type	f(4)
obu_extension_flag	f(1)
obu_has_size_field	f(1)
obu_reserved_1bit	f(1)
if ( obu_extension_flag == 1 )	
obu_extension_header()	
}	

### 5.3.3. OBU extension header syntax

obu_extension_header() {	Type
temporal_id	f(3)
spatial_id	f(2)
extension_header_reserved_3bits	f(3)
}	

### 5.3.4. Trailing bits syntax

trailing_bits( nbBits ) {	Type
trailing_one_bit	f(1)
nbBits--	
while ( nbBits > 0 ) {	
trailing_zero_bit	f(1)
nbBits--	
}	
}	



### 5.3.5. Byte alignment syntax

byte_alignment( ) {	Type
while ( get_position( ) & 7 )	
zero_bit	f(1)
}	

### 5.4. Reserved OBU syntax

reserved_obu( ) {	Type
}	

**Note:** Reserved OBUs do not have a defined syntax. The obu\_type reserved values are reserved for future use. Decoders should ignore the entire OBU if they do not understand the obu\_type. Ignoring the OBU can be done based on obu\_size. The last byte of the valid content of the payload data for this OBU type is considered to be the last byte that is not equal to zero. This rule is to prevent the dropping of valid bytes by systems that interpret trailing zero bytes as a continuation of the trailing bits in an OBU. This implies that when any payload data is present for this OBU type, at least one byte of the payload data (including the trailing bit) shall not be equal to 0.

### 5.5. Sequence header OBU syntax

#### 5.5.1. General sequence header OBU syntax

sequence_header_obu( ) {	Type
seq_profile	f(3)
still_picture	f(1)
reduced_still_picture_header	f(1)
if ( reduced_still_picture_header ) {	
timing_info_present_flag = 0	
decoder_model_info_present_flag = 0	
initial_display_delay_present_flag = 0	
operating_points_cnt_minus_1 = 0	
operating_point_idx[ 0 ] = 0	
seq_level_idx[ 0 ]	f(5)
seq_tier[ 0 ] = 0	
decoder_model_present_for_this_op[ 0 ] = 0	
initial_display_delay_present_for_this_op[ 0 ] = 0	
} else {	
timing_info_present_flag	f(1)
if ( timing_info_present_flag ) {	
timing_info( )	
decoder_model_info_present_flag	f(1)
if ( decoder_model_info_present_flag ) {	
decoder_model_info( )	
}	
} else {	
decoder_model_info_present_flag = 0	

}	
initial_display_delay_present_flag	f(1)
operating_points_cnt_minus_1	f(5)
for ( i = 0; i <= operating_points_cnt_minus_1; i++ ) {	
operating_point_idc[ i ]	f(12)
seq_level_idx[ i ]	f(5)
if ( seq_level_idx[ i ] > 7 ) {	
seq_tier[ i ]	f(1)
} else {	
seq_tier[ i ] = 0	
}	
if ( decoder_model_info_present_flag ) {	
decoder_model_present_for_this_op[ i ]	f(1)
if ( decoder_model_present_for_this_op[ i ] ) {	
operating_parameters_info( i )	
}	
} else {	
decoder_model_present_for_this_op[ i ] = 0	
}	
if ( initial_display_delay_present_flag ) {	
initial_display_delay_present_for_this_op[ i ]	f(1)
if ( initial_display_delay_present_for_this_op[ i ] ) {	
initial_display_delay_minus_1[ i ]	f(4)
}	
}	
}	
operatingPoint = choose_operating_point( )	
OperatingPointIdc = operating_point_idc[ operatingPoint ]	
frame_width_bits_minus_1	f(4)
frame_height_bits_minus_1	f(4)
n = frame_width_bits_minus_1 + 1	
max_frame_width_minus_1	f(n)
n = frame_height_bits_minus_1 + 1	
max_frame_height_minus_1	f(n)
if ( reduced_still_picture_header )	
frame_id_numbers_present_flag = 0	
else	
frame_id_numbers_present_flag	f(1)
if ( frame_id_numbers_present_flag ) {	
delta_frame_id_length_minus_2	f(4)
additional_frame_id_length_minus_1	f(3)
}	
use_128x128_superblock	f(1)
enable_filter_intra	f(1)
enable_intra_edge_filter	f(1)
if ( reduced_still_picture_header ) {	
enable_interintra_compound = 0	
enable_masked_compound = 0	

enable_warped_motion = 0	
enable_dual_filter = 0	
enable_order_hint = 0	
enable_jnt_comp = 0	
enable_ref_frame_mvs = 0	
seq_force_screen_content_tools = SELECT_SCREEN_CONTENT_TOOLS	
seq_force_integer_mv = SELECT_INTEGER_MV	
OrderHintBits = 0	
} else {	
enable_interintra_compound	f(1)
enable_masked_compound	f(1)
enable_warped_motion	f(1)
enable_dual_filter	f(1)
enable_order_hint	f(1)
if ( enable_order_hint ) {	
enable_jnt_comp	f(1)
enable_ref_frame_mvs	f(1)
} else {	
enable_jnt_comp = 0	
enable_ref_frame_mvs = 0	
}	
seq_choose_screen_content_tools	f(1)
if ( seq_choose_screen_content_tools ) {	
seq_force_screen_content_tools = SELECT_SCREEN_CONTENT_TOOLS	
} else {	
seq_force_screen_content_tools	f(1)
}	
if ( seq_force_screen_content_tools > 0 ) {	
seq_choose_integer_mv	f(1)
if ( seq_choose_integer_mv ) {	
seq_force_integer_mv = SELECT_INTEGER_MV	
} else {	
seq_force_integer_mv	f(1)
}	
} else {	
seq_force_integer_mv = SELECT_INTEGER_MV	
}	
if ( enable_order_hint ) {	
order_hint_bits_minus_1	f(3)
OrderHintBits = order_hint_bits_minus_1 + 1	
} else {	
OrderHintBits = 0	
}	
}	
enable_superres	f(1)
enable_cdef	f(1)
enable_restoration	f(1)
color_config( )	

film_grain_params_present	f(1)
}	

## 5.5.2. Color config syntax

color_config( ) {	Type
high_bitdepth	f(1)
if ( seq_profile == 2 && high_bitdepth ) {	
twelve_bit	f(1)
BitDepth = twelve_bit ? 12 : 10	
} else if ( seq_profile <= 2 ) {	
BitDepth = high_bitdepth ? 10 : 8	
}	
if ( seq_profile == 1 ) {	
mono_chrome = 0	
} else {	
mono_chrome	f(1)
}	
NumPlanes = mono_chrome ? 1 : 3	
color_description_present_flag	f(1)
if ( color_description_present_flag ) {	
color_primaries	f(8)
transfer_characteristics	f(8)
matrix_coefficients	f(8)
} else {	
color_primaries = CP_UNSPECIFIED	
transfer_characteristics = TC_UNSPECIFIED	
matrix_coefficients = MC_UNSPECIFIED	
}	
if ( mono_chrome ) {	
color_range	f(1)
subsampling_x = 1	
subsampling_y = 1	
chroma_sample_position = CSP_UNKNOWN	
separate_uv_delta_q = 0	
return	
} else if ( color_primaries == CP_BT_709 &&	
transfer_characteristics == TC_SRGB &&	
matrix_coefficients == MC_IDENTITY ) {	
color_range = 1	
subsampling_x = 0	
subsampling_y = 0	
} else {	
color_range	f(1)
if ( seq_profile == 0 ) {	
subsampling_x = 1	
subsampling_y = 1	
} else if ( seq_profile == 1 ) {	
subsampling_x = 0	

subsampling_y = 0	
} else {	
if ( BitDepth == 12 ) {	
subsampling_x	f(1)
if ( subsampling_x )	
subsampling_y	f(1)
else	
subsampling_y = 0	
} else {	
subsampling_x = 1	
subsampling_y = 0	
}	
}	
if ( subsampling_x && subsampling_y ) {	
chroma_sample_position	f(2)
}	
}	
separate_uv_delta_q	f(1)
}	

### 5.5.3. Timing info syntax

timing_info( ) {	<b>Type</b>
num_units_in_display_tick	f(32)
time_scale	f(32)
equal_picture_interval	f(1)
if ( equal_picture_interval )	
num_ticks_per_picture_minus_1	uvlc()
}	

### 5.5.4. Decoder model info syntax

decoder_model_info( ) {	<b>Type</b>
buffer_delay_length_minus_1	f(5)
num_units_in_decoding_tick	f(32)
buffer_removal_time_length_minus_1	f(5)
frame_presentation_time_length_minus_1	f(5)
}	

### 5.5.5. Operating parameters info syntax

operating_parameters_info( op ) {	<b>Type</b>
n = buffer_delay_length_minus_1 + 1	
decoder_buffer_delay[ op ]	f(n)
encoder_buffer_delay[ op ]	f(n)
low_delay_mode_flag[ op ]	f(1)
}	

## 5.6. Temporal delimiter obu syntax

temporal_delimiter_obu( ) {	Type
SeenFrameHeader = 0	
}	

**Note:** The temporal delimiter has an empty payload.

## 5.7. Padding OBU syntax

padding_obu( ) {	Type
for ( i = 0; i < obu_padding_length; i++ )	
obu_padding_byte	f(8)
}	

**Note:** obu\_padding\_length is not coded in the bitstream but can be computed based on obu\_size minus the number of trailing bytes. In practice, though, since this is padding data meant to be skipped, decoders do not need to determine either that length nor the number of trailing bytes. They can ignore the entire OBU. Ignoring the OBU can be done based on obu\_size. The last byte of the valid content of the payload data for this OBU type is considered to be the last byte that is not equal to zero. This rule is to prevent the dropping of valid bytes by systems that interpret trailing zero bytes as a continuation of the trailing bits in an OBU. This implies that when any payload data is present for this OBU type, at least one byte of the payload data (including the trailing bit) shall not be equal to 0.

## 5.8. Metadata OBU syntax

### 5.8.1. General metadata OBU syntax

metadata_obu( ) {	Type
metadata_type	leb128()
if ( metadata_type == METADATA_TYPE_ITUT_T35 )	
metadata_itut_t35( )	
else if ( metadata_type == METADATA_TYPE_HDR_CLL )	
metadata_hdr_cll( )	
else if ( metadata_type == METADATA_TYPE_HDR_MDCV )	
metadata_hdr_mdcv( )	
else if ( metadata_type == METADATA_TYPE_SCALABILITY )	
metadata_scalability( )	
else if ( metadata_type == METADATA_TYPE_TIMECODE )	
metadata_timecode( )	
}	

### 5.8.2. Metadata ITUT T35 syntax

metadata_itut_t35( ) {	Type
itu_t_t35_country_code	f(8)

if ( itu_t_t35_country_code == 0xFF ) {	
itu_t_t35_country_code_extension_byte	f(8)
}	
itu_t_t35_payload_bytes	
}	

**Note:** The exact syntax of `itu_t_t35_payload_bytes` is not defined in this specification. External specifications can define the syntax. From a decoder perspective, it suffices to say that decoders should ignore the entire OBU if they don't understand it. The last byte of the valid content of the data is considered to be the last byte that is not equal to zero. This rule is to prevent the dropping of valid bytes by systems that interpret trailing zero bytes as a padding continuation of the trailing bits in an OBU. This implies that when any payload data is present for this OBU type, at least one byte of the payload data (including the trailing bit) shall not be equal to 0.

### 5.8.3. Metadata high dynamic range content light level syntax

metadata_hdr_cll( ) {	Type
max_cll	f(16)
max_fall	f(16)
}	

### 5.8.4. Metadata high dynamic range mastering display color volume syntax

metadata_hdr_mdcv( ) {	Type
for ( i = 0; i < 3; i++ ) {	
primary_chromaticity_x[ i ]	f(16)
primary_chromaticity_y[ i ]	f(16)
}	
white_point_chromaticity_x	f(16)
white_point_chromaticity_y	f(16)
luminance_max	f(32)
luminance_min	f(32)
}	

### 5.8.5. Metadata scalability syntax

metadata_scalability( ) {	Type
scalability_mode_idc	f(8)
if ( scalability_mode_idc == SCALABILITY_SS )	
scalability_structure( )	
}	

### 5.8.6. Scalability structure syntax

scalability_structure( ) {	Type
spatial_layers_cnt_minus_1	f(2)
spatial_layer_dimensions_present_flag	f(1)
spatial_layer_description_present_flag	f(1)

temporal_group_description_present_flag	f(1)
scalability_structure_reserved_3bits	f(3)
if ( spatial_layer_dimensions_present_flag ) {	
for ( i = 0; i <= spatial_layers_cnt_minus_1 ; i++ ) {	
spatial_layer_max_width[ i ]	f(16)
spatial_layer_max_height[ i ]	f(16)
}	
}	
if ( spatial_layer_description_present_flag ) {	
for ( i = 0; i <= spatial_layers_cnt_minus_1; i++ )	
spatial_layer_ref_id[ i ]	f(8)
}	
if ( temporal_group_description_present_flag ) {	
temporal_group_size	f(8)
for ( i = 0; i < temporal_group_size; i++ ) {	
temporal_group_temporal_id[ i ]	f(3)
temporal_group_temporal_switching_up_point_flag[ i ]	f(1)
temporal_group_spatial_switching_up_point_flag[ i ]	f(1)
temporal_group_ref_cnt[ i ]	f(3)
for ( j = 0; j < temporal_group_ref_cnt[ i ]; j++ ) {	
temporal_group_ref_pic_diff[ i ][ j ]	f(8)
}	
}	
}	
}	

### 5.8.7. Metadata timecode syntax

metadata_timecode( ) {	Type
counting_type	f(5)
full_timestamp_flag	f(1)
discontinuity_flag	f(1)
cnt_dropped_flag	f(1)
n_frames	f(9)
if ( full_timestamp_flag ) {	
seconds_value	f(6)
minutes_value	f(6)
hours_value	f(5)
} else {	
seconds_flag	f(1)
if ( seconds_flag ) {	
seconds_value	f(6)
minutes_flag	f(1)
if ( minutes_flag ) {	
minutes_value	f(6)
hours_flag	f(1)
if ( hours_flag ) {	
hours_value	f(5)
}	
}	
}	



}	
}	
}	
time_offset_length	f(5)
if ( time_offset_length > 0 ) {	
time_offset_value	f(time_offset_length)
}	
}	

## 5.9. Frame header OBU syntax

### 5.9.1. General frame header OBU syntax

frame_header_obu( ) {	Type
if ( SeenFrameHeader == 1 ) {	
frame_header_copy()	
} else {	
SeenFrameHeader = 1	
uncompressed_header( )	
if ( show_existing_frame ) {	
decode_frame_wrapup( )	
SeenFrameHeader = 0	
} else {	
TileNum = 0	
SeenFrameHeader = 1	
}	
}	
}	

### 5.9.2. Uncompressed header syntax

uncompressed_header( ) {	Type
if ( frame_id_numbers_present_flag ) {	
idLen = ( additional_frame_id_length_minus_1 +	
delta_frame_id_length_minus_2 + 3 )	
}	
allFrames = (1 << NUM_REF_FRAMES) - 1	
if ( reduced_still_picture_header ) {	
show_existing_frame = 0	
frame_type = KEY_FRAME	
FrameIsIntra = 1	
show_frame = 1	
showable_frame = 0	
} else {	
show_existing_frame	f(1)
if ( show_existing_frame == 1 ) {	
frame_to_show_map_idx	f(3)
if ( decoder_model_info_present_flag && !equal_picture_interval ) {	
temporal_point_info( )	

}	
refresh_frame_flags = 0	
if ( frame_id_numbers_present_flag ) {	
display_frame_id	f(idLen)
}	
frame_type = RefFrameType[ frame_to_show_map_idx ]	
if ( frame_type == KEY_FRAME ) {	
refresh_frame_flags = allFrames	
}	
if ( film_grain_params_present ) {	
load_grain_params( frame_to_show_map_idx )	
}	
return	
}	
frame_type	f(2)
FrameIsIntra = (frame_type == INTRA_ONLY_FRAME	
frame_type == KEY_FRAME)	
show_frame	f(1)
if ( show_frame && decoder_model_info_present_flag && !equal_picture_interval ) {	
temporal_point_info( )	
}	
if ( show_frame ) {	
showable_frame = frame_type != KEY_FRAME	
} else {	
showable_frame	f(1)
}	
if ( frame_type == SWITCH_FRAME	
( frame_type == KEY_FRAME && show_frame ) )	
error_resilient_mode = 1	
else	
error_resilient_mode	f(1)
}	
if ( frame_type == KEY_FRAME && show_frame ) {	
for ( i = 0; i < NUM_REF_FRAMES; i++ ) {	
RefValid[ i ] = 0	
RefOrderHint[ i ] = 0	
}	
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
OrderHints[ LAST_FRAME + i ] = 0	
}	
}	
disable_cdf_update	f(1)
if ( seq_force_screen_content_tools == SELECT_SCREEN_CONTENT_TOOLS ) {	
allow_screen_content_tools	f(1)
} else {	
allow_screen_content_tools = seq_force_screen_content_tools	
}	
if ( allow_screen_content_tools ) {	
if ( seq_force_integer_mv == SELECT_INTEGER_MV ) {	

force_integer_mv	f(1)
} else {	
force_integer_mv = seq_force_integer_mv	
}	
} else {	
force_integer_mv = 0	
}	
if ( FrameIsIntra ) {	
force_integer_mv = 1	
}	
if ( frame_id_numbers_present_flag ) {	
PrevFrameID = current_frame_id	
current_frame_id	f(idLen)
mark_ref_frames( idLen )	
} else {	
current_frame_id = 0	
}	
if ( frame_type == SWITCH_FRAME )	
frame_size_override_flag = 1	
else if ( reduced_still_picture_header )	
frame_size_override_flag = 0	
else	
frame_size_override_flag	f(1)
order_hint	f(OrderHintBits)
OrderHint = order_hint	
if ( FrameIsIntra    error_resilient_mode ) {	
primary_ref_frame = PRIMARY_REF_NONE	
} else {	
primary_ref_frame	f(3)
}	
if ( decoder_model_info_present_flag ) {	
buffer_removal_time_present_flag	f(1)
if ( buffer_removal_time_present_flag ) {	
for ( opNum = 0; opNum <= operating_points_cnt_minus_1; opNum++ ) {	
if ( decoder_model_present_for_this_op[ opNum ] ) {	
opPtIdc = operating_point_idc[ opNum ]	
inTemporalLayer = ( opPtIdc >> temporal_id ) & 1	
inSpatialLayer = ( opPtIdc >> ( spatial_id + 8 ) ) & 1	
if ( opPtIdc == 0    ( inTemporalLayer && inSpatialLayer ) ) {	
n = buffer_removal_time_length_minus_1 + 1	
buffer_removal_time[ opNum ]	f(n)
}	
}	
}	
}	
}	
allow_high_precision_mv = 0	
use_ref_frame_mvs = 0	
allow_intrabc = 0	

if ( frame_type == SWITCH_FRAME	
( frame_type == KEY_FRAME && show_frame ) ) {	
refresh_frame_flags = allFrames	
} else {	
refresh_frame_flags	f(8)
}	
if ( !FrameIsIntra    refresh_frame_flags != allFrames ) {	
if ( error_resilient_mode && enable_order_hint ) {	
for ( i = 0; i < NUM_REF_FRAMES; i++ ) {	
ref_order_hint[ i ]	f(OrderHintBits)
if ( ref_order_hint[ i ] != RefOrderHint[ i ] ) {	
RefValid[ i ] = 0	
}	
}	
}	
if ( frame_type == KEY_FRAME ) {	
frame_size( )	
render_size( )	
if ( allow_screen_content_tools && UpscaledWidth == FrameWidth ) {	
allow_intrabc	f(1)
}	
} else {	
if ( frame_type == INTRA_ONLY_FRAME ) {	
frame_size( )	
render_size( )	
if ( allow_screen_content_tools && UpscaledWidth == FrameWidth ) {	
allow_intrabc	f(1)
}	
} else {	
if ( !enable_order_hint ) {	
frame_refs_short_signaling = 0	
} else {	
frame_refs_short_signaling	f(1)
if ( frame_refs_short_signaling ) {	
last_frame_idx	f(3)
gold_frame_idx	f(3)
set_frame_refs()	
}	
}	
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
if ( !frame_refs_short_signaling )	
ref_frame_idx[ i ]	f(3)
if ( frame_id_numbers_present_flag ) {	
n = delta_frame_id_length_minus_2 + 2	
delta_frame_id_minus_1	f(n)
DeltaFrameId = delta_frame_id_minus_1 + 1	
expectedFrameId[ i ] = ((current_frame_id + (1 << idLen) -	
DeltaFrameId ) % (1 << idLen))	

}	
}	
if ( frame_size_override_flag && !error_resilient_mode ) {	
frame_size_with_refs( )	
} else {	
frame_size( )	
render_size( )	
}	
if ( force_integer_mv ) {	
allow_high_precision_mv = 0	
} else {	
allow_high_precision_mv	f(1)
}	
read_interpolation_filter( )	
is_motion_mode_switchable	f(1)
if ( error_resilient_mode	
!enable_ref_frame_mvs ) {	
use_ref_frame_mvs = 0	
} else {	
use_ref_frame_mvs	f(1)
}	
}	
}	
if ( !FrameIsIntra ) {	
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
refFrame = LAST_FRAME + i	
hint = RefOrderHint[ ref_frame_idx[ i ] ]	
OrderHints[ refFrame ] = hint	
if ( !enable_order_hint ) {	
RefFrameSignBias[ refFrame ] = 0	
} else {	
RefFrameSignBias[ refFrame ] = get_relative_dist( hint, OrderHint) > 0	
}	
}	
}	
if ( reduced_still_picture_header    disable_cdf_update )	
disable_frame_end_update_cdf = 1	
else	
disable_frame_end_update_cdf	f(1)
if ( primary_ref_frame == PRIMARY_REF_NONE ) {	
init_non_coeff_cdfs( )	
setup_past_independence( )	
} else {	
load_cdfs( ref_frame_idx[ primary_ref_frame ] )	
load_previous( )	
}	
if ( use_ref_frame_mvs == 1 )	
motion_field_estimation( )	
tile_info( )	

quantization_params( )	
segmentation_params( )	
delta_q_params( )	
delta_lf_params( )	
if ( primary_ref_frame == PRIMARY_REF_NONE ) {	
init_coeff_cdfs( )	
} else {	
load_previous_segment_ids( )	
}	
CodedLossless = 1	
for ( segmentId = 0; segmentId < MAX_SEGMENTS; segmentId++ ) {	
qindex = get_qindex( 1, segmentId )	
LosslessArray[ segmentId ] = qindex == 0 && DeltaQYDc == 0 &&	
DeltaQUAc == 0 && DeltaQUDc == 0 &&	
DeltaQVAc == 0 && DeltaQVDc == 0	
if ( !LosslessArray[ segmentId ] )	
CodedLossless = 0	
if ( using_qmatrix ) {	
if ( LosslessArray[ segmentId ] ) {	
SegQMLLevel[ 0 ][ segmentId ] = 15	
SegQMLLevel[ 1 ][ segmentId ] = 15	
SegQMLLevel[ 2 ][ segmentId ] = 15	
} else {	
SegQMLLevel[ 0 ][ segmentId ] = qm_y	
SegQMLLevel[ 1 ][ segmentId ] = qm_u	
SegQMLLevel[ 2 ][ segmentId ] = qm_v	
}	
}	
}	
AllLossless = CodedLossless && ( FrameWidth == UpscaledWidth )	
loop_filter_params( )	
cdef_params( )	
lr_params( )	
read_tx_mode( )	
frame_reference_mode( )	
skip_mode_params( )	
if ( FrameIsIntra	
error_resilient_mode	
!enable_warped_motion )	
allow_warped_motion = 0	
else	
allow_warped_motion	f(1)
reduced_tx_set	f(1)
global_motion_params( )	
film_grain_params( )	
}	

### 5.9.3. Get relative distance function

This function computes the distance between two order hints by sign extending the result of subtracting the values.

get_relative_dist( a, b ) {	Type
if ( !enable_order_hint )	
return 0	
diff = a - b	
m = 1 << (OrderHintBits - 1)	
diff = (diff & (m - 1)) - (diff & m)	
return diff	
}	

### 5.9.4. Reference frame marking function

mark_ref_frames( idLen ) {	Type
diffLen = delta_frame_id_length_minus_2 + 2	
for ( i = 0; i < NUM_REF_FRAMES; i++ ) {	
if ( current_frame_id > ( 1 << diffLen ) ) {	
if ( RefFrameId[ i ] > current_frame_id	
RefFrameId[ i ] < ( current_frame_id - ( 1 << diffLen ) ) )	
RefValid[ i ] = 0	
} else {	
if ( RefFrameId[ i ] > current_frame_id &&	
RefFrameId[ i ] < ( ( 1 << idLen ) +	
current_frame_id -	
( 1 << diffLen ) ) )	
RefValid[ i ] = 0	
}	
}	
}	

### 5.9.5. Frame size syntax

frame_size( ) {	Type
if ( frame_size_override_flag ) {	
n = frame_width_bits_minus_1 + 1	
frame_width_minus_1	f(n)
n = frame_height_bits_minus_1 + 1	
frame_height_minus_1	f(n)
FrameWidth = frame_width_minus_1 + 1	
FrameHeight = frame_height_minus_1 + 1	
} else {	
FrameWidth = max_frame_width_minus_1 + 1	
FrameHeight = max_frame_height_minus_1 + 1	
}	
superres_params( )	
compute_image_size( )	
}	

## 5.9.6. Render size syntax

render_size( ) {	Type
render_and_frame_size_different	f(1)
if ( render_and_frame_size_different == 1 ) {	
render_width_minus_1	f(16)
render_height_minus_1	f(16)
RenderWidth = render_width_minus_1 + 1	
RenderHeight = render_height_minus_1 + 1	
} else {	
RenderWidth = UpscaledWidth	
RenderHeight = FrameHeight	
}	
}	

## 5.9.7. Frame size with refs syntax

frame_size_with_refs( ) {	Type
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
found_ref	f(1)
if ( found_ref == 1 ) {	
UpscaledWidth = RefUpscaledWidth[ ref_frame_idx[ i ] ]	
FrameWidth = UpscaledWidth	
FrameHeight = RefFrameHeight[ ref_frame_idx[ i ] ]	
RenderWidth = RefRenderWidth[ ref_frame_idx[ i ] ]	
RenderHeight = RefRenderHeight[ ref_frame_idx[ i ] ]	
break	
}	
}	
if ( found_ref == 0 ) {	
frame_size( )	
render_size( )	
} else {	
superres_params( )	
compute_image_size( )	
}	
}	

## 5.9.8. Superres params syntax

superres_params( ) {	Type
if ( enable_superres )	
use_superres	f(1)
else	
use_superres = 0	
if ( use_superres ) {	
coded_denom	f(SUPERRES_DENOM_BITS)
SuperresDenom = coded_denom + SUPERRES_DENOM_MIN	
} else {	
SuperresDenom = SUPERRES_NUM	



}	
UpscaledWidth = FrameWidth	
FrameWidth = (UpscaledWidth * SUPERRES_NUM + (SuperresDenom / 2)) / SuperresDenom	
}	

### 5.9.9. Compute image size function

compute_image_size( ) {	Type
MiCols = 2 * ( ( FrameWidth + 7 ) >> 3 )	
MiRows = 2 * ( ( FrameHeight + 7 ) >> 3 )	
}	

### 5.9.10. Interpolation filter syntax

read_interpolation_filter( ) {	Type
is_filter_switchable	f(1)
if ( is_filter_switchable == 1 ) {	
interpolation_filter = SWITCHABLE	
} else {	
interpolation_filter	f(2)
}	
}	

### 5.9.11. Loop filter params syntax

loop_filter_params( ) {	Type
if ( CodedLossless    allow_intrabc ) {	
loop_filter_level[ 0 ] = 0	
loop_filter_level[ 1 ] = 0	
loop_filter_ref_deltas[ INTRA_FRAME ] = 1	
loop_filter_ref_deltas[ LAST_FRAME ] = 0	
loop_filter_ref_deltas[ LAST2_FRAME ] = 0	
loop_filter_ref_deltas[ LAST3_FRAME ] = 0	
loop_filter_ref_deltas[ BWDREF_FRAME ] = 0	
loop_filter_ref_deltas[ GOLDEN_FRAME ] = -1	
loop_filter_ref_deltas[ ALTREF_FRAME ] = -1	
loop_filter_ref_deltas[ ALTREF2_FRAME ] = -1	
for ( i = 0; i < 2; i++ ) {	
loop_filter_mode_deltas[ i ] = 0	
}	
return	
}	
loop_filter_level[ 0 ]	f(6)
loop_filter_level[ 1 ]	f(6)
if ( NumPlanes > 1 ) {	
if ( loop_filter_level[ 0 ]    loop_filter_level[ 1 ] ) {	
loop_filter_level[ 2 ]	f(6)
loop_filter_level[ 3 ]	f(6)
}	

}	
loop_filter_sharpness	f(3)
loop_filter_delta_enabled	f(1)
if ( loop_filter_delta_enabled == 1 ) {	
loop_filter_delta_update	f(1)
if ( loop_filter_delta_update == 1 ) {	
for ( i = 0; i < TOTAL_REFS_PER_FRAME; i++ ) {	
update_ref_delta	f(1)
if ( update_ref_delta == 1 )	
loop_filter_ref_deltas[ i ]	su(1+6)
}	
for ( i = 0; i < 2; i++ ) {	
update_mode_delta	f(1)
if ( update_mode_delta == 1 )	
loop_filter_mode_deltas[ i ]	su(1+6)
}	
}	
}	
}	

## 5.9.12. Quantization params syntax

quantization_params( ) {	Type
base_q_idx	f(8)
DeltaQYDc = read_delta_q( )	
if ( NumPlanes > 1 ) {	
if ( separate_uv_delta_q )	
diff_uv_delta	f(1)
else	
diff_uv_delta = 0	
DeltaQUdC = read_delta_q( )	
DeltaQUAc = read_delta_q( )	
if ( diff_uv_delta ) {	
DeltaQVDc = read_delta_q( )	
DeltaQVAc = read_delta_q( )	
} else {	
DeltaQVDc = DeltaQUdC	
DeltaQVAc = DeltaQUAc	
}	
} else {	
DeltaQUdC = 0	
DeltaQUAc = 0	
DeltaQVDc = 0	
DeltaQVAc = 0	
}	
using_qmatrix	f(1)
if ( using_qmatrix ) {	
qm_y	f(4)
qm_u	f(4)

if ( !separate_uv_delta_q )	
qm_v = qm_u	
else	
qm_v	f(4)
}	
}	

### 5.9.13. Delta quantizer syntax

read_delta_q( ) {	Type
delta_coded	f(1)
if ( delta_coded ) {	
delta_q	su(1+6)
} else {	
delta_q = 0	
}	
return delta_q	
}	

### 5.9.14. Segmentation params syntax

segmentation_params( ) {	Type
segmentation_enabled	f(1)
if ( segmentation_enabled == 1 ) {	
if ( primary_ref_frame == PRIMARY_REF_NONE ) {	
segmentation_update_map = 1	
segmentation_temporal_update = 0	
segmentation_update_data = 1	
} else {	
segmentation_update_map	f(1)
if ( segmentation_update_map == 1 )	
segmentation_temporal_update	f(1)
segmentation_update_data	f(1)
}	
if ( segmentation_update_data == 1 ) {	
for ( i = 0; i < MAX_SEGMENTS; i++ ) {	
for ( j = 0; j < SEG_LVL_MAX; j++ ) {	
feature_value = 0	
feature_enabled	f(1)
FeatureEnabled[ i ][ j ] = feature_enabled	
clippedValue = 0	
if ( feature_enabled == 1 ) {	
bitsToRead = Segmentation_Feature_Bits[ j ]	
limit = Segmentation_Feature_Max[ j ]	
if ( Segmentation_Feature_Signed[ j ] == 1 ) {	
feature_value	su(1+bitsToRead)
clippedValue = Clip3( -limit, limit, feature_value)	
} else {	
feature_value	f(bitsToRead)
clippedValue = Clip3( 0, limit, feature_value)	
}	
}	
}	

}	
}	
FeatureData[ i ][ j ] = clippedValue	
}	
}	
}	
} else {	
for ( i = 0; i < MAX_SEGMENTS; i++ ) {	
for ( j = 0; j < SEG_LVL_MAX; j++ ) {	
FeatureEnabled[ i ][ j ] = 0	
FeatureData[ i ][ j ] = 0	
}	
}	
}	
}	
SegIdPreSkip = 0	
LastActiveSegId = 0	
for ( i = 0; i < MAX_SEGMENTS; i++ ) {	
for ( j = 0; j < SEG_LVL_MAX; j++ ) {	
if ( FeatureEnabled[ i ][ j ] ) {	
LastActiveSegId = i	
if ( j >= SEG_LVL_REF_FRAME ) {	
SegIdPreSkip = 1	
}	
}	
}	
}	
}	
}	

The constant lookup tables used in this syntax are defined as:

```

Segmentation_Feature_Bits[ SEG_LVL_MAX ] = { 8, 6, 6, 6, 6, 3, 0, 0 }
Segmentation_Feature_Signed[ SEG_LVL_MAX ] = { 1, 1, 1, 1, 1, 0, 0, 0 }
Segmentation_Feature_Max[ SEG_LVL_MAX ] = {
    255, MAX_LOOP_FILTER, MAX_LOOP_FILTER,
    MAX_LOOP_FILTER, MAX_LOOP_FILTER, 7,
    0, 0 }

```

## 5.9.15. Tile info syntax

tile_info ( ) {	Type
sbCols = use_128x128_superblock ? ( ( MiCols + 31 ) >> 5 ) : ( ( MiCols + 15 ) >> 4 )	
sbRows = use_128x128_superblock ? ( ( MiRows + 31 ) >> 5 ) : ( ( MiRows + 15 ) >> 4 )	
sbShift = use_128x128_superblock ? 5 : 4	
sbSize = sbShift + 2	
maxTileWidthSb = MAX_TILE_WIDTH >> sbSize	
maxTileAreaSb = MAX_TILE_AREA >> ( 2 * sbSize )	
minLog2TileCols = tile_log2(maxTileWidthSb, sbCols)	
maxLog2TileCols = tile_log2(1, Min(sbCols, MAX_TILE_COLS))	

maxLog2TileRows = tile_log2(1, Min(sbRows, MAX_TILE_ROWS))	
minLog2Tiles = Max(minLog2TileCols,	
tile_log2(maxTileAreaSb, sbRows * sbCols))	
<b>uniform_tile_spacing_flag</b>	f(1)
if ( uniform_tile_spacing_flag ) {	
TileColsLog2 = minLog2TileCols	
while ( TileColsLog2 < maxLog2TileCols ) {	
increment_tile_cols_log2	f(1)
if ( increment_tile_cols_log2 == 1 )	
TileColsLog2++	
else	
break	
}	
tileWidthSb = (sbCols + (1 << TileColsLog2) - 1) >> TileColsLog2	
i = 0	
for ( startSb = 0; startSb < sbCols; startSb += tileWidthSb ) {	
MiColStarts[ i ] = startSb << sbShift	
i += 1	
}	
MiColStarts[i] = MiCols	
TileCols = i	
minLog2TileRows = Max( minLog2Tiles - TileColsLog2, 0)	
TileRowsLog2 = minLog2TileRows	
while ( TileRowsLog2 < maxLog2TileRows ) {	
increment_tile_rows_log2	f(1)
if ( increment_tile_rows_log2 == 1 )	
TileRowsLog2++	
else	
break	
}	
tileHeightSb = (sbRows + (1 << TileRowsLog2) - 1) >> TileRowsLog2	
i = 0	
for ( startSb = 0; startSb < sbRows; startSb += tileHeightSb ) {	
MiRowStarts[ i ] = startSb << sbShift	
i += 1	
}	
MiRowStarts[i] = MiRows	
TileRows = i	
} else {	
widestTileSb = 0	
startSb = 0	
for ( i = 0; startSb < sbCols; i++ ) {	
MiColStarts[ i ] = startSb << sbShift	
maxWidth = Min(sbCols - startSb, maxTileWidthSb)	
width_in_sbs_minus_1	ns(maxWidth)
sizeSb = width_in_sbs_minus_1 + 1	
widestTileSb = Max( sizeSb, widestTileSb )	

startSb += sizeSb	
}	
MiColStarts[i] = MiCols	
TileCols = i	
TileColsLog2 = tile_log2(1, TileCols)	
if ( minLog2Tiles > 0 )	
maxTileAreaSb = (sbRows * sbCols) >> (minLog2Tiles + 1)	
else	
maxTileAreaSb = sbRows * sbCols	
maxTileHeightSb = Max( maxTileAreaSb / widestTileSb, 1 )	
startSb = 0	
for ( i = 0; startSb < sbRows; i++ ) {	
MiRowStarts[ i ] = startSb << sbShift	
maxHeight = Min(sbRows - startSb, maxTileHeightSb)	
height_in_sbs_minus_1	ns(maxHeight)
sizeSb = height_in_sbs_minus_1 + 1	
startSb += sizeSb	
}	
MiRowStarts[ i ] = MiRows	
TileRows = i	
TileRowsLog2 = tile_log2(1, TileRows)	
}	
if ( TileColsLog2 > 0    TileRowsLog2 > 0 ) {	
context_update_tile_id	f(TileRowsLog2 + TileColsLog2)
tile_size_bytes_minus_1	f(2)
TileSizeBytes = tile_size_bytes_minus_1 + 1	
} else {	
context_update_tile_id = 0	
}	
}	

## 5.9.16. Tile size calculation function

tile\_log2 returns the smallest value for k such that blkSize << k is greater than or equal to target.

tile_log2( blkSize, target ) {	Type
for ( k = 0; (blkSize << k) < target; k++ ) {	
}	
return k	
}	

## 5.9.17. Quantizer index delta parameters syntax

delta_q_params( ) {	Type
delta_q_res = 0	
delta_q_present = 0	

if ( base_q_idx > 0 ) {	
delta_q_present	f(1)
}	
if ( delta_q_present ) {	
delta_q_res	f(2)
}	
}	

### 5.9.18. Loop filter delta parameters syntax

delta_lf_params( ) {	Type
delta_lf_present = 0	
delta_lf_res = 0	
delta_lf_multi = 0	
if ( delta_q_present ) {	
if ( !allow_intrabc )	
delta_lf_present	f(1)
if ( delta_lf_present ) {	
delta_lf_res	f(2)
delta_lf_multi	f(1)
}	
}	
}	

### 5.9.19. CDEF params syntax

cdef_params( ) {	Type
if ( CodedLossless    allow_intrabc	
!enable_cdef) {	
cdef_bits = 0	
cdef_y_pri_strength[0] = 0	
cdef_y_sec_strength[0] = 0	
cdef_uv_pri_strength[0] = 0	
cdef_uv_sec_strength[0] = 0	
CdefDamping = 3	
return	
}	
cdef_damping_minus_3	f(2)
CdefDamping = cdef_damping_minus_3 + 3	
cdef_bits	f(2)
for ( i = 0; i < (1 << cdef_bits); i++ ) {	
cdef_y_pri_strength[i]	f(4)
cdef_y_sec_strength[i]	f(2)
if ( cdef_y_sec_strength[i] == 3 )	
cdef_y_sec_strength[i] += 1	
if ( NumPlanes > 1 ) {	
cdef_uv_pri_strength[i]	f(4)
cdef_uv_sec_strength[i]	f(2)
if ( cdef_uv_sec_strength[i] == 3 )	
cdef_uv_sec_strength[i] += 1	

}	
}	
}	

## 5.9.20. Loop restoration params syntax

lr_params( ) {	Type
if ( AllLossless    allow_intrabc	
!enable_restoration ) {	
FrameRestorationType[0] = RESTORE_NONE	
FrameRestorationType[1] = RESTORE_NONE	
FrameRestorationType[2] = RESTORE_NONE	
UsesLr = 0	
return	
}	
UsesLr = 0	
usesChromaLr = 0	
for ( i = 0; i < NumPlanes; i++ ) {	
lr_type	f(2)
FrameRestorationType[i] = Remap_Lr_Type[lr_type]	
if ( FrameRestorationType[i] != RESTORE_NONE ) {	
UsesLr = 1	
if ( i > 0 ) {	
usesChromaLr = 1	
}	
}	
}	
if ( UsesLr ) {	
if ( use_128x128_superblock ) {	
lr_unit_shift	f(1)
lr_unit_shift++	
} else {	
lr_unit_shift	f(1)
if ( lr_unit_shift ) {	
lr_unit_extra_shift	f(1)
lr_unit_shift += lr_unit_extra_shift	
}	
}	
LoopRestorationSize[ 0 ] = RESTORATION_TILESIZE_MAX >> (2 - lr_unit_shift)	
if ( subsampling_x && subsampling_y && usesChromaLr ) {	
lr_uv_shift	f(1)
} else {	
lr_uv_shift = 0	
}	
LoopRestorationSize[ 1 ] = LoopRestorationSize[ 0 ] >> lr_uv_shift	
LoopRestorationSize[ 2 ] = LoopRestorationSize[ 0 ] >> lr_uv_shift	
}	
}	

where Remap\_Lr\_Type is a constant lookup table specified as:



```
Remap_Lr_Type[4] = {
    RESTORE_NONE, RESTORE_SWITCHABLE, RESTORE_WIENER, RESTORE_SGRPROJ
}
```

## 5.9.21. TX mode syntax

read_tx_mode( ) {	Type
if ( CodedLossless == 1 ) {	
TxMode = ONLY_4X4	
} else {	
tx_mode_select	f(1)
if ( tx_mode_select ) {	
TxMode = TX_MODE_SELECT	
} else {	
TxMode = TX_MODE_LARGEST	
}	
}	
}	

## 5.9.22. Skip mode params syntax

skip_mode_params( ) {	Type
if ( FrameIsIntra    !reference_select    !enable_order_hint ) {	
skipModeAllowed = 0	
} else {	
forwardIdx = -1	
backwardIdx = -1	
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
refHint = RefOrderHint[ ref_frame_idx[ i ] ]	
if ( get_relative_dist( refHint, OrderHint ) < 0 ) {	
if ( forwardIdx < 0	
get_relative_dist( refHint, forwardHint ) > 0 ) {	
forwardIdx = i	
forwardHint = refHint	
}	
} else if ( get_relative_dist( refHint, OrderHint ) > 0 ) {	
if ( backwardIdx < 0	
get_relative_dist( refHint, backwardHint ) < 0 ) {	
backwardIdx = i	
backwardHint = refHint	
}	
}	
}	
}	
if ( forwardIdx < 0 ) {	
skipModeAllowed = 0	
} else if ( backwardIdx >= 0 ) {	
skipModeAllowed = 1	
SkipModeFrame[ 0 ] = LAST_FRAME + Min(forwardIdx, backwardIdx)	

SkipModeFrame[ 1 ] = LAST_FRAME + Max(forwardIdx, backwardIdx)	
} else {	
secondForwardIdx = -1	
for ( i = 0; i < REFS_PER_FRAME; i++ ) {	
refHint = RefOrderHint[ ref_frame_idx[ i ] ]	
if ( get_relative_dist( refHint, forwardHint ) < 0 ) {	
if ( secondForwardIdx < 0	
get_relative_dist( refHint, secondForwardHint ) > 0 ) {	
secondForwardIdx = i	
secondForwardHint = refHint	
}	
}	
}	
if ( secondForwardIdx < 0 ) {	
skipModeAllowed = 0	
} else {	
skipModeAllowed = 1	
SkipModeFrame[ 0 ] = LAST_FRAME + Min(forwardIdx, secondForwardIdx)	
SkipModeFrame[ 1 ] = LAST_FRAME + Max(forwardIdx, secondForwardIdx)	
}	
}	
if ( skipModeAllowed ) {	
skip_mode_present	f(1)
} else {	
skip_mode_present = 0	
}	
}	

### 5.9.23. Frame reference mode syntax

frame_reference_mode( ) {	Type
if ( FrameIsIntra ) {	
reference_select = 0	
} else {	
reference_select	f(1)
}	
}	

### 5.9.24. Global motion params syntax

global_motion_params( ) {	Type
for ( ref = LAST_FRAME; ref <= ALTREF_FRAME; ref++ ) {	
GmType[ ref ] = IDENTITY	
for ( i = 0; i < 6; i++ ) {	
gm_params[ ref ][ i ] = ( ( i % 3 == 2 ) ?	
1 << WARPEDMODEL_PREC_BITS : 0 )	
}	
}	
if ( FrameIsIntra )	

return	
for ( ref = LAST_FRAME; ref <= ALTREF_FRAME; ref++ ) {	
is_global	f(1)
if ( is_global ) {	
is_rot_zoom	f(1)
if ( is_rot_zoom ) {	
type = ROTZOOM	
} else {	
is_translation	f(1)
type = is_translation ? TRANSLATION : AFFINE	
}	
} else {	
type = IDENTITY	
}	
GmType[ref] = type	
if ( type >= ROTZOOM ) {	
read_global_param(type,ref,2)	
read_global_param(type,ref,3)	
if ( type == AFFINE ) {	
read_global_param(type,ref,4)	
read_global_param(type,ref,5)	
} else {	
gm_params[ref][4] = -gm_params[ref][3]	
gm_params[ref][5] = gm_params[ref][2]	
}	
}	
if ( type >= TRANSLATION ) {	
read_global_param(type,ref,0)	
read_global_param(type,ref,1)	
}	
}	
}	

## 5.9.25. Global param syntax

read_global_param( type, ref, idx ) {	Type
absBits = GM_ABS_ALPHA_BITS	
precBits = GM_ALPHA_PREC_BITS	
if ( idx < 2 ) {	
if ( type == TRANSLATION ) {	
absBits = GM_ABS_TRANS_ONLY_BITS - !allow_high_precision_mv	
precBits = GM_TRANS_ONLY_PREC_BITS - !allow_high_precision_mv	
} else {	
absBits = GM_ABS_TRANS_BITS	
precBits = GM_TRANS_PREC_BITS	
}	
}	
precDiff = WARPEDMODEL_PREC_BITS - precBits	

round = (idx % 3) == 2 ? (1 << WARPEDMODEL_PREC_BITS) : 0	
sub = (idx % 3) == 2 ? (1 << precBits) : 0	
mx = (1 << absBits)	
r = (PrevGmParams[ref][idx] >> precDiff) - sub	
gm_params[ref][idx] =	
(decode_signed_subexp_with_ref( -mx, mx + 1, r ) << precDiff) + round	
}	

**Note:** When force\_integer\_mv is equal to 1, some fractional bits are still read for the translation components. However, these fractional bits will be discarded during the Setup Global MV process.

## 5.9.26. Decode signed subexp with ref syntax

decode_signed_subexp_with_ref( low, high, r ) {	Type
x = decode_unsigned_subexp_with_ref(high - low, r - low)	
return x + low	
}	

**Note:** decode\_signed\_subexp\_with\_ref will return a value in the range low to high - 1 (inclusive).

## 5.9.27. Decode unsigned subexp with ref syntax

decode_unsigned_subexp_with_ref( mx, r ) {	Type
v = decode_subexp( mx )	
if ( (r << 1) <= mx ) {	
return inverse_recenter(r, v)	
} else {	
return mx - 1 - inverse_recenter(mx - 1 - r, v)	
}	
}	

**Note:** decode\_unsigned\_subexp\_with\_ref will return a value in the range 0 to mx - 1 (inclusive).

## 5.9.28. Decode subexp syntax

decode_subexp( numSyms ) {	Type
i = 0	
mk = 0	
k = 3	
while ( 1 ) {	
b2 = i ? k + i - 1 : k	
a = 1 << b2	
if ( numSyms <= mk + 3 * a ) {	
subexp_final_bits	ns(numSyms - mk)
return subexp_final_bits + mk	
} else {	

subexp_more_bits	f(1)
if ( subexp_more_bits ) {	
i++	
mk += a	
} else {	
subexp_bits	f(b2)
return subexp_bits + mk	
}	
}	
}	
}	

### 5.9.29. Inverse recenter function

inverse_recenter( r, v ) {	Type
if ( v > 2 * r )	
return v	
else if ( v & 1 )	
return r - ((v + 1) >> 1)	
else	
return r + (v >> 1)	
}	

### 5.9.30. Film grain params syntax

film_grain_params( ) {	Type
if ( !film_grain_params_present	
(!show_frame && !showable_frame) ) {	
reset_grain_params()	
return	
}	
apply_grain	f(1)
if ( !apply_grain ) {	
reset_grain_params()	
return	
}	
grain_seed	f(16)
if ( frame_type == INTER_FRAME )	
update_grain	f(1)
else	
update_grain = 1	
if ( !update_grain ) {	
film_grain_params_ref_idx	f(3)
tempGrainSeed = grain_seed	
load_grain_params( film_grain_params_ref_idx )	
grain_seed = tempGrainSeed	
return	
}	
num_y_points	f(4)
for ( i = 0; i < num_y_points; i++ ) {	

point_y_value[ i ]	f(8)
point_y_scaling[ i ]	f(8)
}	
if ( mono_chrome ) {	
chroma_scaling_from_luma = 0	
} else {	
chroma_scaling_from_luma	f(1)
}	
if ( mono_chrome    chroma_scaling_from_luma	
( subsampling_x == 1 && subsampling_y == 1 &&	
num_y_points == 0 )	
) {	
num_cb_points = 0	
num_cr_points = 0	
} else {	
num_cb_points	f(4)
for ( i = 0; i < num_cb_points; i++ ) {	
point_cb_value[ i ]	f(8)
point_cb_scaling[ i ]	f(8)
}	
num_cr_points	f(4)
for ( i = 0; i < num_cr_points; i++ ) {	
point_cr_value[ i ]	f(8)
point_cr_scaling[ i ]	f(8)
}	
}	
grain_scaling_minus_8	f(2)
ar_coeff_lag	f(2)
numPosLuma = 2 * ar_coeff_lag * ( ar_coeff_lag + 1 )	
if ( num_y_points ) {	
numPosChroma = numPosLuma + 1	
for ( i = 0; i < numPosLuma; i++ )	
ar_coeffs_y_plus_128[ i ]	f(8)
} else {	
numPosChroma = numPosLuma	
}	
if ( chroma_scaling_from_luma    num_cb_points ) {	
for ( i = 0; i < numPosChroma; i++ )	
ar_coeffs_cb_plus_128[ i ]	f(8)
}	
if ( chroma_scaling_from_luma    num_cr_points ) {	
for ( i = 0; i < numPosChroma; i++ )	
ar_coeffs_cr_plus_128[ i ]	f(8)
}	
ar_coeff_shift_minus_6	f(2)
grain_scale_shift	f(2)
if ( num_cb_points ) {	
cb_mult	f(8)
cb_luma_mult	f(8)

cb_offset	f(9)
}	
if ( num_cr_points ) {	
cr_mult	f(8)
cr_luma_mult	f(8)
cr_offset	f(9)
}	
overlap_flag	f(1)
clip_to_restricted_range	f(1)
}	

### 5.9.31. Temporal point info syntax

temporal_point_info( ) {	Type
n = frame_presentation_time_length_minus_1 + 1	
frame_presentation_time	f(n)
}	

## 5.10. Frame OBU syntax

frame_obu( sz ) {	Type
startBitPos = get_position( )	
frame_header_obu( )	
byte_alignment( )	
endBitPos = get_position( )	
headerBytes = (endBitPos - startBitPos) / 8	
sz -= headerBytes	
tile_group_obu( sz )	
}	

## 5.11. Tile group OBU syntax

### 5.11.1. General tile group OBU syntax

tile_group_obu( sz ) {	Type
NumTiles = TileCols * TileRows	
startBitPos = get_position( )	
tile_start_and_end_present_flag = 0	
if ( NumTiles > 1 )	
tile_start_and_end_present_flag	f(1)
if ( NumTiles == 1    !tile_start_and_end_present_flag ) {	
tg_start = 0	
tg_end = NumTiles - 1	
} else {	
tileBits = TileColsLog2 + TileRowsLog2	
tg_start	f(tileBits)
tg_end	f(tileBits)
}	
byte_alignment( )	

endBitPos = get_position( )	
headerBytes = (endBitPos - startBitPos) / 8	
sz -= headerBytes	
for ( TileNum = tg_start; TileNum <= tg_end; TileNum++ ) {	
tileRow = TileNum / TileCols	
tileCol = TileNum % TileCols	
lastTile = TileNum == tg_end	
if ( lastTile ) {	
tileSize = sz	
} else {	
tile_size_minus_1	le(TileSizeBytes)
tileSize = tile_size_minus_1 + 1	
sz -= tileSize + TileSizeBytes	
}	
MiRowStart = MiRowStarts[ tileRow ]	
MiRowEnd = MiRowStarts[ tileRow + 1 ]	
MiColStart = MiColStarts[ tileCol ]	
MiColEnd = MiColStarts[ tileCol + 1 ]	
CurrentQIndex = base_q_idx	
init_symbol( tileSize )	
decode_tile( )	
exit_symbol( )	
}	
if ( tg_end == NumTiles - 1 ) {	
if ( !disable_frame_end_update_cdf ) {	
frame_end_update_cdf( )	
}	
decode_frame_wrapup( )	
SeenFrameHeader = 0	
}	
}	

## 5.11.2. Decode tile syntax

decode_tile( ) {	Type
clear_above_context( )	
for ( i = 0; i < FRAME_LF_COUNT; i++ )	
DeltaLF[ i ] = 0	
for ( plane = 0; plane < NumPlanes; plane++ ) {	
for ( pass = 0; pass < 2; pass++ ) {	
RefSgrXqd[ plane ][ pass ] = Sgrproj_Xqd_Mid[ pass ]	
for ( i = 0; i < WIENER_COEFFS; i++ ) {	
RefLrWiener[ plane ][ pass ][ i ] = Wiener_Taps_Mid[ i ]	
}	
}	
}	
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64	
sbSize4 = Num_4x4_Blocks_Wide[ sbSize ]	



for ( r = MiRowStart; r < MiRowEnd; r += sbSize4 ) {	
clear_left_context( )	
for ( c = MiColStart; c < MiColEnd; c += sbSize4 ) {	
ReadDeltas = delta_q_present	
clear_cdef( r, c )	
clear_block_decoded_flags( r, c, sbSize4 )	
read_lr( r, c, sbSize )	
decode_partition( r, c, sbSize )	
}	
}	
}	

where Sgrproj\_Xqd\_Mid and Wiener\_Taps\_Mid are constant lookup tables specified as:

```

Wiener_Taps_Mid[3] = { 3, -7, 15 }

Sgrproj_Xqd_Mid[2] = { -32, 31 }

```

### 5.11.3. Clear block decoded flags function

	Type
clear_block_decoded_flags( r, c, sbSize4 ) {	
for ( plane = 0; plane < NumPlanes; plane++ ) {	
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
sbWidth4 = ( MiColEnd - c ) >> subX	
sbHeight4 = ( MiRowEnd - r ) >> subY	
for ( y = -1; y <= ( sbSize4 >> subY ); y++ )	
for ( x = -1; x <= ( sbSize4 >> subX ); x++ ) {	
if ( y < 0 && x < sbWidth4 )	
BlockDecoded[ plane ][ y ][ x ] = 1	
else if ( x < 0 && y < sbHeight4 )	
BlockDecoded[ plane ][ y ][ x ] = 1	
else	
BlockDecoded[ plane ][ y ][ x ] = 0	
}	
BlockDecoded[ plane ][ sbSize4 >> subY ][ -1 ] = 0	
}	
}	

### 5.11.4. Decode partition syntax

	Type
decode_partition( r, c, bSize ) {	
if ( r >= MiRows    c >= MiCols )	
return 0	
AvailU = is_inside( r - 1, c )	
AvailL = is_inside( r, c - 1 )	
num4x4 = Num_4x4_Blocks_Wide[ bSize ]	
halfBlock4x4 = num4x4 >> 1	

quarterBlock4x4 = halfBlock4x4 >> 1	
hasRows = ( r + halfBlock4x4 ) < MiRows	
hasCols = ( c + halfBlock4x4 ) < MiCols	
if ( bSize < BLOCK_8X8 ) {	
partition = PARTITION_NONE	
} else if ( hasRows && hasCols ) {	
<b>partition</b>	S()
} else if ( hasCols ) {	
<b>split_or_horz</b>	S()
partition = split_or_horz ? PARTITION_SPLIT : PARTITION_HORZ	
} else if ( hasRows ) {	
<b>split_or_vert</b>	S()
partition = split_or_vert ? PARTITION_SPLIT : PARTITION_VERT	
} else {	
partition = PARTITION_SPLIT	
}	
subSize = Partition_Subsize[ partition ][ bSize ]	
splitSize = Partition_Subsize[ PARTITION_SPLIT ][ bSize ]	
if ( partition == PARTITION_NONE ) {	
decode_block( r, c, subSize )	
} else if ( partition == PARTITION_HORZ ) {	
decode_block( r, c, subSize )	
if ( hasRows )	
decode_block( r + halfBlock4x4, c, subSize )	
} else if ( partition == PARTITION_VERT ) {	
decode_block( r, c, subSize )	
if ( hasCols )	
decode_block( r, c + halfBlock4x4, subSize )	
} else if ( partition == PARTITION_SPLIT ) {	
decode_partition( r, c, subSize )	
decode_partition( r, c + halfBlock4x4, subSize )	
decode_partition( r + halfBlock4x4, c, subSize )	
decode_partition( r + halfBlock4x4, c + halfBlock4x4, subSize )	
} else if ( partition == PARTITION_HORZ_A ) {	
decode_block( r, c, splitSize )	
decode_block( r, c + halfBlock4x4, splitSize )	
decode_block( r + halfBlock4x4, c, subSize )	
} else if ( partition == PARTITION_HORZ_B ) {	
decode_block( r, c, subSize )	
decode_block( r + halfBlock4x4, c, splitSize )	
decode_block( r + halfBlock4x4, c + halfBlock4x4, splitSize )	
} else if ( partition == PARTITION_VERT_A ) {	
decode_block( r, c, splitSize )	
decode_block( r + halfBlock4x4, c, splitSize )	
decode_block( r, c + halfBlock4x4, subSize )	
} else if ( partition == PARTITION_VERT_B ) {	
decode_block( r, c, subSize )	
decode_block( r, c + halfBlock4x4, splitSize )	
decode_block( r + halfBlock4x4, c + halfBlock4x4, splitSize )	

<code>} else if ( partition == PARTITION_HORZ_4 ) {</code>	
<code>    decode_block( r + quarterBlock4x4 * 0, c, subSize )</code>	
<code>    decode_block( r + quarterBlock4x4 * 1, c, subSize )</code>	
<code>    decode_block( r + quarterBlock4x4 * 2, c, subSize )</code>	
<code>    if ( r + quarterBlock4x4 * 3 &lt; MiRows )</code>	
<code>        decode_block( r + quarterBlock4x4 * 3, c, subSize )</code>	
<code>} else {</code>	
<code>    decode_block( r, c + quarterBlock4x4 * 0, subSize )</code>	
<code>    decode_block( r, c + quarterBlock4x4 * 1, subSize )</code>	
<code>    decode_block( r, c + quarterBlock4x4 * 2, subSize )</code>	
<code>    if ( c + quarterBlock4x4 * 3 &lt; MiCols )</code>	
<code>        decode_block( r, c + quarterBlock4x4 * 3, subSize )</code>	
<code>}</code>	
<code>}</code>	

### 5.11.5. Decode block syntax

decode_block( r, c, subSize ) {	Type
MiRow = r	
MiCol = c	
MiSize = subSize	
bw4 = Num_4x4_Blocks_Wide[ subSize ]	
bh4 = Num_4x4_Blocks_High[ subSize ]	
if ( bh4 == 1 && subsampling_y && (MiRow & 1) == 0 )	
HasChroma = 0	
else if ( bw4 == 1 && subsampling_x && (MiCol & 1) == 0 )	
HasChroma = 0	
else	
HasChroma = NumPlanes > 1	
AvailU = is_inside( r - 1, c )	
AvailL = is_inside( r, c - 1 )	
AvailUChroma = AvailU	
AvailLChroma = AvailL	
if ( HasChroma ) {	
if ( subsampling_y && bh4 == 1 )	
AvailUChroma = is_inside( r - 2, c )	
if ( subsampling_x && bw4 == 1 )	
AvailLChroma = is_inside( r, c - 2 )	
} else {	
AvailUChroma = 0	
AvailLChroma = 0	
}	
mode_info( )	
palette_tokens( )	
read_block_tx_size( )	
if ( skip )	
reset_block_context( bw4, bh4 )	
isCompound = RefFrame[ 1 ] > INTRA_FRAME	

for ( y = 0; y < bh4; y++ ) {	
for ( x = 0; x < bw4; x++ ) {	
YModes [ r + y ][ c + x ] = YMode	
if ( RefFrame[ 0 ] == INTRA_FRAME && HasChroma )	
UVModes [ r + y ][ c + x ] = UVMode	
for ( refList = 0; refList < 2; refList++ )	
RefFrames[ r + y ][ c + x ][ refList ] = RefFrame[ refList ]	
if ( is_inter ) {	
if ( !use_intrabc ) {	
CompGroupIdxs[ r + y ][ c + x ] = comp_group_idx	
CompoundIdxs[ r + y ][ c + x ] = compound_idx	
}	
for ( dir = 0; dir < 2; dir++ ) {	
InterpFilters[ r + y ][ c + x ][ dir ] = interp_filter[ dir ]	
}	
for ( refList = 0; refList < 1 + isCompound; refList++ ) {	
Mvs[ r + y ][ c + x ][ refList ] = Mv[ refList ]	
}	
}	
}	
}	
compute_prediction( )	
residual( )	
for ( y = 0; y < bh4; y++ ) {	
for ( x = 0; x < bw4; x++ ) {	
IsInters[ r + y ][ c + x ] = is_inter	
SkipModes[ r + y ][ c + x ] = skip_mode	
Skips[ r + y ][ c + x ] = skip	
TxSizes[ r + y ][ c + x ] = TxSize	
MiSizes[ r + y ][ c + x ] = MiSize	
SegmentIds[ r + y ][ c + x ] = segment_id	
PaletteSizes[ 0 ][ r + y ][ c + x ] = PaletteSizeY	
PaletteSizes[ 1 ][ r + y ][ c + x ] = PaletteSizeUV	
for ( i = 0; i < PaletteSizeY; i++ )	
PaletteColors[ 0 ][ r + y ][ c + x ][ i ] = palette_colors_y[ i ]	
for ( i = 0; i < PaletteSizeUV; i++ )	
PaletteColors[ 1 ][ r + y ][ c + x ][ i ] = palette_colors_u[ i ]	
for ( i = 0; i < FRAME_LF_COUNT; i++ )	
DeltaLFs[ r + y ][ c + x ][ i ] = DeltaLF[ i ]	
}	
}	
}	

where reset\_block\_context( ) is specified as:

```

reset_block_context( bw4, bh4 ) {
    for ( plane = 0; plane < 1 + 2 * HasChroma; plane++ ) {
        subX = (plane > 0) ? subsampling_x : 0
        subY = (plane > 0) ? subsampling_y : 0
        for ( i = MiCol >> subX; i < ( ( MiCol + bw4 ) >> subX ); i++ ) {
            AboveLevelContext[ plane ][ i ] = 0
            AboveDcContext[ plane ][ i ] = 0
        }
        for ( i = MiRow >> subY; i < ( ( MiRow + bh4 ) >> subY ); i++ ) {
            LeftLevelContext[ plane ][ i ] = 0
            LeftDcContext[ plane ][ i ] = 0
        }
    }
}

```

## 5.11.6. Mode info syntax

mode_info( ) {	Type
if ( FrameIsIntra )	
intra_frame_mode_info( )	
else	
inter_frame_mode_info( )	
}	

## 5.11.7. Intra frame mode info syntax

intra_frame_mode_info( ) {	Type
skip = 0	
if ( SegIdPreSkip )	
intra_segment_id( )	
skip_mode = 0	
read_skip( )	
if ( !SegIdPreSkip )	
intra_segment_id( )	
read_cdef( )	
read_delta_qindex( )	
read_delta_lf( )	
ReadDeltas = 0	
RefFrame[ 0 ] = INTRA_FRAME	
RefFrame[ 1 ] = NONE	
if ( allow_intrabc ) {	
use_intrabc	S()
} else {	
use_intrabc = 0	
}	
if ( use_intrabc ) {	
is_inter = 1	
YMode = DC_PRED	

UVMode = DC_PRED	
motion_mode = SIMPLE	
compound_type = COMPOUND_AVERAGE	
PaletteSizeY = 0	
PaletteSizeUV = 0	
interp_filter[ 0 ] = BILINEAR	
interp_filter[ 1 ] = BILINEAR	
find_mv_stack( 0 )	
assign_mv( 0 )	
} else {	
is_inter = 0	
intra_frame_y_mode	S()
YMode = intra_frame_y_mode	
intra_angle_info_y( )	
if ( HasChroma ) {	
uv_mode	S()
UVMode = uv_mode	
if ( UVMode == UV_CFL_PRED ) {	
read_cfl_alphas( )	
}	
intra_angle_info_uv( )	
}	
PaletteSizeY = 0	
PaletteSizeUV = 0	
if ( MiSize >= BLOCK_8X8 &&	
Block_Width[ MiSize ] <= 64 &&	
Block_Height[ MiSize ] <= 64 &&	
allow_screen_content_tools ) {	
palette_mode_info( )	
}	
filter_intra_mode_info( )	
}	
}	

### 5.11.8. Intra segment ID syntax

intra_segment_id( ) {	Type
if ( segmentation_enabled )	
read_segment_id( )	
else	
segment_id = 0	
Lossless = LosslessArray[ segment_id ]	
}	

### 5.11.9. Read segment ID syntax

read_segment_id( ) {	Type
if ( AvailU && AvailL )	
prevUL = SegmentIds[ MiRow - 1 ][ MiCol - 1 ]	
else	

prevUL = -1	
if ( AvailU )	
prevU = SegmentIds[ MiRow - 1 ][ MiCol ]	
else	
prevU = -1	
if ( AvailL )	
prevL = SegmentIds[ MiRow ][ MiCol - 1 ]	
else	
prevL = -1	
if ( prevU == -1 )	
pred = (prevL == -1) ? 0 : prevL	
else if ( prevL == -1 )	
pred = prevU	
else	
pred = (prevUL == prevU) ? prevU : prevL	
if ( skip ) {	
segment_id = pred	
} else {	
segment_id	S()
segment_id = neg_deinterleave( segment_id, pred,	
LastActiveSegId + 1 )	
}	
}	

where neg\_deinterleave is a function defined as:

```

neg_deinterleave(diff, ref, max) {
    if ( !ref )
        return diff
    if ( ref >= (max - 1) )
        return max - diff - 1
    if ( 2 * ref < max ) {
        if ( diff <= 2 * ref ) {
            if ( diff & 1 )
                return ref + ((diff + 1) >> 1)
            else
                return ref - (diff >> 1)
        }
        return diff
    } else {
        if ( diff <= 2 * (max - ref - 1) ) {
            if ( diff & 1 )
                return ref + ((diff + 1) >> 1)
            else
                return ref - (diff >> 1)
        }
        return max - (diff + 1)
    }
}

```

## 5.11.10. Skip mode syntax

read_skip_mode() {	Type
if ( seg_feature_active( SEG_LVL_SKIP )	
seg_feature_active( SEG_LVL_REF_FRAME )	
seg_feature_active( SEG_LVL_GLOBALMV )	
!skip_mode_present	
Block_Width[ MiSize ] < 8	
Block_Height[ MiSize ] < 8 ) {	
skip_mode = 0	
} else {	
skip_mode	S()
}	
}	

## 5.11.11. Skip syntax

read_skip() {	Type
if ( SegIdPreSkip && seg_feature_active( SEG_LVL_SKIP ) ) {	
skip = 1	
} else {	
skip	S()
}	
}	

## 5.11.12. Quantizer index delta syntax

read_delta_qindex( ) {	Type
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64	
if ( MiSize == sbSize && skip )	
return	
if ( ReadDeltas ) {	
delta_q_abs	S()
if ( delta_q_abs == DELTA_Q_SMALL ) {	
delta_q_rem_bits	L(3)
delta_q_rem_bits++	
delta_q_abs_bits	L(delta_q_rem_bits)
delta_q_abs = delta_q_abs_bits + (1 << delta_q_rem_bits) + 1	
}	
if ( delta_q_abs ) {	
delta_q_sign_bit	L(1)
reducedDeltaQIndex = delta_q_sign_bit ? -delta_q_abs : delta_q_abs	
CurrentQIndex = Clip3(1, 255,	
CurrentQIndex + (reducedDeltaQIndex << delta_q_res))	
}	
}	
}	



### 5.11.13. Loop filter delta syntax

read_delta_lf( ) {	Type
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64	
if ( MiSize == sbSize && skip )	
return	
if ( ReadDeltas && delta_lf_present ) {	
frameLfCount = 1	
if ( delta_lf_multi ) {	
frameLfCount = ( NumPlanes > 1 ) ? FRAME_LF_COUNT : ( FRAME_LF_COUNT - 2 )	
}	
for ( i = 0; i < frameLfCount; i++ ) {	
delta_lf_abs	S()
if ( delta_lf_abs == DELTA_LF_SMALL ) {	
delta_lf_rem_bits	L(3)
n = delta_lf_rem_bits + 1	
delta_lf_abs_bits	L(n)
deltaLfAbs = delta_lf_abs_bits +	
( 1 << n ) + 1	
} else {	
deltaLfAbs = delta_lf_abs	
}	
if ( deltaLfAbs ) {	
delta_lf_sign_bit	L(1)
reducedDeltaLfLevel = delta_lf_sign_bit ?	
-deltaLfAbs :	
deltaLfAbs	
DeltaLF[ i ] = Clip3( -MAX_LOOP_FILTER, MAX_LOOP_FILTER, DeltaLF[ i ] +	
(reducedDeltaLfLevel << delta_lf_res) )	
}	
}	
}	
}	

### 5.11.14. Segmentation feature active function

seg_feature_active_idx( idx, feature ) {	Type
return segmentation_enabled && FeatureEnabled[ idx ][ feature ]	
}	
seg_feature_active( feature ) {	
return seg_feature_active_idx( segment_id, feature )	
}	

### 5.11.15. TX size syntax

read_tx_size( allowSelect ) {	Type
if ( Lossless ) {	
TxSize = TX_4X4	
return	

}	
maxRectTxSize = Max_Tx_Size_Rect[ MiSize ]	
maxTxDepth = Max_Tx_Depth[ MiSize ]	
TxSize = maxRectTxSize	
if ( MiSize > BLOCK_4X4 && allowSelect && TxMode == TX_MODE_SELECT ) {	
tx_depth	S()
for ( i = 0; i < tx_depth; i++ )	
TxSize = Split_Tx_Size[ TxSize ]	
}	
}	
}	

The Max\_Tx\_Depth table specifies the maximum transform depth which can be encoded for each block size:

```
Max_Tx_Depth[ BLOCK_SIZES ] = {
    0, 1, 1, 1,
    2, 2, 2, 3,
    3, 3, 4, 4,
    4, 4, 4, 4,
    2, 2, 3, 3,
    4, 4
}
```

## 5.11.16. Block TX size syntax

read_block_tx_size( ) {	Type
bw4 = Num_4x4_Blocks_Wide[ MiSize ]	
bh4 = Num_4x4_Blocks_High[ MiSize ]	
if ( TxMode == TX_MODE_SELECT &&	
MiSize > BLOCK_4X4 && is_inter &&	
!skip && !lossless ) {	
maxTxSz = Max_Tx_Size_Rect[ MiSize ]	
txW4 = Tx_Width[ maxTxSz ] / MI_SIZE	
txH4 = Tx_Height[ maxTxSz ] / MI_SIZE	
for ( row = MiRow; row < MiRow + bh4; row += txH4 )	
for ( col = MiCol; col < MiCol + bw4; col += txW4 )	
read_var_tx_size( row, col, maxTxSz, 0 )	
} else {	
read_tx_size(!skip    !is_inter)	
for ( row = MiRow; row < MiRow + bh4; row++ )	
for ( col = MiCol; col < MiCol + bw4; col++ )	
InterTxSizes[ row ][ col ] = TxSize	
}	
}	

## 5.11.17. Var TX size syntax

read\_var\_tx\_size is used to read a transform size tree.

read_var_tx_size( row, col, txSz, depth) {	Type
if ( row >= MiRows    col >= MiCols )	
return	
if ( txSz == TX_4X4    depth == MAX_VARTX_DEPTH ) {	
txfm_split = 0	
} else {	
txfm_split	S()
}	
w4 = Tx_Width[ txSz ] / MI_SIZE	
h4 = Tx_Height[ txSz ] / MI_SIZE	
if ( txfm_split ) {	
subTxSz = Split_Tx_Size[ txSz ]	
stepW = Tx_Width[ subTxSz ] / MI_SIZE	
stepH = Tx_Height[ subTxSz ] / MI_SIZE	
for ( i = 0; i < h4; i += stepH )	
for ( j = 0; j < w4; j += stepW )	
read_var_tx_size( row + i, col + j, subTxSz, depth+1)	
} else {	
for ( i = 0; i < h4; i++ )	
for ( j = 0; j < w4; j++ )	
InterTxSizes[ row + i ][ col + j ] = txSz	
TxSize = txSz	
}	
}	

## 5.11.18. Inter frame mode info syntax

inter_frame_mode_info( ) {	Type
use_intrabc = 0	
LeftRefFrame[ 0 ] = AvailL ? RefFrames[ MiRow ][ MiCol-1 ][ 0 ] : INTRA_FRAME	
AboveRefFrame[ 0 ] = AvailU ? RefFrames[ MiRow-1 ][ MiCol ][ 0 ] : INTRA_FRAME	
LeftRefFrame[ 1 ] = AvailL ? RefFrames[ MiRow ][ MiCol-1 ][ 1 ] : NONE	
AboveRefFrame[ 1 ] = AvailU ? RefFrames[ MiRow-1 ][ MiCol ][ 1 ] : NONE	
LeftIntra = LeftRefFrame[ 0 ] <= INTRA_FRAME	
AboveIntra = AboveRefFrame[ 0 ] <= INTRA_FRAME	
LeftSingle = LeftRefFrame[ 1 ] <= INTRA_FRAME	
AboveSingle = AboveRefFrame[ 1 ] <= INTRA_FRAME	
skip = 0	
inter_segment_id( 1 )	
read_skip_mode( )	
if ( skip_mode )	
skip = 1	
else	
read_skip( )	
if ( !SegIdPreSkip )	
inter_segment_id( 0 )	
Lossless = LosslessArray[ segment_id ]	
read_cdef( )	
read_delta_qindex( )	

read_delta_lf( )	
ReadDeltas = 0	
read_is_inter( )	
if ( is_inter )	
inter_block_mode_info( )	
else	
intra_block_mode_info( )	
}	

## 5.11.19. Inter segment ID syntax

This is called before (preSkip equal to 1) and after (preSkip equal to 0) the skip syntax element has been read.

inter_segment_id( preSkip ) {	Type
if ( segmentation_enabled ) {	
predictedSegmentId = get_segment_id( )	
if ( segmentation_update_map ) {	
if ( preSkip && !SegIdPreSkip ) {	
segment_id = 0	
return	
}	
if ( !preSkip ) {	
if ( skip ) {	
seg_id_predicted = 0	
for ( i = 0; i < Num_4x4_Blocks_Wide[ MiSize ]; i++ )	
AboveSegPredContext[ MiCol + i ] = seg_id_predicted	
for ( i = 0; i < Num_4x4_Blocks_High[ MiSize ]; i++ )	
LeftSegPredContext[ MiRow + i ] = seg_id_predicted	
read_segment_id( )	
return	
}	
}	
if ( segmentation_temporal_update == 1 ) {	
seg_id_predicted	S()
if ( seg_id_predicted )	
segment_id = predictedSegmentId	
else	
read_segment_id( )	
for ( i = 0; i < Num_4x4_Blocks_Wide[ MiSize ]; i++ )	
AboveSegPredContext[ MiCol + i ] = seg_id_predicted	
for ( i = 0; i < Num_4x4_Blocks_High[ MiSize ]; i++ )	
LeftSegPredContext[ MiRow + i ] = seg_id_predicted	
} else {	
read_segment_id( )	
}	
} else {	
segment_id = predictedSegmentId	
}	
} else {	

segment_id = 0	
}	
}	

## 5.11.20. Is inter syntax

read_is_inter( ) {	Type
if ( skip_mode ) {	
is_inter = 1	
} else if ( seg_feature_active ( SEG_LVL_REF_FRAME ) ) {	
is_inter = FeatureData[ segment_id ][ SEG_LVL_REF_FRAME ] != INTRA_FRAME	
} else if ( seg_feature_active ( SEG_LVL_GLOBALMV ) ) {	
is_inter = 1	
} else {	
is_inter	S()
}	
}	

## 5.11.21. Get segment ID function

The predicted segment id is the smallest value found in the on-screen region of the segmentation map covered by the current block.

get_segment_id( ) {	Type
bw4 = Num_4x4_Blocks_Wide[ MiSize ]	
bh4 = Num_4x4_Blocks_High[ MiSize ]	
xMis = Min( MiCols - MiCol, bw4 )	
yMis = Min( MiRows - MiRow, bh4 )	
seg = 7	
for ( y = 0; y < yMis; y++ )	
for ( x = 0; x < xMis; x++ )	
seg = Min( seg, PrevSegmentIds[ MiRow + y ][ MiCol + x ] )	
return seg	
}	

## 5.11.22. Intra block mode info syntax

intra_block_mode_info( ) {	Type
RefFrame[ 0 ] = INTRA_FRAME	
RefFrame[ 1 ] = NONE	
y_mode	S()
YMode = y_mode	
intra_angle_info_y( )	
if ( HasChroma ) {	
uv_mode	S()
UVMode = uv_mode	
if ( UVMode == UV_CFL_PRED ) {	
read_cfl_alphas( )	
}	
intra_angle_info_uv( )	

}	
PaletteSizeY = 0	
PaletteSizeUV = 0	
if ( MiSize >= BLOCK_8X8 &&	
Block_Width[ MiSize ] <= 64 &&	
Block_Height[ MiSize ] <= 64 &&	
allow_screen_content_tools )	
palette_mode_info( )	
filter_intra_mode_info( )	
}	

### 5.11.23. Inter block mode info syntax

inter_block_mode_info( ) {	Type
PaletteSizeY = 0	
PaletteSizeUV = 0	
read_ref_frames( )	
isCompound = RefFrame[ 1 ] > INTRA_FRAME	
find_mv_stack( isCompound )	
if ( skip_mode ) {	
YMode = NEAREST_NEARESTMV	
} else if ( seg_feature_active( SEG_LVL_SKIP )	
seg_feature_active( SEG_LVL_GLOBALMV ) ) {	
YMode = GLOBALMV	
} else if ( isCompound ) {	
compound_mode	S()
YMode = NEAREST_NEARESTMV + compound_mode	
} else {	
new_mv	S()
if ( new_mv == 0 ) {	
YMode = NEWMV	
} else {	
zero_mv	S()
if ( zero_mv == 0 ) {	
YMode = GLOBALMV	
} else {	
ref_mv	S()
YMode = (ref_mv == 0) ? NEARESTMV : NEARMV	
}	
}	
}	
}	
RefMvIdx = 0	
if ( YMode == NEWMV    YMode == NEW_NEWMV ) {	
for ( idx = 0; idx < 2; idx++ ) {	
if ( NumMvFound > idx + 1 ) {	
drl_mode	S()
if ( drl_mode == 0 ) {	
RefMvIdx = idx	
break	

}	
RefMvIdx = idx + 1	
}	
}	
} else if ( has_nearmv( ) ) {	
RefMvIdx = 1	
for ( idx = 1; idx < 3; idx++ ) {	
if ( NumMvFound > idx + 1 ) {	
drl_mode	S()
if ( drl_mode == 0 ) {	
RefMvIdx = idx	
break	
}	
RefMvIdx = idx + 1	
}	
}	
}	
assign_mv( isCompound )	
read_interintra_mode( isCompound )	
read_motion_mode( isCompound )	
read_compound_type( isCompound )	
if ( interpolation_filter == SWITCHABLE ) {	
for ( dir = 0; dir < ( enable_dual_filter ? 2 : 1 ); dir++ ) {	
if ( needs_interp_filter( ) ) {	
interp_filter[ dir ]	S()
} else {	
interp_filter[ dir ] = EIGHTTAP	
}	
}	
if ( !enable_dual_filter )	
interp_filter[ 1 ] = interp_filter[ 0 ]	
} else {	
for ( dir = 0; dir < 2; dir++ )	
interp_filter[ dir ] = interpolation_filter	
}	
}	

The function has\_nearmv is defined as:

```
has_nearmv( ) {
    return (YMode == NEARMV || YMode == NEAR_NEARMV
           || YMode == NEAR_NEWMV || YMode == NEW_NEARMV)
}
```

The function needs\_interp\_filter is defined as:

```

needs_interp_filter( ) {
    large = (Min(Block_Width[MiSize], Block_Height[MiSize]) >= 8)
    if ( skip_mode || motion_mode == LOCALWARP ) {
        return 0
    } else if ( large && YMode == GLOBALMV ) {
        return GmType[ RefFrame[ 0 ] ] == TRANSLATION
    } else if ( large && YMode == GLOBAL_GLOBALMV ) {
        return GmType[ RefFrame[ 0 ] ] == TRANSLATION || GmType[ RefFrame[ 1 ] ] == TRANSLATION
    } else {
        return 1
    }
}

```

## 5.11.24. Filter intra mode info syntax

filter_intra_mode_info( ) {	Type
use_filter_intra = 0	
if ( enable_filter_intra &&	
YMode == DC_PRED && PaletteSizeY == 0 &&	
Max( Block_Width[ MiSize ], Block_Height[ MiSize ] ) <= 32 ) {	
use_filter_intra	S()
if ( use_filter_intra ) {	
filter_intra_mode	S()
}	
}	
}	

## 5.11.25. Ref frames syntax

read_ref_frames( ) {	Type
if ( skip_mode ) {	
RefFrame[ 0 ] = SkipModeFrame[ 0 ]	
RefFrame[ 1 ] = SkipModeFrame[ 1 ]	
} else if ( seg_feature_active( SEG_LVL_REF_FRAME ) ) {	
RefFrame[ 0 ] = FeatureData[ segment_id ][ SEG_LVL_REF_FRAME ]	
RefFrame[ 1 ] = NONE	
} else if ( seg_feature_active( SEG_LVL_SKIP )	
seg_feature_active( SEG_LVL_GLOBALMV ) ) {	
RefFrame[ 0 ] = LAST_FRAME	
RefFrame[ 1 ] = NONE	
} else {	
bw4 = Num_4x4_Blocks_Wide[ MiSize ]	
bh4 = Num_4x4_Blocks_High[ MiSize ]	
if ( reference_select && ( Min( bw4, bh4 ) >= 2 ) )	
comp_mode	S()
else	
comp_mode = SINGLE_REFERENCE	
if ( comp_mode == COMPOUND_REFERENCE ) {	



comp_ref_type	S()
if ( comp_ref_type == UNIDIR_COMP_REFERENCE ) {	
uni_comp_ref	S()
if ( uni_comp_ref ) {	
RefFrame[0] = BWDREF_FRAME	
RefFrame[1] = ALTREF_FRAME	
} else {	
uni_comp_ref_p1	S()
if ( uni_comp_ref_p1 ) {	
uni_comp_ref_p2	S()
if ( uni_comp_ref_p2 ) {	
RefFrame[0] = LAST_FRAME	
RefFrame[1] = GOLDEN_FRAME	
} else {	
RefFrame[0] = LAST_FRAME	
RefFrame[1] = LAST3_FRAME	
}	
} else {	
RefFrame[0] = LAST_FRAME	
RefFrame[1] = LAST2_FRAME	
}	
}	
} else {	
comp_ref	S()
if ( comp_ref == 0 ) {	
comp_ref_p1	S()
RefFrame[ 0 ] = comp_ref_p1 ?	
LAST2_FRAME : LAST_FRAME	
} else {	
comp_ref_p2	S()
RefFrame[ 0 ] = comp_ref_p2 ?	
GOLDEN_FRAME : LAST3_FRAME	
}	
comp_bwdref	S()
if ( comp_bwdref == 0 ) {	
comp_bwdref_p1	S()
RefFrame[ 1 ] = comp_bwdref_p1 ?	
ALTREF2_FRAME : BWDREF_FRAME	
} else {	
RefFrame[ 1 ] = ALTREF_FRAME	
}	
}	
} else {	
single_ref_p1	S()
if ( single_ref_p1 ) {	
single_ref_p2	S()
if ( single_ref_p2 == 0 ) {	
single_ref_p6	S()
RefFrame[ 0 ] = single_ref_p6 ?	

ALTREF2_FRAME : BWDREF_FRAME	
} else {	
RefFrame[ 0 ] = ALTREF_FRAME	
}	
} else {	
single_ref_p3	S()
if ( single_ref_p3 ) {	
single_ref_p5	S()
RefFrame[ 0 ] = single_ref_p5 ?	
GOLDEN_FRAME : LAST3_FRAME	
} else {	
single_ref_p4	S()
RefFrame[ 0 ] = single_ref_p4 ?	
LAST2_FRAME : LAST_FRAME	
}	
}	
RefFrame[ 1 ] = NONE	
}	
}	
}	

## 5.11.26. Assign MV syntax

assign_mv( isCompound ) {	Type
for ( i = 0; i < 1 + isCompound; i++ ) {	
if ( use_intrabc ) {	
compMode = NEWMV	
} else {	
compMode = get_mode( i )	
}	
if ( use_intrabc ) {	
PredMv[ 0 ] = RefStackMv[ 0 ][ 0 ]	
if ( PredMv[ 0 ][ 0 ] == 0 && PredMv[ 0 ][ 1 ] == 0 ) {	
PredMv[ 0 ] = RefStackMv[ 1 ][ 0 ]	
}	
if ( PredMv[ 0 ][ 0 ] == 0 && PredMv[ 0 ][ 1 ] == 0 ) {	
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64	
sbSize4 = Num_4x4_Blocks_High[ sbSize ]	
if ( MiRow - sbSize4 < MiRowStart ) {	
PredMv[ 0 ][ 0 ] = 0	
PredMv[ 0 ][ 1 ] = -(sbSize4 * MI_SIZE + INTRABC_DELAY_PIXELS) * 8	
} else {	
PredMv[ 0 ][ 0 ] = -(sbSize4 * MI_SIZE * 8)	
PredMv[ 0 ][ 1 ] = 0	
}	
}	
} else if ( compMode == GLOBALMV ) {	
PredMv[ i ] = GlobalMvs[ i ]	
} else {	

pos = ( compMode == NEARESTMV ) ? 0 : RefMvIdx	
if ( compMode == NEWMV && NumMvFound <= 1 )	
pos = 0	
PredMv[ i ] = RefStackMv[ pos ][ i ]	
}	
if ( compMode == NEWMV ) {	
read_mv( i )	
} else {	
Mv[ i ] = PredMv[ i ]	
}	
}	
}	

## 5.11.27. Read motion mode syntax

read_motion_mode( isCompound ) {	Type
if ( skip_mode ) {	
motion_mode = SIMPLE	
return	
}	
if ( !is_motion_mode_switchable ) {	
motion_mode = SIMPLE	
return	
}	
if ( Min( Block_Width[ MiSize ],	
Block_Height[ MiSize ] ) < 8 ) {	
motion_mode = SIMPLE	
return	
}	
if ( !force_integer_mv &&	
( YMode == GLOBALMV    YMode == GLOBAL_GLOBALMV ) ) {	
if ( GmType[ RefFrame[ 0 ] ] > TRANSLATION ) {	
motion_mode = SIMPLE	
return	
}	
}	
if ( isCompound    RefFrame[ 1 ] == INTRA_FRAME    !has_overlappable_candidates( ) ) {	
motion_mode = SIMPLE	
return	
}	
find_warp_samples()	
if ( force_integer_mv    NumSamples == 0	
!allow_warped_motion    is_scaled( RefFrame[0] ) ) {	
use_obmc	S()
motion_mode = use_obmc ? OBMC : SIMPLE	
} else {	
motion_mode	S()
}	
}	

where `is_scaled` is a function that determines whether a reference frame uses scaling and is specified as:

```
is_scaled( refFrame ) {
    refIdx = ref_frame_idx[ refFrame - LAST_FRAME ]
    xScale = ( ( RefUpscaledWidth[ refIdx ] << REF_SCALE_SHIFT ) + ( FrameWidth / 2 ) ) / FrameWidth
    yScale = ( ( RefFrameHeight[ refIdx ] << REF_SCALE_SHIFT ) + ( FrameHeight / 2 ) ) / FrameHeight
    noScale = 1 << REF_SCALE_SHIFT
    return xScale != noScale || yScale != noScale
}
```

## 5.11.28. Read inter intra syntax

<code>read_interintra_mode( isCompound ) {</code>	Type
<code>if ( !skip_mode &amp;&amp; enable_interintra_compound &amp;&amp; !isCompound &amp;&amp;</code>	
<code>    MiSize &gt;= BLOCK_8X8 &amp;&amp; MiSize &lt;= BLOCK_32X32) {</code>	
<code>        interintra</code>	S()
<code>        if ( interintra ) {</code>	
<code>            interintra_mode</code>	S()
<code>            RefFrame[1] = INTRA_FRAME</code>	
<code>            AngleDeltaY = 0</code>	
<code>            AngleDeltaUV = 0</code>	
<code>            use_filter_intra = 0</code>	
<code>            wedge_interintra</code>	S()
<code>            if ( wedge_interintra ) {</code>	
<code>                wedge_index</code>	S()
<code>                wedge_sign = 0</code>	
<code>            }</code>	
<code>        }</code>	
<code>    } else {</code>	
<code>        interintra = 0</code>	
<code>    }</code>	
<code>}</code>	

## 5.11.29. Read compound type syntax

<code>read_compound_type( isCompound ) {</code>	Type
<code>    comp_group_idx = 0</code>	
<code>    compound_idx = 1</code>	
<code>    if ( skip_mode ) {</code>	
<code>        compound_type = COMPOUND_AVERAGE</code>	
<code>        return</code>	
<code>    }</code>	
<code>    if ( isCompound ) {</code>	
<code>        n = Wedge_Bits[ MiSize ]</code>	
<code>        if ( enable_masked_compound ) {</code>	
<code>            comp_group_idx</code>	S()
<code>        }</code>	
<code>        if ( comp_group_idx == 0 ) {</code>	

if ( enable_jnt_comp ) {	
compound_idx	S()
compound_type = compound_idx ? COMPOUND_AVERAGE :	
COMPOUND_DISTANCE	
} else {	
compound_type = COMPOUND_AVERAGE	
}	
} else {	
if ( n == 0 ) {	
compound_type = COMPOUND_DIFFWTD	
} else {	
compound_type	S()
}	
}	
if ( compound_type == COMPOUND_WEDGE ) {	
wedge_index	S()
wedge_sign	L(1)
} else if ( compound_type == COMPOUND_DIFFWTD ) {	
mask_type	L(1)
}	
} else {	
if ( interintra ) {	
compound_type = wedge_interintra ? COMPOUND_WEDGE : COMPOUND_INTRA	
} else {	
compound_type = COMPOUND_AVERAGE	
}	
}	
}	

## 5.11.30. Get mode function

get_mode( refList ) {	Type
if ( refList == 0 ) {	
if ( YMode < NEAREST_NEARESTMV )	
compMode = YMode	
else if ( YMode == NEW_NEWMV    YMode == NEW_NEARESTMV    YMode == NEW_NEARMV )	
compMode = NEWMV	
else if ( YMode == NEAREST_NEARESTMV    YMode == NEAREST_NEWMV )	
compMode = NEARESTMV	
else if ( YMode == NEAR_NEARMV    YMode == NEAR_NEWMV )	
compMode = NEARMV	
else	
compMode = GLOBALMV	
} else {	
if ( YMode == NEW_NEWMV    YMode == NEAREST_NEWMV    YMode == NEAR_NEWMV )	
compMode = NEWMV	
else if ( YMode == NEAREST_NEARESTMV    YMode == NEW_NEARESTMV )	
compMode = NEARESTMV	
else if ( YMode == NEAR_NEARMV    YMode == NEW_NEARMV )	

compMode = NEARMV	
else	
compMode = GLOBALMV	
}	
return compMode	
}	

### 5.11.31. MV syntax

read_mv( ref ) {	Type
diffMv[ 0 ] = 0	
diffMv[ 1 ] = 0	
if ( use_intrabc ) {	
MvCtx = MV_INTRABC_CONTEXT	
} else {	
MvCtx = 0	
}	
mv_joint	S()
if ( mv_joint == MV_JOINT_HZVNZ    mv_joint == MV_JOINT_HNZVNZ )	
diffMv[ 0 ] = read_mv_component( 0 )	
if ( mv_joint == MV_JOINT_HNZVZ    mv_joint == MV_JOINT_HNZVNZ )	
diffMv[ 1 ] = read_mv_component( 1 )	
Mv[ ref ][ 0 ] = PredMv[ ref ][ 0 ] + diffMv[ 0 ]	
Mv[ ref ][ 1 ] = PredMv[ ref ][ 1 ] + diffMv[ 1 ]	
}	

### 5.11.32. MV component syntax

read_mv_component( comp ) {	Type
mv_sign	S()
mv_class	S()
if ( mv_class == MV_CLASS_0 ) {	
mv_class0_bit	S()
if ( force_integer_mv )	
mv_class0_fr = 3	
else	
mv_class0_fr	S()
if ( allow_high_precision_mv )	
mv_class0_hp	S()
else	
mv_class0_hp = 1	
mag = ( ( mv_class0_bit << 3 )	
( mv_class0_fr << 1 )	
mv_class0_hp ) + 1	
} else {	
d = 0	
for ( i = 0; i < mv_class; i++ ) {	
mv_bit	S()
d  = mv_bit << i	
}	

mag = CLASS0_SIZE << ( mv_class + 2 )	
if ( force_integer_mv )	
mv_fr = 3	
else	
mv_fr	S()
if ( allow_high_precision_mv )	
mv_hp	S()
else	
mv_hp = 1	
mag += ( ( d << 3 )   ( mv_fr << 1 )   mv_hp ) + 1	
}	
return mv_sign ? -mag : mag	
}	

### 5.11.33. Compute prediction syntax

compute_prediction() {	Type
sbMask = use_128x128_superblock ? 31 : 15	
subBlockMiRow = MiRow & sbMask	
subBlockMiCol = MiCol & sbMask	
for ( plane = 0; plane < 1 + HasChroma * 2; plane++ ) {	
planeSz = get_plane_residual_size( MiSize, plane )	
num4x4W = Num_4x4_Blocks_Wide[ planeSz ]	
num4x4H = Num_4x4_Blocks_High[ planeSz ]	
log2W = MI_SIZE_LOG2 + Mi_Width_Log2[ planeSz ]	
log2H = MI_SIZE_LOG2 + Mi_Height_Log2[ planeSz ]	
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
baseX = (MiCol >> subX) * MI_SIZE	
baseY = (MiRow >> subY) * MI_SIZE	
candRow = (MiRow >> subY) << subY	
candCol = (MiCol >> subX) << subX	
IsInterIntra = ( is_inter && RefFrame[ 1 ] == INTRA_FRAME )	
if ( IsInterIntra ) {	
if ( interintra_mode == II_DC_PRED ) mode = DC_PRED	
else if ( interintra_mode == II_V_PRED ) mode = V_PRED	
else if ( interintra_mode == II_H_PRED ) mode = H_PRED	
else mode = SMOOTH_PRED	
predict_intra( plane, baseX, baseY,	
plane == 0 ? AvailL : AvailLChroma,	
plane == 0 ? AvailU : AvailUChroma,	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) - 1 ]	
[ ( subBlockMiCol >> subX ) + num4x4W ],	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) + num4x4H ]	
[ ( subBlockMiCol >> subX ) - 1 ],	
mode,	

log2W, log2H )	
}	
if ( is_inter ) {	
predW = Block_Width[ MiSize ] >> subX	
predH = Block_Height[ MiSize ] >> subY	
someUseIntra = 0	
for ( r = 0; r < (num4x4H << subY); r++ )	
for ( c = 0; c < (num4x4W << subX); c++ )	
if ( RefFrames[ candRow + r ][ candCol + c ][ 0 ] == INTRA_FRAME )	
someUseIntra = 1	
if ( someUseIntra ) {	
predW = num4x4W * 4	
predH = num4x4H * 4	
candRow = MiRow	
candCol = MiCol	
}	
r = 0	
for ( y = 0; y < num4x4H * 4; y += predH ) {	
c = 0	
for ( x = 0; x < num4x4W * 4; x += predW ) {	
predict_inter( plane, baseX + x, baseY + y,	
predW, predH,	
candRow + r, candCol + c)	
c++	
}	
r++	
}	
}	
}	

### 5.11.34. Residual syntax

residual( ) {	Type
sbMask = use_128x128_superblock ? 31 : 15	
widthChunks = Max( 1, Block_Width[ MiSize ] >> 6 )	
heightChunks = Max( 1, Block_Height[ MiSize ] >> 6 )	
miSizeChunk = ( widthChunks > 1    heightChunks > 1 ) ? BLOCK_64X64 : MiSize	
for ( chunkY = 0; chunkY < heightChunks; chunkY++ ) {	
for ( chunkX = 0; chunkX < widthChunks; chunkX++ ) {	
miRowChunk = MiRow + ( chunkY << 4 )	
miColChunk = MiCol + ( chunkX << 4 )	
subBlockMiRow = miRowChunk & sbMask	
subBlockMiCol = miColChunk & sbMask	
for ( plane = 0; plane < 1 + HasChroma * 2; plane++ ) {	



txSz = Lossless ? TX_4X4 : get_tx_size( plane, TxSize )	
stepX = Tx_Width[ txSz ] >> 2	
stepY = Tx_Height[ txSz ] >> 2	
planeSz = get_plane_residual_size( miSizeChunk, plane )	
num4x4W = Num_4x4_Blocks_Wide[ planeSz ]	
num4x4H = Num_4x4_Blocks_High[ planeSz ]	
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
baseX = (miColChunk >> subX) * MI_SIZE	
baseY = (miRowChunk >> subY) * MI_SIZE	
if ( is_inter && !Lossless && !plane ) {	
transform_tree( baseX, baseY, num4x4W * 4, num4x4H * 4 )	
} else {	
baseXBlock = (MiCol >> subX) * MI_SIZE	
baseYBlock = (MiRow >> subY) * MI_SIZE	
for ( y = 0; y < num4x4H; y += stepY )	
for ( x = 0; x < num4x4W; x += stepX )	
transform_block( plane, baseXBlock, baseYBlock, txSz,	
x + ( ( chunkX << 4 ) >> subX ),	
y + ( ( chunkY << 4 ) >> subY ) )	
}	
}	
}	
}	

## 5.11.35. Transform block syntax

transform_block(plane, baseX, baseY, txSz, x, y) {	Type
startX = baseX + 4 * x	
startY = baseY + 4 * y	
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
row = ( startY << subY ) >> MI_SIZE_LOG2	
col = ( startX << subX ) >> MI_SIZE_LOG2	
sbMask = use_128x128_superblock ? 31 : 15	
subBlockMiRow = row & sbMask	
subBlockMiCol = col & sbMask	
stepX = Tx_Width[ txSz ] >> MI_SIZE_LOG2	
stepY = Tx_Height[ txSz ] >> MI_SIZE_LOG2	
maxX = (MiCols * MI_SIZE) >> subX	
maxY = (MiRows * MI_SIZE) >> subY	
if ( startX >= maxX    startY >= maxY ) {	
return	
}	
if ( !is_inter ) {	
if ( ( ( plane == 0 ) && PaletteSizeY )	
( ( plane != 0 ) && PaletteSizeUV ) ) {	
predict_palette( plane, startX, startY, x, y, txSz )	

} else {	
isCfl = (plane > 0 && UVMode == UV_CFL_PRED)	
if ( plane == 0 ) {	
mode = YMode	
} else {	
mode = ( isCfl ) ? DC_PRED : UVMode	
}	
log2W = Tx_Width_Log2[ txSz ]	
log2H = Tx_Height_Log2[ txSz ]	
predict_intra( plane, startX, startY,	
( plane == 0 ? AvailL : AvailLChroma )    x > 0,	
( plane == 0 ? AvailU : AvailUChroma )    y > 0,	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) - 1 ]	
[ ( subBlockMiCol >> subX ) + stepX ],	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) + stepY ]	
[ ( subBlockMiCol >> subX ) - 1 ],	
mode,	
log2W, log2H )	
if ( isCfl ) {	
predict_chroma_from_luma( plane, startX, startY, txSz )	
}	
}	
if ( plane == 0 ) {	
MaxLumaW = startX + stepX * 4	
MaxLumaH = startY + stepY * 4	
}	
}	
if ( !skip ) {	
eob = coeffs( plane, startX, startY, txSz )	
if ( eob > 0 )	
reconstruct( plane, startX, startY, txSz )	
}	
for ( i = 0; i < stepY; i++ ) {	
for ( j = 0; j < stepX; j++ ) {	
LoopfilterTxSizes[ plane ]	
[ (row >> subY) + i ]	
[ (col >> subX) + j ] = txSz	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) + i ]	
[ ( subBlockMiCol >> subX ) + j ] = 1	
}	
}	
}	

## 5.11.36. Transform tree syntax

`transform_tree` is used to read a number of transform blocks arranged in a transform tree.

	Type
<code>transform_tree( startX, startY, w, h ) {</code>	
<code>maxX = MiCols * MI_SIZE</code>	
<code>maxY = MiRows * MI_SIZE</code>	
<code>if ( startX &gt;= maxX    startY &gt;= maxY ) {</code>	
<code>return</code>	
<code>}</code>	
<code>row = startY &gt;&gt; MI_SIZE_LOG2</code>	
<code>col = startX &gt;&gt; MI_SIZE_LOG2</code>	
<code>lumaTxSz = InterTxSizes[ row ][ col ]</code>	
<code>lumaW = Tx_Width[ lumaTxSz ]</code>	
<code>lumaH = Tx_Height[ lumaTxSz ]</code>	
<code>uses64 = ( w == 64    h == 64 )</code>	
<code>if ( w &lt;= lumaW &amp;&amp; h &lt;= lumaH ) {</code>	
<code>txSz = find_tx_size( w, h )</code>	
<code>transform_block( 0, startX, startY, txSz, 0, 0 )</code>	
<code>} else {</code>	
<code>if ( w &gt; h ) {</code>	
<code>transform_tree( startX, startY, w/2, h )</code>	
<code>transform_tree( startX + w / 2, startY, w/2, h )</code>	
<code>} else if ( w &lt; h ) {</code>	
<code>transform_tree( startX, startY, w, h/2 )</code>	
<code>transform_tree( startX, startY + h/2, w, h/2 )</code>	
<code>} else {</code>	
<code>transform_tree( startX, startY, w/2, h/2 )</code>	
<code>transform_tree( startX + w/2, startY, w/2, h/2 )</code>	
<code>transform_tree( startX, startY + h/2, w/2, h/2 )</code>	
<code>transform_tree( startX + w/2, startY + h/2, w/2, h/2 )</code>	
<code>}</code>	
<code>}</code>	
<code>}</code>	

where `find_tx_size` finds the transform size matching the given dimensions and is defined as:

```
find_tx_size( w, h ) {
    for ( txSz = 0; txSz < TX_SIZES_ALL; txSz++ )
        if ( Tx_Width[ txSz ] == w && Tx_Height[ txSz ] == h )
            break
    return txSz
}
```

## 5.11.37. Get TX size function

	Type
<code>get_tx_size( plane, txSz ) {</code>	
<code>if ( plane == 0 )</code>	

return txSz	
uvTx = Max_Tx_Size_Rect[ get_plane_residual_size( MiSize, plane ) ]	
if ( Tx_Width[ uvTx ] == 64    Tx_Height[ uvTx ] == 64 ){	
if ( Tx_Width[ uvTx ] == 16 ) {	
return TX_16X32	
}	
if ( Tx_Height[ uvTx ] == 16 ) {	
return TX_32X16	
}	
return TX_32X32	
}	
return uvTx	
}	

## 5.11.38. Get plane residual size function

The `get_plane_residual_size` returns the size of a residual block for the specified plane. (The residual block will always have width and height at least equal to 4.)

<code>get_plane_residual_size( subsize, plane ) {</code>	<b>Type</b>
<code>subx = plane &gt; 0 ? subsampling_x : 0</code>	
<code>suby = plane &gt; 0 ? subsampling_y : 0</code>	
<code>return Subsampled_Size[ subsize ][ subx ][ suby ]</code>	
<code>}</code>	

The `Subsampled_Size` table is defined as:

```

Subsampled_Size[ BLOCK_SIZES ][ 2 ][ 2 ] = {
  { { BLOCK_4X4,    BLOCK_4X4},    {BLOCK_4X4,    BLOCK_4X4} },
  { { BLOCK_4X8,    BLOCK_4X4},    {BLOCK_INVALID, BLOCK_4X4} },
  { { BLOCK_8X4,    BLOCK_INVALID}, {BLOCK_4X4,    BLOCK_4X4} },
  { { BLOCK_8X8,    BLOCK_8X4},    {BLOCK_4X8,    BLOCK_4X4} },
  { {BLOCK_8X16,    BLOCK_8X8},    {BLOCK_INVALID, BLOCK_4X8} },
  { {BLOCK_16X8,    BLOCK_INVALID}, {BLOCK_8X8,    BLOCK_8X4} },
  { {BLOCK_16X16,   BLOCK_16X8},    {BLOCK_8X16,   BLOCK_8X8} },
  { {BLOCK_16X32,   BLOCK_16X16},    {BLOCK_INVALID, BLOCK_8X16} },
  { {BLOCK_32X16,   BLOCK_INVALID}, {BLOCK_16X16,   BLOCK_16X8} },
  { {BLOCK_32X32,   BLOCK_32X16},    {BLOCK_16X32,   BLOCK_16X16} },
  { {BLOCK_32X64,   BLOCK_32X32},    {BLOCK_INVALID, BLOCK_16X32} },
  { {BLOCK_64X32,   BLOCK_INVALID}, {BLOCK_32X32,   BLOCK_32X16} },
  { {BLOCK_64X64,   BLOCK_64X32},    {BLOCK_32X64,   BLOCK_32X32} },
  { {BLOCK_64X128,  BLOCK_64X64},    {BLOCK_INVALID, BLOCK_32X64} },
  { {BLOCK_128X64,  BLOCK_INVALID}, {BLOCK_64X64,   BLOCK_64X32} },
  { {BLOCK_128X128, BLOCK_128X64},   {BLOCK_64X128,  BLOCK_64X64} },
  { {BLOCK_4X16,    BLOCK_4X8},      {BLOCK_INVALID, BLOCK_4X8} },
  { {BLOCK_16X4,    BLOCK_INVALID}, {BLOCK_8X4,     BLOCK_8X4} },
  { {BLOCK_8X32,    BLOCK_8X16},     {BLOCK_INVALID, BLOCK_4X16} },
  { {BLOCK_32X8,    BLOCK_INVALID}, {BLOCK_16X8,    BLOCK_16X4} },
  { {BLOCK_16X64,   BLOCK_16X32},    {BLOCK_INVALID, BLOCK_8X32} },
  { {BLOCK_64X16,   BLOCK_INVALID}, {BLOCK_32X16,   BLOCK_32X8} },
}

```

## 5.11.39. Coefficients syntax

coeffs( plane, startX, startY, txSz ) {	Type
x4 = startX >> 2	
y4 = startY >> 2	
w4 = Tx_Width[ txSz ] >> 2	
h4 = Tx_Height[ txSz ] >> 2	
txSzCtx = ( Tx_Size_Sqr[txSz] + Tx_Size_Sqr_Up[txSz] + 1 ) >> 1	
ptype = plane > 0	
segEob = ( txSz == TX_16X64    txSz == TX_64X16 ) ? 512 :	
Min( 1024, Tx_Width[ txSz ] * Tx_Height[ txSz ] )	
for ( c = 0; c < segEob; c++ )	
Quant[c] = 0	
for ( i = 0; i < 64; i++ )	
for ( j = 0; j < 64; j++ )	
Dequant[ i ][ j ] = 0	
eob = 0	
culLevel = 0	
dcCategory = 0	
all_zero	S()

if ( all_zero ) {	
c = 0	
if ( plane == 0 ) {	
for ( i = 0; i < w4; i++ ) {	
for ( j = 0; j < h4; j++ ) {	
TxTypes[ y4 + j ][ x4 + i ] = DCT_DCT	
}	
}	
}	
} else {	
if ( plane == 0 )	
transform_type( x4, y4, txSz )	
PlaneTxType = compute_tx_type( plane, txSz, x4, y4 )	
scan = get_scan( plane, txSz )	
eobMultisize = Min( Tx_Width_Log2[ txSz ], 5 ) + Min( Tx_Height_Log2[ txSz ], 5 ) - 4	
if ( eobMultisize == 0 ) {	
eob_pt_16	S()
eobPt = eob_pt_16 + 1	
} else if ( eobMultisize == 1 ) {	
eob_pt_32	S()
eobPt = eob_pt_32 + 1	
} else if ( eobMultisize == 2 ) {	
eob_pt_64	S()
eobPt = eob_pt_64 + 1	
} else if ( eobMultisize == 3 ) {	
eob_pt_128	S()
eobPt = eob_pt_128 + 1	
} else if ( eobMultisize == 4 ) {	
eob_pt_256	S()
eobPt = eob_pt_256 + 1	
} else if ( eobMultisize == 5 ) {	
eob_pt_512	S()
eobPt = eob_pt_512 + 1	
} else {	
eob_pt_1024	S()
eobPt = eob_pt_1024 + 1	
}	
eob = ( eobPt < 2 ) ? eobPt : ( ( 1 << ( eobPt - 2 ) ) + 1 )	
eobShift = Max( -1, eobPt - 3 )	
if ( eobShift >= 0 ) {	
eob_extra	S()
if ( eob_extra ) {	
eob += ( 1 << eobShift )	
}	
for ( i = 1; i < Max( 0, eobPt - 2 ); i++ ) {	
eobShift = Max( 0, eobPt - 2 ) - 1 - i	

eob_extra_bit	L(1)
if ( eob_extra_bit ) {	
eob += ( 1 << eobShift )	
}	
}	
for ( c = eob - 1; c >= 0; c-- ) {	
pos = scan[ c ]	
if ( c == ( eob - 1 ) ) {	
coeff_base_eob	S()
level = coeff_base_eob + 1	
} else {	
coeff_base	S()
level = coeff_base	
}	
if ( level > NUM_BASE_LEVELS ) {	
for ( idx = 0;	
idx < COEFF_BASE_RANGE / ( BR_CDF_SIZE - 1 );	
idx++ ) {	
coeff_br	S()
level += coeff_br	
if ( coeff_br < ( BR_CDF_SIZE - 1 ) )	
break	
}	
}	
Quant[ pos ] = level	
}	
for ( c = 0; c < eob; c++ ) {	
pos = scan[ c ]	
if ( Quant[ pos ] != 0 ) {	
if ( c == 0 ) {	
dc_sign	S()
sign = dc_sign	
} else {	
sign_bit	L(1)
sign = sign_bit	
}	
} else {	
sign = 0	
}	
if ( Quant[ pos ] >	
( NUM_BASE_LEVELS + COEFF_BASE_RANGE ) ) {	
length = 0	
do {	
length++	
golomb_length_bit	L(1)
} while ( !golomb_length_bit )	

x = 1	
for ( i = length - 2; i >= 0; i-- ) {	
golomb_data_bit	L(1)
x = ( x << 1 )   golomb_data_bit	
}	
Quant[ pos ] = x + COEFF_BASE_RANGE + NUM_BASE_LEVELS	
}	
if ( pos == 0 && Quant[ pos ] > 0 ) {	
dcCategory = sign ? 1 : 2	
}	
Quant[ pos ] = Quant[ pos ] & 0xFFFFF	
culLevel += Quant[ pos ]	
if ( sign )	
Quant[ pos ] = - Quant[ pos ]	
}	
culLevel = Min( 63, culLevel )	
}	
for ( i = 0; i < w4; i++ ) {	
AboveLevelContext[ plane ][ x4 + i ] = culLevel	
AboveDcContext[ plane ][ x4 + i ] = dcCategory	
}	
for ( i = 0; i < h4; i++ ) {	
LeftLevelContext[ plane ][ y4 + i ] = culLevel	
LeftDcContext[ plane ][ y4 + i ] = dcCategory	
}	
return eob	
}	

## 5.11.40. Compute transform type function

compute_tx_type( plane, txSz, blockX, blockY ) {	Type
txSzSqrUp = Tx_Size_Sqr_Up[ txSz ]	
if ( Lossless    txSzSqrUp > TX_32X32 )	
return DCT_DCT	
txSet = get_tx_set( txSz )	
if ( plane == 0 ) {	
return TxTypes[ blockY ][ blockX ]	
}	
if ( is_inter ) {	
x4 = Max( MiCol, blockX << subsampling_x )	
y4 = Max( MiRow, blockY << subsampling_y )	
txType = TxTypes[ y4 ][ x4 ]	
if ( !is_tx_type_in_set( txSet, txType ) )	



return DCT_DCT	
return txType	
}	
txType = Mode_To_Txfrm[ UVMode ]	
if ( !is_tx_type_in_set( txSet, txType ) )	
return DCT_DCT	
return txType	
}	
is_tx_type_in_set( txSet, txType ) {	
return is_inter ? Tx_Type_In_Set_Inter[ txSet ][ txType ] :	
Tx_Type_In_Set_Intra[ txSet ][ txType ]	
}	

where the tables Tx\_Type\_In\_Set\_Inter and Tx\_Type\_In\_Set\_Intra are specified as follows:

```

Tx_Type_In_Set_Intra[ TX_SET_TYPES_INTRA ][ TX_TYPES ] = {
{
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
},
{
1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
},
{
1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
}
}

Tx_Type_In_Set_Inter[ TX_SET_TYPES_INTER ][ TX_TYPES ] = {
{
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
},
{
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
},
{
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
},
{
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
}
}

```

## 5.11.41. Get scan function

get_mrow_scan( txSz ) {	Type
if ( txSz == TX_4X4 )	
return Mrow_Scan_4x4	

else if ( txSz == TX_4X8 )	
return Mrow_Scan_4x8	
else if ( txSz == TX_8X4 )	
return Mrow_Scan_8x4	
else if ( txSz == TX_8X8 )	
return Mrow_Scan_8x8	
else if ( txSz == TX_8X16 )	
return Mrow_Scan_8x16	
else if ( txSz == TX_16X8 )	
return Mrow_Scan_16x8	
else if ( txSz == TX_16X16 )	
return Mrow_Scan_16x16	
else if ( txSz == TX_4X16 )	
return Mrow_Scan_4x16	
return Mrow_Scan_16x4	
}	
get_mcol_scan( txSz ) {	
if ( txSz == TX_4X4 )	
return Mcol_Scan_4x4	
else if ( txSz == TX_4X8 )	
return Mcol_Scan_4x8	
else if ( txSz == TX_8X4 )	
return Mcol_Scan_8x4	
else if ( txSz == TX_8X8 )	
return Mcol_Scan_8x8	
else if ( txSz == TX_8X16 )	
return Mcol_Scan_8x16	
else if ( txSz == TX_16X8 )	
return Mcol_Scan_16x8	
else if ( txSz == TX_16X16 )	
return Mcol_Scan_16x16	
else if ( txSz == TX_4X16 )	
return Mcol_Scan_4x16	
return Mcol_Scan_16x4	
}	
get_default_scan( txSz ) {	
if ( txSz == TX_4X4 )	
return Default_Scan_4x4	
else if ( txSz == TX_4X8 )	
return Default_Scan_4x8	
else if ( txSz == TX_8X4 )	
return Default_Scan_8x4	
else if ( txSz == TX_8X8 )	
return Default_Scan_8x8	
else if ( txSz == TX_8X16 )	
return Default_Scan_8x16	
else if ( txSz == TX_16X8 )	

return Default_Scan_16x8	
else if ( txSz == TX_16X16 )	
return Default_Scan_16x16	
else if ( txSz == TX_16X32 )	
return Default_Scan_16x32	
else if ( txSz == TX_32X16 )	
return Default_Scan_32x16	
else if ( txSz == TX_4X16 )	
return Default_Scan_4x16	
else if ( txSz == TX_16X4 )	
return Default_Scan_16x4	
else if ( txSz == TX_8X32 )	
return Default_Scan_8x32	
else if ( txSz == TX_32X8 )	
return Default_Scan_32x8	
return Default_Scan_32x32	
}	
get_scan( plane, txSz ) {	
if ( txSz == TX_16X64 ) {	
return Default_Scan_16x32	
}	
if ( txSz == TX_64X16 ) {	
return Default_Scan_32x16	
}	
if ( Tx_Size_Sqr_Up[ txSz ] == TX_64X64 ) {	
return Default_Scan_32x32	
}	
if ( PlaneTxType == IDTX ) {	
return get_default_scan( txSz )	
}	
preferRow = ( PlaneTxType == V_DCT	
PlaneTxType == V_ADST	
PlaneTxType == V_FLIPADST )	
preferCol = ( PlaneTxType == H_DCT	
PlaneTxType == H_ADST	
PlaneTxType == H_FLIPADST )	
if ( preferRow ) {	
return get_mrow_scan( txSz )	
} else if ( preferCol ) {	
return get_mcol_scan( txSz )	
}	
return get_default_scan( txSz )	
}	

## 5.11.42. Intra angle info luma syntax

intra_angle_info_y( ) {	Type
AngleDeltaY = 0	
if ( MiSize >= BLOCK_8X8 ) {	
if ( is_directional_mode( YMode ) ) {	
angle_delta_y	S()
AngleDeltaY = angle_delta_y - MAX_ANGLE_DELTA	
}	
}	
}	

## 5.11.43. Intra angle info chroma syntax

intra_angle_info_uv( ) {	Type
AngleDeltaUV = 0	
if ( MiSize >= BLOCK_8X8 ) {	
if ( is_directional_mode( UVMode ) ) {	
angle_delta_uv	S()
AngleDeltaUV = angle_delta_uv - MAX_ANGLE_DELTA	
}	
}	
}	

## 5.11.44. Is directional mode function

is_directional_mode( mode ) {	Type
if ( ( mode >= V_PRED ) && ( mode <= D67_PRED ) ) {	
return 1	
}	
return 0	
}	

## 5.11.45. Read CFL alphas syntax

read_cfl_alphas() {	Type
cfl_alpha_signs	S()
signU = (cfl_alpha_signs + 1 ) / 3	
signV = (cfl_alpha_signs + 1 ) % 3	
if ( signU != CFL_SIGN_ZERO ) {	
cfl_alpha_u	S()
CflAlphaU = 1 + cfl_alpha_u	
if ( signU == CFL_SIGN_NEG )	
CflAlphaU = -CflAlphaU	
} else {	
CflAlphaU = 0	
}	
if ( signV != CFL_SIGN_ZERO ) {	
cfl_alpha_v	S()
CflAlphaV = 1 + cfl_alpha_v	

if ( signV == CFL_SIGN_NEG )	
CflAlphaV = -CflAlphaV	
} else {	
CflAlphaV = 0	
}	

## 5.11.46. Palette mode info syntax

palette_mode_info( ) {	Type
bsizeCtx = Mi_Width_Log2[ MiSize ] + Mi_Height_Log2[ MiSize ] - 2	
if ( YMode == DC_PRED ) {	
has_palette_y	S()
if ( has_palette_y ) {	
palette_size_y_minus_2	S()
PaletteSizeY = palette_size_y_minus_2 + 2	
cacheN = get_palette_cache( 0 )	
idx = 0	
for ( i = 0; i < cacheN && idx < PaletteSizeY; i++ ) {	
use_palette_color_cache_y	L(1)
if ( use_palette_color_cache_y ) {	
palette_colors_y[ idx ] = PaletteCache[ i ]	
idx++	
}	
}	
if ( idx < PaletteSizeY ) {	
palette_colors_y[ idx ]	L(BitDepth)
idx++	
}	
if ( idx < PaletteSizeY ) {	
minBits = BitDepth - 3	
palette_num_extra_bits_y	L(2)
paletteBits = minBits + palette_num_extra_bits_y	
}	
while ( idx < PaletteSizeY ) {	
palette_delta_y	L(paletteBits)
palette_delta_y++	
palette_colors_y[ idx ] =	
Clip1( palette_colors_y[ idx - 1 ] +	
palette_delta_y )	
range = ( 1 << BitDepth ) - palette_colors_y[ idx ] - 1	
paletteBits = Min( paletteBits, CeilLog2( range ) )	
idx++	
}	
sort( palette_colors_y, 0, PaletteSizeY - 1 )	
}	
}	
if ( HasChroma && UVMode == DC_PRED ) {	
has_palette_uv	S()
if ( has_palette_uv ) {	

<code>palette_size_uv_minus_2</code>	<code>S()</code>
<code>PaletteSizeUV = palette_size_uv_minus_2 + 2</code>	
<code>cacheN = get_palette_cache( 1 )</code>	
<code>idx = 0</code>	
<code>for ( i = 0; i &lt; cacheN &amp;&amp; idx &lt; PaletteSizeUV; i++ ) {</code>	
<code>use_palette_color_cache_u</code>	<code>L(1)</code>
<code>if ( use_palette_color_cache_u ) {</code>	
<code>palette_colors_u[ idx ] = PaletteCache[ i ]</code>	
<code>idx++</code>	
<code>}</code>	
<code>}</code>	
<code>if ( idx &lt; PaletteSizeUV ) {</code>	
<code>palette_colors_u[ idx ]</code>	<code>L(BitDepth)</code>
<code>idx++</code>	
<code>}</code>	
<code>if ( idx &lt; PaletteSizeUV ) {</code>	
<code>minBits = BitDepth - 3</code>	
<code>palette_num_extra_bits_u</code>	<code>L(2)</code>
<code>paletteBits = minBits + palette_num_extra_bits_u</code>	
<code>}</code>	
<code>while ( idx &lt; PaletteSizeUV ) {</code>	
<code>palette_delta_u</code>	<code>L(paletteBits)</code>
<code>palette_colors_u[ idx ] =</code>	
<code>Clip1( palette_colors_u[ idx - 1 ] +</code>	
<code>palette_delta_u )</code>	
<code>range = ( 1 &lt;&lt; BitDepth ) - palette_colors_u[ idx ]</code>	
<code>paletteBits = Min( paletteBits, CeilLog2( range ) )</code>	
<code>idx++</code>	
<code>}</code>	
<code>sort( palette_colors_u, 0, PaletteSizeUV - 1 )</code>	
 <code>delta_encode_palette_colors_v</code>	 <code>L(1)</code>
<code>if ( delta_encode_palette_colors_v ) {</code>	
<code>minBits = BitDepth - 4</code>	
<code>maxVal = 1 &lt;&lt; BitDepth</code>	
<code>palette_num_extra_bits_v</code>	<code>L(2)</code>
<code>paletteBits = minBits + palette_num_extra_bits_v</code>	
<code>palette_colors_v[ 0 ]</code>	<code>L(BitDepth)</code>
<code>for ( idx = 1; idx &lt; PaletteSizeUV; idx++ ) {</code>	
<code>palette_delta_v</code>	<code>L(paletteBits)</code>
<code>if ( palette_delta_v ) {</code>	
<code>palette_delta_sign_bit_v</code>	<code>L(1)</code>
<code>if ( palette_delta_sign_bit_v ) {</code>	
<code>palette_delta_v = -palette_delta_v</code>	
<code>}</code>	
<code>}</code>	
<code>val = palette_colors_v[ idx - 1 ] + palette_delta_v</code>	
<code>if ( val &lt; 0 ) val += maxVal</code>	
<code>if ( val &gt;= maxVal ) val -= maxVal</code>	

palette_colors_v[ idx ] = Clip1( val )	
}	
} else {	
for ( idx = 0; idx < PaletteSizeUV; idx++ ) {	
palette_colors_v[ idx ]	L(BitDepth)
}	
}	
}	
}	
}	

The function sort( arr, i1, i2 ) sorts a subarray of the array arr in-place into ascending order. The subarray to be sorted is between indices i1 and i2 inclusive.

**Note:** The palette colors are generated in ascending order. The palette cache is also in ascending order. This means that the sort function can be replaced in implementations by a merge of two sorted lists.

where the function get\_palette\_cache, which merges the above and left palettes to form a cache, is specified as follows:

get_palette_cache( plane ) {	Type
aboveN = 0	
if ( ( MiRow * MI_SIZE ) % 64 ) {	
aboveN = PaletteSizes[ plane ][ MiRow - 1 ][ MiCol ]	
}	
leftN = 0	
if ( AvailL ) {	
leftN = PaletteSizes[ plane ][ MiRow ][ MiCol - 1 ]	
}	
aboveIdx = 0	
leftIdx = 0	
n = 0	
while ( aboveIdx < aboveN && leftIdx < leftN ) {	
aboveC = PaletteColors[ plane ][ MiRow - 1 ][ MiCol ][ aboveIdx ]	
leftC = PaletteColors[ plane ][ MiRow ][ MiCol - 1 ][ leftIdx ]	
if ( leftC < aboveC ) {	
if ( n == 0    leftC != PaletteCache[ n - 1 ] ) {	
PaletteCache[ n ] = leftC	
n++	
}	
leftIdx++	
} else {	
if ( n == 0    aboveC != PaletteCache[ n - 1 ] ) {	
PaletteCache[ n ] = aboveC	
n++	
}	
aboveIdx++	
if ( leftC == aboveC ) {	
leftIdx++	

}	
}	
}	
while ( aboveIdx < aboveN ) {	
val = PaletteColors[ plane ][ MiRow - 1 ][ MiCol ][ aboveIdx ]	
aboveIdx++	
if ( n == 0    val != PaletteCache[ n - 1 ] ) {	
PaletteCache[ n ] = val	
n++	
}	
}	
while ( leftIdx < leftN ) {	
val = PaletteColors[ plane ][ MiRow ][ MiCol - 1 ][ leftIdx ]	
leftIdx++	
if ( n == 0    val != PaletteCache[ n - 1 ] ) {	
PaletteCache[ n ] = val	
n++	
}	
}	
return n	
}	

**Note:** `get_palette_cache` is equivalent to sorting the available palette colors from above and left together and removing any duplicates.

## 5.11.47. Transform type syntax

transform_type( x4, y4, txSz ) {	Type
set = get_tx_set( txSz )	
if ( set > 0 &&	
( segmentation_enabled ? get_qindex( 1, segment_id ) : base_q_idx ) > 0 ) {	
if ( is_inter ) {	
inter_tx_type	S()
if ( set == TX_SET_INTER_1 )	
TxType = Tx_Type_Inter_Inv_Set1[ inter_tx_type ]	
else if ( set == TX_SET_INTER_2 )	
TxType = Tx_Type_Inter_Inv_Set2[ inter_tx_type ]	
else	
TxType = Tx_Type_Inter_Inv_Set3[ inter_tx_type ]	
} else {	
intra_tx_type	S()
if ( set == TX_SET_INTRA_1 )	
TxType = Tx_Type_Intra_Inv_Set1[ intra_tx_type ]	
else	
TxType = Tx_Type_Intra_Inv_Set2[ intra_tx_type ]	
}	
} else {	



TxType = DCT_DCT	
}	
for ( i = 0; i < ( Tx_Width[ txSz ] >> 2 ); i++ ) {	
for ( j = 0; j < ( Tx_Height[ txSz ] >> 2 ); j++ ) {	
TxTypes[ y4 + j ][ x4 + i ] = TxType	
}	
}	
}	

where the inversion tables used in the function are specified as follows:

```

Tx_Type_Intra_Inv_Set1[ 7 ] = { IDTX, DCT_DCT, V_DCT, H_DCT, ADST_ADST, ADST_DCT, DCT_ADST }
Tx_Type_Intra_Inv_Set2[ 5 ] = { IDTX, DCT_DCT, ADST_ADST, ADST_DCT, DCT_ADST }
Tx_Type_Inter_Inv_Set1[ 16 ] = { IDTX, V_DCT, H_DCT, V_ADST, H_ADST, V_FLIPADST, H_FLIPADST,
                                DCT_DCT, ADST_DCT, DCT_ADST, FLIPADST_DCT, DCT_FLIPADST, ADST_ADST,
                                FLIPADST_FLIPADST, ADST_FLIPADST, FLIPADST_ADST }
Tx_Type_Inter_Inv_Set2[ 12 ] = { IDTX, V_DCT, H_DCT, DCT_DCT, ADST_DCT, DCT_ADST, FLIPADST_DCT,
                                DCT_FLIPADST, ADST_ADST, FLIPADST_FLIPADST, ADST_FLIPADST,
                                FLIPADST_ADST }
Tx_Type_Inter_Inv_Set3[ 2 ] = { IDTX, DCT_DCT }

```

## 5.11.48. Get transform set function

get_tx_set( txSz ) {	Type
txSzSqr = Tx_Size_Sqr[ txSz ]	
txSzSqrUp = Tx_Size_Sqr_Up[ txSz ]	
if ( txSzSqrUp > TX_32X32 )	
return TX_SET_DCTONLY	
if ( is_inter ) {	
if ( reduced_tx_set    txSzSqrUp == TX_32X32 ) return TX_SET_INTER_3	
else if ( txSzSqr == TX_16X16 ) return TX_SET_INTER_2	
return TX_SET_INTER_1	
} else {	
if ( txSzSqrUp == TX_32X32 ) return TX_SET_DCTONLY	
else if ( reduced_tx_set ) return TX_SET_INTRA_2	
else if ( txSzSqr == TX_16X16 ) return TX_SET_INTRA_2	
return TX_SET_INTRA_1	
}	
}	

## 5.11.49. Palette tokens syntax

palette_tokens( ) {	Type
blockHeight = Block_Height[ MiSize ]	
blockWidth = Block_Width[ MiSize ]	
onscreenHeight = Min( blockHeight, (MiRows - MiRow) * MI_SIZE )	
onscreenWidth = Min( blockWidth, (MiCols - MiCol) * MI_SIZE )	

if ( PaletteSizeY ) {	
<b>color_index_map_y</b>	NS(PaletteSizeY)
ColorMapY[0][0] = color_index_map_y	
for ( i = 1; i < onscreenHeight + onscreenWidth - 1; i++ ) {	
for ( j = Min( i, onscreenWidth - 1 );	
j >= Max( 0, i - onscreenHeight + 1 ); j-- ) {	
get_palette_color_context(	
ColorMapY, ( i - j ), j, PaletteSizeY )	
<b>palette_color_idx_y</b>	S()
ColorMapY[ i - j ][ j ] = ColorOrder[ palette_color_idx_y ]	
}	
}	
for ( i = 0; i < onscreenHeight; i++ ) {	
for ( j = onscreenWidth; j < blockWidth; j++ ) {	
ColorMapY[ i ][ j ] = ColorMapY[ i ][ onscreenWidth - 1 ]	
}	
}	
for ( i = onscreenHeight; i < blockHeight; i++ ) {	
for ( j = 0; j < blockWidth; j++ ) {	
ColorMapY[ i ][ j ] = ColorMapY[ onscreenHeight - 1 ][ j ]	
}	
}	
}	
if ( PaletteSizeUV ) {	
<b>color_index_map_uv</b>	NS(PaletteSizeUV)
ColorMapUV[0][0] = color_index_map_uv	
blockHeight = blockHeight >> subsampling_y	
blockWidth = blockWidth >> subsampling_x	
onscreenHeight = onscreenHeight >> subsampling_y	
onscreenWidth = onscreenWidth >> subsampling_x	
if ( blockWidth < 4 ) {	
blockWidth += 2	
onscreenWidth += 2	
}	
if ( blockHeight < 4 ) {	
blockHeight += 2	
onscreenHeight += 2	
}	
for ( i = 1; i < onscreenHeight + onscreenWidth - 1; i++ ) {	
for ( j = Min( i, onscreenWidth - 1 );	
j >= Max( 0, i - onscreenHeight + 1 ); j-- ) {	
get_palette_color_context(	
ColorMapUV, ( i - j ), j, PaletteSizeUV )	
<b>palette_color_idx_uv</b>	S()
ColorMapUV[ i - j ][ j ] = ColorOrder[ palette_color_idx_uv ]	
}	
}	

for ( i = 0; i < onscreenHeight; i++ ) {	
for ( j = onscreenWidth; j < blockWidth; j++ ) {	
ColorMapUV[ i ][ j ] = ColorMapUV[ i ][ onscreenWidth - 1 ]	
}	
}	
for ( i = onscreenHeight; i < blockHeight; i++ ) {	
for ( j = 0; j < blockWidth; j++ ) {	
ColorMapUV[ i ][ j ] = ColorMapUV[ onscreenHeight - 1 ][ j ]	
}	
}	
}	
}	

## 5.11.50. Palette color context function

get_palette_color_context( colorMap, r, c, n ) {	Type
for ( i = 0; i < PALETTE_COLORS; i++ ) {	
scores[ i ] = 0	
ColorOrder[i] = i	
}	
if ( c > 0 ) {	
neighbor = colorMap[ r ][ c - 1 ]	
scores[ neighbor ] += 2	
}	
if ( ( r > 0 ) && ( c > 0 ) ) {	
neighbor = colorMap[ r - 1 ][ c - 1 ]	
scores[ neighbor ] += 1	
}	
if ( r > 0 ) {	
neighbor = colorMap[ r - 1 ][ c ]	
scores[ neighbor ] += 2	
}	
for ( i = 0; i < PALETTE_NUM_NEIGHBORS; i++ ) {	
maxScore = scores[ i ]	
maxIdx = i	
for ( j = i + 1; j < n; j++ ) {	
if ( scores[ j ] > maxScore ) {	
maxScore = scores[ j ]	
maxIdx = j	
}	
}	
if ( maxIdx != i ) {	
maxScore = scores[ maxIdx ]	
maxColorOrder = ColorOrder[ maxIdx ]	
for ( k = maxIdx; k > i; k-- ) {	
scores[ k ] = scores[ k - 1 ]	
ColorOrder[ k ] = ColorOrder[ k - 1 ]	
}	
scores[ i ] = maxScore	

ColorOrder[ i ] = maxColorOrder	
}	
}	
ColorContextHash = 0	
for ( i = 0; i < PALETTE_NUM_NEIGHBORS; i++ ) {	
ColorContextHash += scores[ i ] * Palette_Color_Hash_Multipliers[ i ]	
}	
}	

### 5.11.51. Is inside function

is\_inside determines whether a candidate position is inside the current tile.

is_inside( candidateR, candidateC ) {	Type
return ( candidateC >= MiColStart &&	
candidateC < MiColEnd &&	
candidateR >= MiRowStart &&	
candidateR < MiRowEnd )	
}	

### 5.11.52. Is inside filter region function

is\_inside\_filter\_region determines whether a candidate position is inside the region that is being used for CDEF filtering.

is_inside_filter_region( candidateR, candidateC ) {	Type
colStart = 0	
colEnd = MiCols	
rowStart = 0	
rowEnd = MiRows	
return (candidateC >= colStart &&	
candidateC < colEnd &&	
candidateR >= rowStart &&	
candidateR < rowEnd)	
}	

### 5.11.53. Clamp MV row function

clamp_mv_row( mvec, border ) {	Type
bh4 = Num_4x4_Blocks_High[ MiSize ]	
mbToTopEdge = -((MiRow * MI_SIZE) * 8)	
mbToBottomEdge = ((MiRows - bh4 - MiRow) * MI_SIZE) * 8	
return Clip3( mbToTopEdge - border, mbToBottomEdge + border, mvec )	
}	

### 5.11.54. Clamp MV col function

clamp_mv_col( mvec, border ) {	Type
bw4 = Num_4x4_Blocks_Wide[ MiSize ]	
mbToLeftEdge = -((MiCol * MI_SIZE) * 8)	
mbToRightEdge = ((MiCols - bw4 - MiCol) * MI_SIZE) * 8	

return Clip3( mbToLeftEdge - border, mbToRightEdge + border, mvec )	
}	

## 5.11.55. Clear CDEF function

clear_cdef( r, c ) {	Type
cdef_idx[ r ][ c ] = -1	
if ( use_128x128_superblock ) {	
cdefSize4 = Num_4x4_Blocks_Wide[ BLOCK_64X64 ]	
cdef_idx[ r ][ c + cdefSize4 ] = -1	
cdef_idx[ r + cdefSize4 ][ c ] = -1	
cdef_idx[ r + cdefSize4 ][ c + cdefSize4 ] = -1	
}	
}	

## 5.11.56. Read CDEF syntax

read_cdef( ) {	Type
if ( skip    CodedLossless    !enable_cdef    allow_intrabc ) {	
return	
}	
cdefSize4 = Num_4x4_Blocks_Wide[ BLOCK_64X64 ]	
cdefMask4 = ~(cdefSize4 - 1)	
r = MiRow & cdefMask4	
c = MiCol & cdefMask4	
if ( cdef_idx[ r ][ c ] == -1 ) {	
cdef_idx[ r ][ c ]	L(cdef_bits)
w4 = Num_4x4_Blocks_Wide[ MiSize ]	
h4 = Num_4x4_Blocks_High[ MiSize ]	
for ( i = r; i < r + h4 ; i += cdefSize4 ) {	
for ( j = c; j < c + w4 ; j += cdefSize4 ) {	
cdef_idx[ i ][ j ] = cdef_idx[ r ][ c ]	
}	
}	
}	
}	

## 5.11.57. Read loop restoration syntax

read_lr( r, c, bSize ) {	Type
if ( allow_intrabc ) {	
return	
}	
w = Num_4x4_Blocks_Wide[ bSize ]	
h = Num_4x4_Blocks_High[ bSize ]	
for ( plane = 0; plane < NumPlanes; plane++ ) {	
if ( FrameRestorationType[ plane ] != RESTORE_NONE ) {	
subX = (plane == 0) ? 0 : subsampling_x	
subY = (plane == 0) ? 0 : subsampling_y	
unitSize = LoopRestorationSize[ plane ]	

unitRows = count_units_in_frame( unitSize, Round2( FrameHeight, subY )	
unitCols = count_units_in_frame( unitSize, Round2( UpscaledWidth, subX )	
unitRowStart = ( r * ( MI_SIZE >> subY ) +	
unitSize - 1 ) / unitSize	
unitRowEnd = Min( unitRows, ( (r + h) * ( MI_SIZE >> subY ) +	
unitSize - 1 ) / unitSize)	
if ( use_superres ) {	
numerator = (MI_SIZE >> subX) * SuperresDenom	
denominator = unitSize * SUPERRES_NUM	
} else {	
numerator = MI_SIZE >> subX	
denominator = unitSize	
}	
unitColStart = ( c * numerator + denominator - 1 ) / denominator	
unitColEnd = Min( unitCols, ( (c + w) * numerator +	
denominator - 1 ) / denominator)	
for ( unitRow = unitRowStart; unitRow < unitRowEnd; unitRow++ ) {	
for ( unitCol = unitColStart; unitCol < unitColEnd; unitCol++ ) {	
read_lr_unit(plane, unitRow, unitCol)	
}	
}	
}	
}	

where count\_units\_in\_frame is a function specified as:

```
count_units_in_frame(unitSize, tileSize) {
    return Max((tileSize + (unitSize >> 1)) / unitSize, 1)
}
```

## 5.11.58. Read loop restoration unit syntax

read_lr_unit(plane, unitRow, unitCol) {	Type
if ( FrameRestorationType[ plane ] == RESTORE_WIENER ) {	
use_wiener	S()
restoration_type = use_wiener ? RESTORE_WIENER : RESTORE_NONE	
} else if ( FrameRestorationType[ plane ] == RESTORE_SGRPROJ ) {	
use_sgrproj	S()
restoration_type = use_sgrproj ? RESTORE_SGRPROJ : RESTORE_NONE	
} else {	
restoration_type	S()
}	
LrType[ plane ][ unitRow ][ unitCol ] = restoration_type	
if ( restoration_type == RESTORE_WIENER ) {	
for ( pass = 0; pass < 2; pass++ ) {	
if ( plane ) {	
firstCoeff = 1	

LrWiener[ plane ]	
[ unitRow ][ unitCol ][ pass ][0] = 0	
} else {	
firstCoeff = 0	
}	
for ( j = firstCoeff; j < 3; j++ ) {	
min = Wiener_Taps_Min[ j ]	
max = Wiener_Taps_Max[ j ]	
k = Wiener_Taps_K[ j ]	
v = decode_signed_subexp_with_ref_bool(	
min, max + 1, k, RefLrWiener[ plane ][ pass ][ j ] )	
LrWiener[ plane ]	
[ unitRow ][ unitCol ][ pass ][ j ] = v	
RefLrWiener[ plane ][ pass ][ j ] = v	
}	
}	
} else if ( restoration_type == RESTORE_SGRPROJ ) {	
lr_sgr_set	L( SGRPROJ_PARAMS_BITS )
LrSgrSet[ plane ][ unitRow ][ unitCol ] = lr_sgr_set	
for ( i = 0; i < 2; i++ ) {	
radius = Sgr_Params[ lr_sgr_set ][ i * 2 ]	
min = Sgrproj_Xqd_Min[i]	
max = Sgrproj_Xqd_Max[i]	
if ( radius ) {	
v = decode_signed_subexp_with_ref_bool(	
min, max + 1, SGRPROJ_PRJ_SUBEXP_K,	
RefSgrXqd[ plane ][ i ] )	
} else {	
v = 0	
if ( i == 1 ) {	
v = Clip3( min, max, (1 << SGRPROJ_PRJ_BITS) -	
RefSgrXqd[ plane ][ 0 ] )	
}	
}	
LrSgrXqd[ plane ][ unitRow ][ unitCol ][ i ] = v	
RefSgrXqd[ plane ][ i ] = v	
}	
}	
}	

where Wiener\_Taps\_Min, Wiener\_Taps\_Max, Wiener\_Taps\_K, Sgrproj\_Xqd\_Min, and Sgrproj\_Xqd\_Max are constant lookup tables:

```

Wiener_Taps_Min[3] = { -5, -23, -17 }
Wiener_Taps_Max[3] = { 10, 8, 46 }
Wiener_Taps_K[3] = { 1, 2, 3 }

Sgrproj_Xqd_Min[2] = { -96, -32 }
Sgrproj_Xqd_Max[2] = { 31, 95 }

```

Sgr\_Params is a constant lookup table defined in [section 7.17.3](#) and decode\_signed\_subexp\_with\_ref\_bool is a function specified as follows:

decode_signed_subexp_with_ref_bool( low, high, k, r ) {	Type
x = decode_unsigned_subexp_with_ref_bool(high - low, k, r - low)	
return x + low	
}	
decode_unsigned_subexp_with_ref_bool( mx, k, r ) {	
v = decode_subexp_bool( mx, k )	
if ( (r << 1) <= mx ) {	
return inverse_recenter(r, v)	
} else {	
return mx - 1 - inverse_recenter(mx - 1 - r, v)	
}	
}	
decode_subexp_bool( numSyms, k ) {	
i = 0	
mk = 0	
while ( 1 ) {	
b2 = i ? k + i - 1 : k	
a = 1 << b2	
if ( numSyms <= mk + 3 * a ) {	
subexp_unif_bools	NS(numSyms - mk)
return subexp_unif_bools + mk	
} else {	
subexp_more_bools	L(1)
if ( subexp_more_bools ) {	
i++	
mk += a	
} else {	
subexp_bools	L(b2)
return subexp_bools + mk	
}	
}	
}	
}	



**Note:** The `decode_signed_subexp_with_ref_bool` function is the same as the `decode_signed_subexp_with_ref` function except that the bits used to represent the symbol are arithmetic coded instead of being read directly from the bitstream.

## 5.12. Tile list OBU syntax

### 5.12.1. General tile list OBU syntax

tile_list_obu( ) {	Type
output_frame_width_in_tiles_minus_1	f(8)
output_frame_height_in_tiles_minus_1	f(8)
tile_count_minus_1	f(16)
for ( tile = 0; tile <= tile_count_minus_1; tile++ )	
tile_list_entry( )	
}	

### 5.12.2. Tile list entry syntax

tile_list_entry( ) {	Type
anchor_frame_idx	f(8)
anchor_tile_row	f(8)
anchor_tile_col	f(8)
tile_data_size_minus_1	f(16)
N = 8 * (tile_data_size_minus_1 + 1)	
coded_tile_data	f(N)
}	

## 6. Syntax structures semantics

### 6.1. General

This section specifies the meaning of the syntax elements read in the syntax structures.

Important variables and function calls are also described.

### 6.2. OBU semantics

#### 6.2.1. General OBU semantics

An ordered series of OBUs is presented to the decoding process. Each OBU is given to the decoding process as a string of bytes along with a variable `sz` that identifies the total number of bytes in the OBU.

If the syntax element `obu_has_size_field` (in the OBU header) is equal to 1, then the variable `sz` will be unused and does not have to be provided.

**obu\_size** contains the size in bytes of the OBU not including the bytes within `obu_header` or the `obu_size` syntax element.

Methods of framing the OBUs (i.e. of identifying the series of OBUs and their size and payload data) in a delivery or container format may be established in a manner outside the scope of this Specification. One simple method is described in Annex B.

OBU data starts on the first (most significant) bit and ends on the last bit of the given bytes. The payload of an OBU lies between the first bit of the given bytes and the last bit before the first trailing bit. Trailing bits are always present, unless the OBU consists of only the header. Trailing bits achieve byte alignment when the payload of an OBU is not byte aligned. The trailing bits may also be used for additional byte padding, and if used are taken into account in the `sz` value. In all cases, the pattern used for the trailing bits guarantees that all OBUs (except header-only OBUs) end with the same pattern: one bit set to one, optionally followed by zeros.

**Note:** As a validity check for malformed encoded data and for operation in environments in which losses and errors can occur, decoders may detect an error if the end of the parsed data is not directly followed by the correct trailing bits pattern or if the parsing of the OBU header and payload leads to the consumption of bits within the trailing bits (except for tile group data which is allowed to read a small distance into the trailing bits as described in section 8.2.4).

**drop\_obu( )** is a function call that indicates when the decoding process should ignore an OBU because it is not contained in the selected operating point. When an OBU is not in the selected operating point the contents have no effect on the decoding process.

When this function is called, the bitstream position indicator should be advanced by `obu_size * 8` bits.

## 6.2.2. OBU header semantics

OBUs are structured with a header and a payload. The header identifies the type of the payload using the `obu_type` header parameter.

**`obu_forbidden_bit`** must be set to 0.

**Note:** This ensures that MPEG2 transport is possible by preventing emulation of MPEG2 transport stream ids.

**`obu_type`** specifies the type of data structure contained in the OBU payload:

<b><code>obu_type</code></b>	<b>Name of <code>obu_type</code></b>
0	Reserved
1	OBU_SEQUENCE_HEADER
2	OBU_TEMPORAL_DELIMITER
3	OBU_FRAME_HEADER
4	OBU_TILE_GROUP
5	OBU_METADATA
6	OBU_FRAME
7	OBU_REDUNDANT_FRAME_HEADER
8	OBU_TILE_LIST
9-14	Reserved
15	OBU_PADDING

Reserved units are for future use and shall be ignored by AV1 decoder.

**`obu_extension_flag`** indicates if the optional `obu_extension_header` is present.

**`obu_has_size_field`** equal to 1 indicates that the `obu_size` syntax element will be present. `obu_has_size_field` equal to 0 indicates that the `obu_size` syntax element will not be present.

**`obu_reserved_1bit`** must be set to 0. The value is ignored by a decoder.

## 6.2.3. OBU extension header semantics

**`temporal_id`** specifies the temporal level of the data contained in the OBU.

**`spatial_id`** specifies the spatial level of the data contained in the OBU.

**Note:** The term “spatial” refers to the fact that the enhancement here occurs in the spatial dimension: either as an increase in spatial resolution, or an increase in spatial fidelity (increased SNR).

Tile group OBU data associated with `spatial_id` and `temporal_id` equal to 0 are referred to as the base layer, whereas tile group OBU data that are associated with `spatial_id` greater than 0 or `temporal_id` greater than 0 are referred to as enhancement layer(s).

Coded video data of a temporal level with `temporal_id` T and spatial level with `spatial_id` S are only allowed to reference previously coded video data of `temporal_id` T' and `spatial_id` S', where  $T' \leq T$  and  $S' \leq S$ .

**extension\_header\_reserved\_3bits** must be set to 0. The value is ignored by a decoder.

## 6.2.4. Trailing bits semantics

**Note:** Tile group OBUs and frame OBUs do end with trailing bits, but for these two cases, the trailing bits are consumed by the `exit_symbol` process.

**trailing\_one\_bit** shall be equal to 1.

When the syntax element `trailing_one_bit` is read, it is a requirement that `nbBits` is greater than zero.

**trailing\_zero\_bit** shall be equal to 0 and is inserted into the bitstream to align the bit position to a multiple of 8 bits and add optional zero padding bytes to the OBU.

## 6.2.5. Byte alignment semantics

**zero\_bit** shall be equal to 0 and is inserted into the bitstream to align the bit position to a multiple of 8 bits.

## 6.3. Reserved OBU semantics

The reserved OBU allows the extension of this specification with additional OBU types in a way that allows older decoders to ignore them.

## 6.4. Sequence header OBU semantics

### 6.4.1. General sequence header OBU semantics

**seq\_profile** specifies the features that can be used in the coded video sequence.

<b>seq_profile</b>	<b>Bit depth</b>	<b>Monochrome support</b>	<b>Chroma subsampling</b>
0	8 or 10	Yes	YUV 4:2:0
1	8 or 10	No	YUV 4:4:4
2	8 or 10	Yes	YUV 4:2:2

seq_profile	Bit depth	Monochrome support	Chroma subsampling
2	12	Yes	YUV 4:2:0, YUV 4:2:2, YUV 4:4:4

It is a requirement of bitstream conformance that seq\_profile is not greater than 2 (values 3 to 7 are reserved).

Monochrome can only be signaled when seq\_profile is equal to 0 or 2.

AV1 profiles are defined in Annex A.

**still\_picture** equal to 1 specifies that the coded video sequence contains only one coded frame. still\_picture equal to 0 specifies that the coded video sequence contains one or more coded frames.

**reduced\_still\_picture\_header** specifies that the syntax elements not needed by a still picture are omitted.

If reduced\_still\_picture\_header is equal to 1, it is a requirement of bitstream conformance that still\_picture is equal to 1.

**Note:** It is allowed to have still\_picture equal to 1 and reduced\_still\_picture\_header equal to 0. This allows a video frame to be converted to a still picture by changing a single bit.

**timing\_info\_present\_flag** specifies whether timing info is present in the coded video sequence.

**decoder\_model\_info\_present\_flag** specifies whether decoder model information is present in the coded video sequence.

**initial\_display\_delay\_present\_flag** specifies whether initial display delay information is present in the coded video sequence.

**operating\_points\_cnt\_minus\_1** indicates the number of operating points minus 1 present in the coded video sequence.

An operating point specifies which spatial and temporal layers should be decoded.

**operating\_point\_idc[ i ]** contains a bitmask that indicates which spatial and temporal layers should be decoded for operating point i. Bit k is equal to 1 if temporal layer k should be decoded (for k between 0 and 7). Bit j+8 is equal to 1 if spatial layer j should be decoded (for j between 0 and 3).

However, if operating\_point\_idc[ i ] is equal to 0 then the coded video sequence has no scalability information in OBU extension headers and the operating point applies to the entire coded video sequence. This means that all OBUs must be decoded.

It is a requirement of bitstream conformance that operating\_point\_idc[ i ] is not equal to operating\_point\_idc[ j ] for j = 0..(i - 1).

**Note:** This constraint means it is not allowed for two operating points to have the same value of operating\_point\_idc.

If `operating_point_idc[ op ]` is not equal to 0 for any value of `op` from 0 to `operating_points_cnt_minus_1`, it is a requirement of bitstream conformance that `obu_extension_flag` is equal to 1.

**`seq_level_idx[ i ]`** specifies the level that the coded video sequence conforms to when operating point `i` is selected.

**Note:** Encoders should select the lowest level that is satisfied by the operating point to maximize the number of decoders that can decode the stream, but this is not a requirement of bitstream conformance.

**`seq_tier[ i ]`** specifies the tier that the coded video sequence conforms to when operating point `i` is selected.

**`decoder_model_present_for_this_op[ i ]`** equal to one indicates that there is a decoder model associated with operating point `i`. `decoder_model_present_for_this_op[ i ]` equal to zero indicates that there is not a decoder model associated with operating point `i`.

**`initial_display_delay_present_for_this_op[ i ]`** equal to 1 indicates that `initial_display_delay_minus_1` is specified for operating point `i`. `initial_display_delay_present_for_this_op[ i ]` equal to 0 indicates that `initial_display_delay_minus_1` is not specified for operating point `i`.

**`initial_display_delay_minus_1[ i ]`** plus 1 specifies, for operating point `i`, the number of decoded frames that should be present in the buffer pool before the first presentable frame is displayed. This will ensure that all presentable frames in the sequence can be decoded at or before the time that they are scheduled for display. If not signaled then `initial_display_delay_minus_1[ i ] = BufferPoolMaxSize - 1`.

**`choose_operating_point( )`** is a function call that indicates that the operating point should be selected.

The implementation of this function depends on the capabilities of the chosen implementation. The order of operating points indicates the preferred order for producing an output: a decoder should select the earliest operating point in the list that meets its decoding capabilities as expressed by the level associated with each operating point.

A decoder must return a value from `choose_operating_point` between 0 and `operating_points_cnt_minus_1`, or abandon the decoding process if no level within the decoder's capabilities can be found.

**Note:** To help with conformance testing, decoders may allow the operating point to be explicitly signaled by external means.

**`OperatingPointIdc`** specifies the value of `operating_point_idc` for the selected operating point.

It is a requirement of bitstream conformance that if `OperatingPointIdc` is equal to 0, then `obu_extension_flag` is equal to 0 for all OBUs that follow this sequence header until the next sequence header.

**`frame_width_bits_minus_1`** specifies the number of bits minus 1 used for transmitting the frame width syntax elements.

**`frame_height_bits_minus_1`** specifies the number of bits minus 1 used for transmitting the frame height syntax elements.

**max\_frame\_width\_minus\_1** specifies the maximum frame width minus 1 for the frames represented by this sequence header.

**max\_frame\_height\_minus\_1** specifies the maximum frame height minus 1 for the frames represented by this sequence header.

**frame\_id\_numbers\_present\_flag** specifies whether frame id numbers are present in the coded video sequence.

**Note:** The frame id numbers (represented in `display_frame_id`, `current_frame_id`, and `RefFrameId[i]`) are not needed by the decoding process, but allow decoders to spot when frames have been missed and take an appropriate action.

**additional\_frame\_id\_length\_minus\_1** is used to calculate the number of bits used to encode the `frame_id` syntax element.

**delta\_frame\_id\_length\_minus\_2** specifies the number of bits minus 2 used to encode `delta_frame_id` syntax elements.

**use\_128x128\_superblock**, when equal to 1, indicates that superblocks contain 128x128 luma samples. When equal to 0, it indicates that superblocks contain 64x64 luma samples. (The number of contained chroma samples depends on `subsampling_x` and `subsampling_y`.)

**enable\_filter\_intra** equal to 1 specifies that the `use_filter_intra` syntax element may be present. `enable_filter_intra` equal to 0 specifies that the `use_filter_intra` syntax element will not be present.

**enable\_intra\_edge\_filter** specifies whether the intra edge filtering process should be enabled.

**enable\_interintra\_compound** equal to 1 specifies that the mode info for inter blocks may contain the syntax element `interintra`. `enable_interintra_compound` equal to 0 specifies that the syntax element `interintra` will not be present.

**enable\_masked\_compound** equal to 1 specifies that the mode info for inter blocks may contain the syntax element `compound_type`. `enable_masked_compound` equal to 0 specifies that the syntax element `compound_type` will not be present.

**enable\_warped\_motion** equal to 1 indicates that the `allow_warped_motion` syntax element may be present. `enable_warped_motion` equal to 0 indicates that the `allow_warped_motion` syntax element will not be present.

**enable\_order\_hint** equal to 1 indicates that tools based on the values of order hints may be used. `enable_order_hint` equal to 0 indicates that tools based on order hints are disabled.

**enable\_dual\_filter** equal to 1 indicates that the inter prediction filter type may be specified independently in the horizontal and vertical directions. If the flag is equal to 0, only one filter type may be specified, which is then used in both directions.

**enable\_jnt\_comp** equal to 1 indicates that the distance weights process may be used for inter prediction.

**enable\_ref\_frame\_mvs** equal to 1 indicates that the `use_ref_frame_mvs` syntax element may be present. `enable_ref_frame_mvs` equal to 0 indicates that the `use_ref_frame_mvs` syntax element will not be present.

**seq\_choose\_screen\_content\_tools** equal to 0 indicates that the **seq\_force\_screen\_content\_tools** syntax element will be present. **seq\_choose\_screen\_content\_tools** equal to 1 indicates that **seq\_force\_screen\_content\_tools** should be set equal to **SELECT\_SCREEN\_CONTENT\_TOOLS**.

**seq\_force\_screen\_content\_tools** equal to **SELECT\_SCREEN\_CONTENT\_TOOLS** indicates that the **allow\_screen\_content\_tools** syntax element will be present in the frame header. Otherwise, **seq\_force\_screen\_content\_tools** contains the value for **allow\_screen\_content\_tools**.

**seq\_choose\_integer\_mv** equal to 0 indicates that the **seq\_force\_integer\_mv** syntax element will be present. **seq\_choose\_integer\_mv** equal to 1 indicates that **seq\_force\_integer\_mv** should be set equal to **SELECT\_INTEGER\_MV**.

**seq\_force\_integer\_mv** equal to **SELECT\_INTEGER\_MV** indicates that the **force\_integer\_mv** syntax element will be present in the frame header (providing **allow\_screen\_content\_tools** is equal to 1). Otherwise, **seq\_force\_integer\_mv** contains the value for **force\_integer\_mv**.

**order\_hint\_bits\_minus\_1** is used to compute **OrderHintBits**.

**OrderHintBits** specifies the number of bits used for the **order\_hint** syntax element.

**enable\_superres** equal to 1 specifies that the **use\_superres** syntax element will be present in the uncompressed header. **enable\_superres** equal to 0 specifies that the **use\_superres** syntax element will not be present (instead **use\_superres** will be set to 0 in the uncompressed header without being read).

**Note:** It is allowed to set **enable\_superres** equal to 1 even when **use\_superres** is not equal to 1 for any frame in the coded video sequence.

**enable\_cdef** equal to 1 specifies that cdef filtering may be enabled. **enable\_cdef** equal to 0 specifies that cdef filtering is disabled.

**Note:** It is allowed to set **enable\_cdef** equal to 1 even when cdef filtering is not used on any frame in the coded video sequence.

**enable\_restoration** equal to 1 specifies that loop restoration filtering may be enabled. **enable\_restoration** equal to 0 specifies that loop restoration filtering is disabled.

**Note:** It is allowed to set **enable\_restoration** equal to 1 even when loop restoration is not used on any frame in the coded video sequence.

**film\_grain\_params\_present** specifies whether film grain parameters are present in the coded video sequence.

## 6.4.2. Color config semantics

**high\_bitdepth** and **twelve\_bit** are syntax elements which, together with **seq\_profile**, determine the bit depth.



**mono\_chrome** equal to 1 indicates that the video does not contain U and V color planes. **mono\_chrome** equal to 0 indicates that the video contains Y, U, and V color planes.

**color\_description\_present\_flag** equal to 1 specifies that **color\_primaries**, **transfer\_characteristics**, and **matrix\_coefficients** are present. **color\_description\_present\_flag** equal to 0 specifies that **color\_primaries**, **transfer\_characteristics** and **matrix\_coefficients** are not present.

**color\_primaries** is an integer that is defined by the “Color primaries” section of ISO/IEC 23091-4/ITU-T H.273.

<b>color_primaries</b>	<b>Name of color primaries</b>	<b>Description</b>
1	CP_BT_709	BT.709
2	CP_UNSPECIFIED	Unspecified
4	CP_BT_470_M	BT.470 System M (historical)
5	CP_BT_470_B_G	BT.470 System B, G (historical)
6	CP_BT_601	BT.601
7	CP_SMPTE_240	SMPTE 240
8	CP_GENERIC_FILM	Generic film (color filters using illuminant C)
9	CP_BT_2020	BT.2020, BT.2100
10	CP_XYZ	SMPTE 428 (CIE 1921 XYZ)
11	CP_SMPTE_431	SMPTE RP 431-2
12	CP_SMPTE_432	SMPTE EG 432-1
22	CP_EBU_3213	EBU Tech. 3213-E

**transfer\_characteristics** is an integer that is defined by the “Transfer characteristics” section of ISO/IEC 23091-4/ITU-T H.273.

<b>transfer_characteristics</b>	<b>Name of transfer characteristics</b>	<b>Description</b>
0	TC_RESERVED_0	For future use
1	TC_BT_709	BT.709
2	TC_UNSPECIFIED	Unspecified
3	TC_RESERVED_3	For future use
4	TC_BT_470_M	BT.470 System M (historical)
5	TC_BT_470_B_G	BT.470 System B, G (historical)
6	TC_BT_601	BT.601

<b>transfer_characteristics</b>	<b>Name of transfer characteristics</b>	<b>Description</b>
7	TC_SMPTE_240	SMPTE 240 M
8	TC_LINEAR	Linear
9	TC_LOG_100	Logarithmic (100 : 1 range)
10	TC_LOG_100_SQRT10	Logarithmic (100 * Sqrt(10) : 1 range)
11	TC_IEC_61966	IEC 61966-2-4
12	TC_BT_1361	BT.1361
13	TC_SRGB	sRGB or sYCC
14	TC_BT_2020_10_BIT	BT.2020 10-bit systems
15	TC_BT_2020_12_BIT	BT.2020 12-bit systems
16	TC_SMPTE_2084	SMPTE ST 2084, ITU BT.2100 PQ
17	TC_SMPTE_428	SMPTE ST 428
18	TC_HLG	BT.2100 HLG, ARIB STD-B67

**matrix\_coefficients** is an integer that is defined by the “Matrix coefficients” section of ISO/IEC 23091-4/ITU-T H.273.

<b>matrix_coefficients</b>	<b>Name of matrix coefficients</b>	<b>Description</b>
0	MC_IDENTITY	Identity matrix
1	MC_BT_709	BT.709
2	MC_UNSPECIFIED	Unspecified
3	MC_RESERVED_3	For future use
4	MC_FCC	US FCC 73.628
5	MC_BT_470_B_G	BT.470 System B, G (historical)
6	MC_BT_601	BT.601
7	MC_SMPTE_240	SMPTE 240 M
8	MC_SMPTE_YCGCO	YCgCo
9	MC_BT_2020_NCL	BT.2020 non-constant luminance, BT.2100 YCbCr
10	MC_BT_2020_CL	BT.2020 constant luminance
11	MC_SMPTE_2085	SMPTE ST 2085 YDzDx
12	MC_CHROMAT_NCL	Chromaticity-derived non-constant luminance
13	MC_CHROMAT_CL	Chromaticity-derived constant luminance

<b>matrix_coefficients</b>	<b>Name of matrix coefficients</b>	<b>Description</b>
14	MC_ICTCP	BT.2100 ICtCp

**color\_range** is a binary value that is associated with the VideoFullRangeFlag variable specified in ISO/IEC 23091-4/ITU-T H.273. color range equal to 0 shall be referred to as the studio swing representation and color range equal to 1 shall be referred to as the full swing representation for all intents relating to this specification.

**Note:** Note that this specification does not enforce the range when signaled as Studio swing. Therefore the application should perform additional clamping and color conversion operations according to the specified range.

**subsampling\_x**, **subsampling\_y** specify the chroma subsampling format:

<b>subsampling_x</b>	<b>subsampling_y</b>	<b>mono_chrome</b>	<b>Description</b>
0	0	0	YUV 4:4:4
1	0	0	YUV 4:2:2
1	1	0	YUV 4:2:0
1	1	1	Monochrome 4:0:0

If **matrix\_coefficients** is equal to MC\_IDENTITY, it is a requirement of bitstream conformance that **subsampling\_x** is equal to 0 and **subsampling\_y** is equal to 0.

**chroma\_sample\_position** specifies the sample position for subsampled streams:

<b>chroma_sample_position</b>	<b>Name of chroma sample position</b>	<b>Description</b>
0	CSP_UNKNOWN	Unknown (in this case the source video transfer function must be signaled outside the AV1 bitstream)
1	CSP_VERTICAL	Horizontally co-located with (0, 0) luma sample, vertical position in the middle between two luma samples
2	CSP_COLOCATED	co-located with (0, 0) luma sample
3	CSP_RESERVED	

**separate\_uv\_delta\_q** equal to 1 indicates that the U and V planes may have separate delta quantizer values.

**separate\_uv\_delta\_q** equal to 0 indicates that the U and V planes will share the same delta quantizer value.

### 6.4.3. Timing info semantics

**num\_units\_in\_display\_tick** is the number of time units of a clock operating at the frequency `time_scale` Hz that corresponds to one increment of a clock tick counter. A display clock tick, in seconds, is equal to `num_units_in_display_tick` divided by `time_scale`:

$$\text{DispCT} = \text{num\_units\_in\_display\_tick} \div \text{time\_scale}$$

**Note:** The  $\div$  operator represents standard mathematical division (in contrast to the  $/$  operator which represents integer division).

It is a requirement of bitstream conformance that `num_units_in_display_tick` is greater than 0.

**time\_scale** is the number of time units that pass in one second.

It is a requirement of bitstream conformance that `time_scale` is greater than 0.

**equal\_picture\_interval** equal to 1 indicates that pictures should be displayed according to their output order with the number of ticks between two consecutive pictures (without dropping frames) specified by `num_ticks_per_picture_minus_1` + 1. `equal_picture_interval` equal to 0 indicates that the interval between two consecutive pictures is not specified.

**num\_ticks\_per\_picture\_minus\_1** plus 1 specifies the number of clock ticks corresponding to output time between two consecutive pictures in the output order.

It is a requirement of bitstream conformance that the value of `num_ticks_per_picture_minus_1` shall be in the range of 0 to  $(1 \ll 32) - 2$ , inclusive.

**Note:** The frame rate, when specified explicitly, applies to the top temporal layer of the bitstream. If bitstream is expected to be manipulated, e.g. by intermediate network elements, then the resulting frame rate may not match the specified one. In this case, an encoder is advised to use explicit time codes or some mechanisms that convey picture timing information outside the bitstream.

### 6.4.4. Decoder model info semantics

**buffer\_delay\_length\_minus\_1** plus 1 specifies the length of the `decoder_buffer_delay` and the `encoder_buffer_delay` syntax elements, in bits.

**num\_units\_in\_decoding\_tick** is the number of time units of a decoding clock operating at the frequency `time_scale` Hz that corresponds to one increment of a clock tick counter:

$$\text{DecCT} = \text{num\_units\_in\_decoding\_tick} \div \text{time\_scale}$$

**Note:** The  $\div$  operator represents standard mathematical division (in contrast to the  $/$  operator which represents integer division).

`num_units_in_decoding_tick` shall be greater than 0. `DecCT` represents the expected time to decode a single frame or a common divisor of the expected times to decode frames of different sizes and dimensions present in the coded video sequence.

`buffer_removal_time_length_minus_1` plus 1 specifies the length of the `buffer_removal_time` syntax element, in bits.

`frame_presentation_time_length_minus_1` plus 1 specifies the length of the `frame_presentation_time` syntax element, in bits.

## 6.4.5. Operating parameters info semantics

`decoder_buffer_delay[ op ]` specifies the time interval between the arrival of the first bit in the smoothing buffer and the subsequent removal of the data that belongs to the first coded frame for operating point `op`, measured in units of 1/90000 seconds. The length of `decoder_buffer_delay` is specified by `buffer_delay_length_minus_1 + 1`, in bits.

`encoder_buffer_delay[ op ]` specifies, in combination with `decoder_buffer_delay[ op ]` syntax element, the first bit arrival time of frames to be decoded to the smoothing buffer. `encoder_buffer_delay` is measured in units of 1/90000 seconds.

For a video sequence that includes one or more random access points the sum of `decoder_buffer_delay` and `encoder_buffer_delay` shall be kept constant.

`low_delay_mode_flag[ op ]` equal to 1 indicates that the smoothing buffer operates in low-delay mode for operating point `op`. In low-delay mode late decode times and buffer underflow are both permitted. `low_delay_mode_flag[ op ]` equal to 0 indicates that the smoothing buffer operates in strict mode, where buffer underflow is not allowed.

## 6.5. Temporal delimiter OBU semantics

**SeenFrameHeader** is a variable used to mark whether the frame header for the current frame has been received. It is initialized to zero.

## 6.6. Padding OBU semantics

Multiple padding units can be present, each padding with an arbitrary number of bytes.

**obu\_padding\_byte** is a padding byte. Padding bytes may have arbitrary values and have no effect on the decoding process.

## 6.7. Metadata OBU semantics

### 6.7.1. General metadata OBU semantics

**metadata\_type** indicates the type of metadata:

<b>metadata_type</b>	<b>Name of metadata_type</b>
0	Reserved for AOM use
1	METADATA_TYPE_HDR_CLL
2	METADATA_TYPE_HDR_MDCV
3	METADATA_TYPE_SCALABILITY
4	METADATA_TYPE_ITUT_T35
5	METADATA_TYPE_TIMECODE
6-31	Unregistered user private
32 and greater	Reserved for AOM use

## 6.7.2. Metadata ITUT T35 semantics

**itu\_t\_t35\_country\_code** shall be a byte having a value specified as a country code by Annex A of Recommendation ITU-T T.35.

**itu\_t\_t35\_country\_code\_extension\_byte** shall be a byte having a value specified as a country code by Annex B of Recommendation ITU-T T.35.

**itu\_t\_t35\_payload\_bytes** shall be bytes containing data registered as specified in Recommendation ITU-T T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the **itu\_t\_t35\_payload\_bytes**, in the format specified by the Administration that issued the terminal provider code. Any remaining bytes in **itu\_t\_t35\_payload\_bytes** data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

## 6.7.3. Metadata high dynamic range content light level semantics

**max\_cll** specifies the maximum content light level as specified in CEA-861.3, Appendix A.

**max\_fall** specifies the maximum frame-average light level as specified in CEA-861.3, Appendix A.

## 6.7.4. Metadata high dynamic range mastering display color volume semantics

**primary\_chromaticity\_x[i]** specifies a 0.16 fixed-point X chromaticity coordinate as defined by CIE 1931, where  $i = 0, 1, 2$  specifies Red, Green, Blue respectively.

**primary\_chromaticity\_y[i]** specifies a 0.16 fixed-point Y chromaticity coordinate as defined by CIE 1931, where  $i = 0, 1, 2$  specifies Red, Green, Blue respectively.

**white\_point\_chromaticity\_x** specifies a 0.16 fixed-point white X chromaticity coordinate as defined by CIE 1931.

**white\_point\_chromaticity\_y** specifies a 0.16 fixed-point white Y chromaticity coordinate as defined by CIE 1931.

**luminance\_max** is a 24.8 fixed-point maximum luminance, represented in candelas per square meter.

**luminance\_min** is a 18.14 fixed-point minimum luminance, represented in candelas per square meter.

## 6.7.5. Metadata scalability semantics

**scalability\_mode\_idc** indicates the picture prediction structure of the coded video sequence.

<b>scalability_mode_idc</b>	<b>Name of scalability_mode_idc</b>
0	SCALABILITY_L1T2
1	SCALABILITY_L1T3
2	SCALABILITY_L2T1
3	SCALABILITY_L2T2
4	SCALABILITY_L2T3
5	SCALABILITY_S2T1
6	SCALABILITY_S2T2
7	SCALABILITY_S2T3
8	SCALABILITY_L2T1h
9	SCALABILITY_L2T2h
10	SCALABILITY_L2T3h
11	SCALABILITY_S2T1h
12	SCALABILITY_S2T2h
13	SCALABILITY_S2T3h
14	SCALABILITY_SS
15-255	reserved

The scalability metadata provides two mechanisms for describing the underlying picture prediction structure of the bitstream:

1. Selection among a set of preconfigured structures, or modes, covering a number of cases that have found wide use in applications.
2. A facility for specifying picture prediction structures to accommodate a variety of special cases.

The preconfigured modes are described below. The mechanism for describing alternative structures is described in `scalability_structure()` below.

All predefined modes follow a dyadic, hierarchical picture prediction structure. They support up to three temporal layers, in combinations with one or two spatial layers. The second spatial layer may have twice or one and a half times the resolution of the base layer in each dimension, depending on the mode.

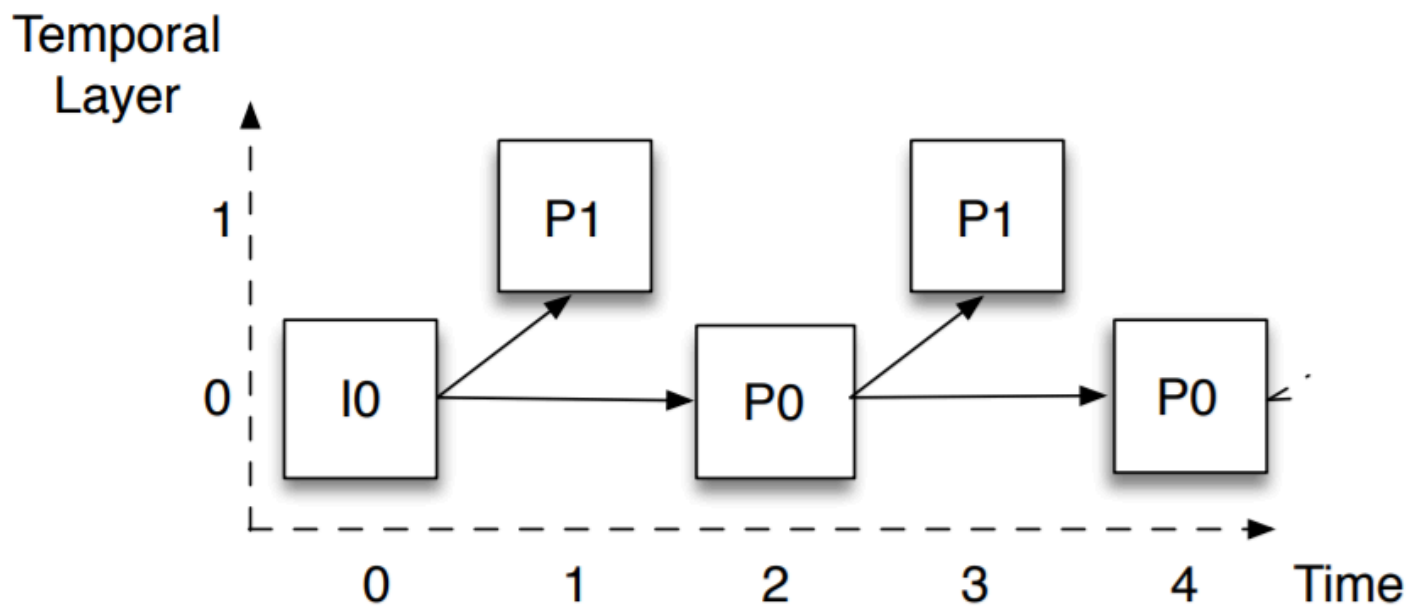
There is also support for a spatial layer that uses no inter-layer prediction (i.e., the second spatial layer does not use its corresponding base layer as a reference).

The following table lists the predefined scalability structures.

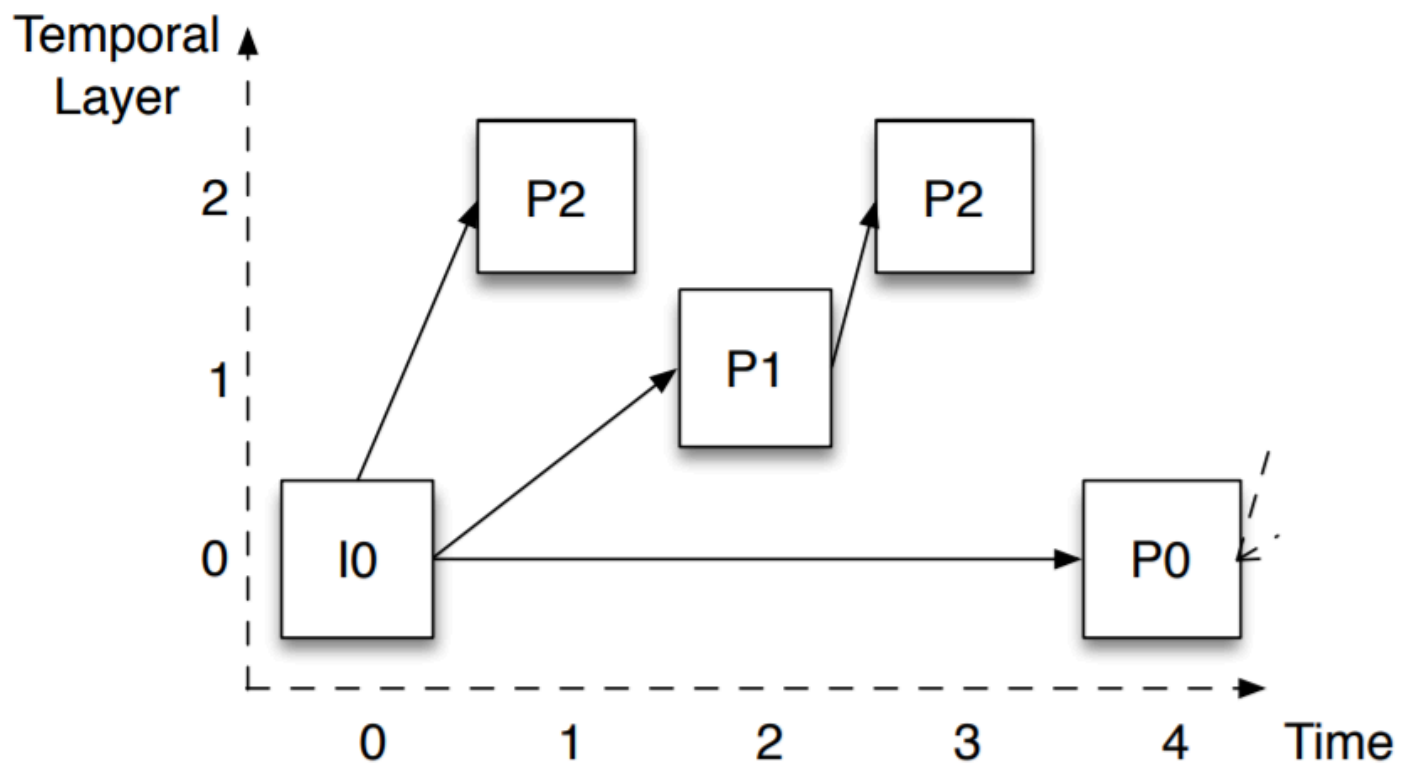
Name of scalability_mode_idc	Spatial Layers	Resolution Ratio	Temporal Layers	Inter-layer dependency
SCALABILITY_L1T2	1		2	
SCALABILITY_L1T3	1		3	
SCALABILITY_L2T1	2	2:1	1	Yes
SCALABILITY_L2T2	2	2:1	2	Yes
SCALABILITY_L2T3	2	2:1	3	Yes
SCALABILITY_S2T1	2	2:1	1	No
SCALABILITY_S2T2	2	2:1	2	No
SCALABILITY_S2T3	2	2:1	3	No
SCALABILITY_L2T1h	2	1.5:1	1	Yes
SCALABILITY_L2T2h	2	1.5:1	2	Yes
SCALABILITY_L2T3h	2	1.5:1	3	Yes
SCALABILITY_S2T1h	2	1.5:1	1	No
SCALABILITY_S2T2h	2	1.5:1	2	No
SCALABILITY_S2T3h	2	1.5:1	3	No

The following figures show the picture prediction structures for certain modes:

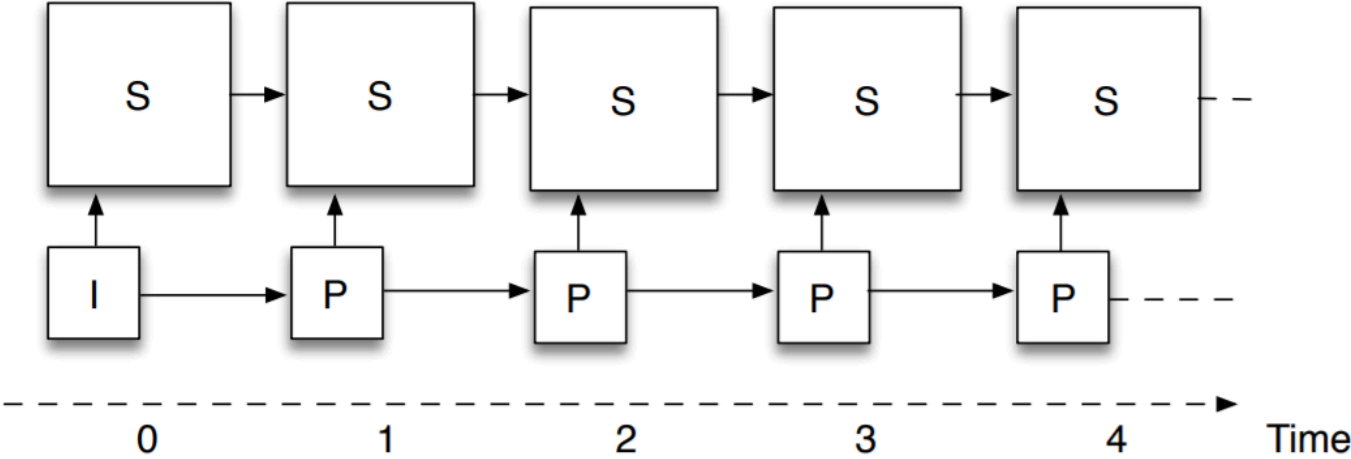




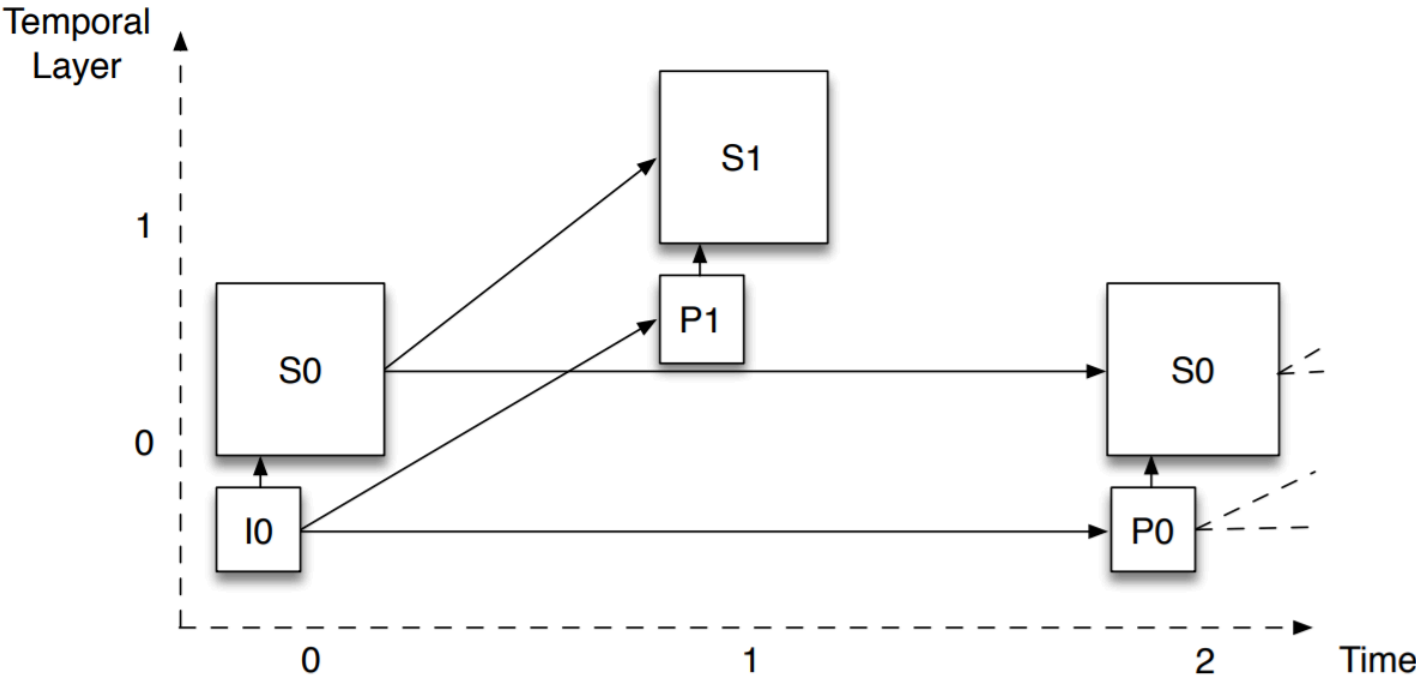
L1T2



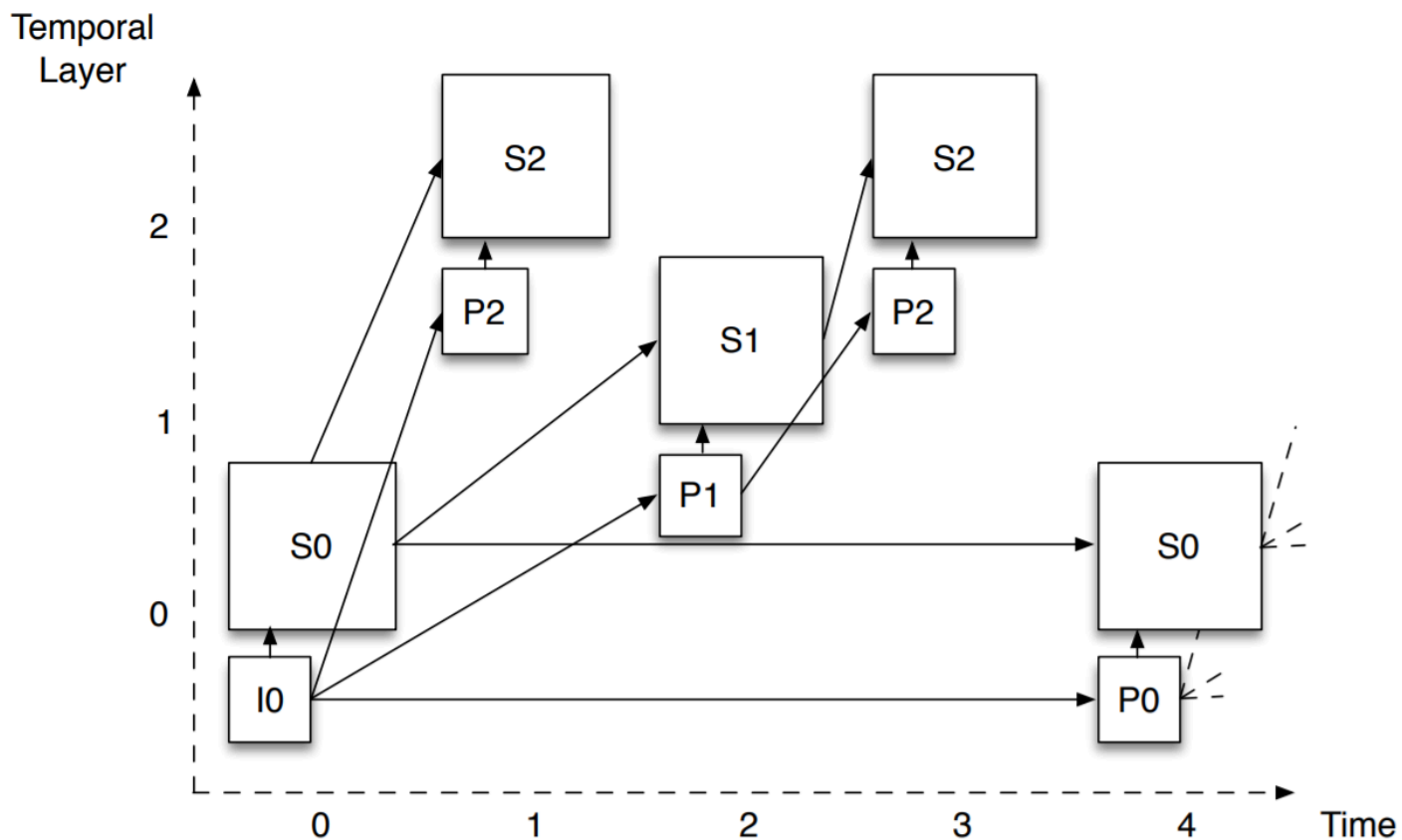
L1T3



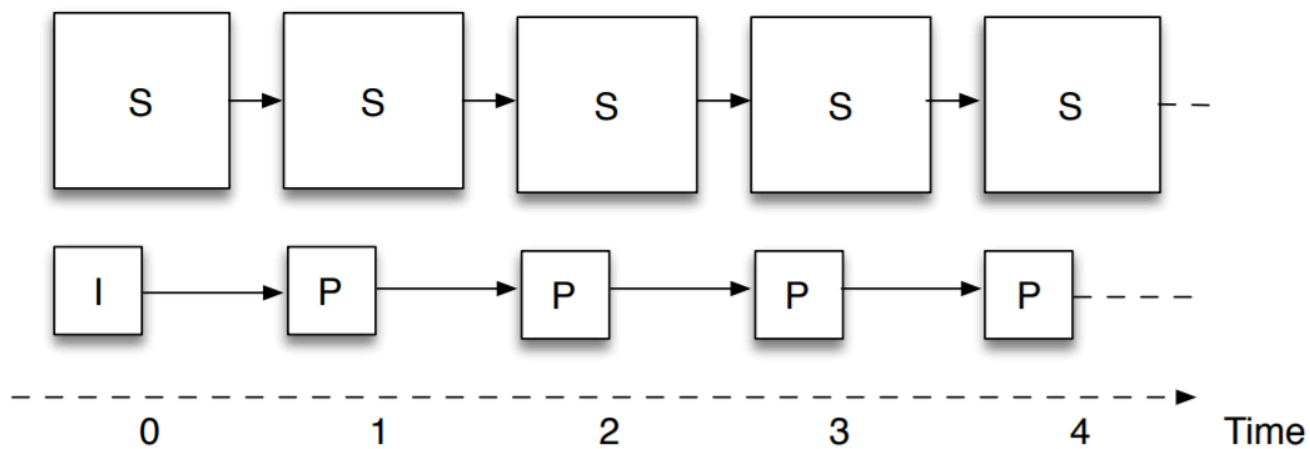
L2T1



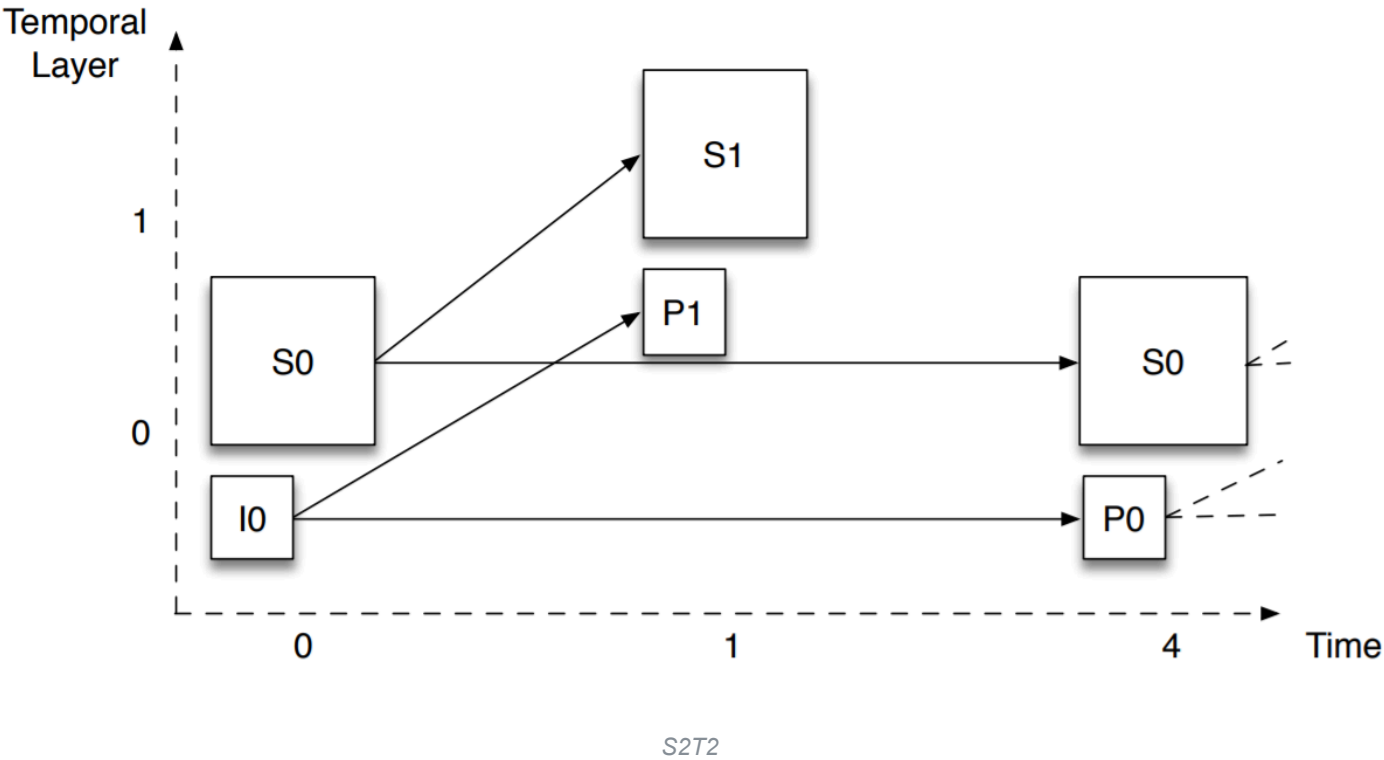
L2T2

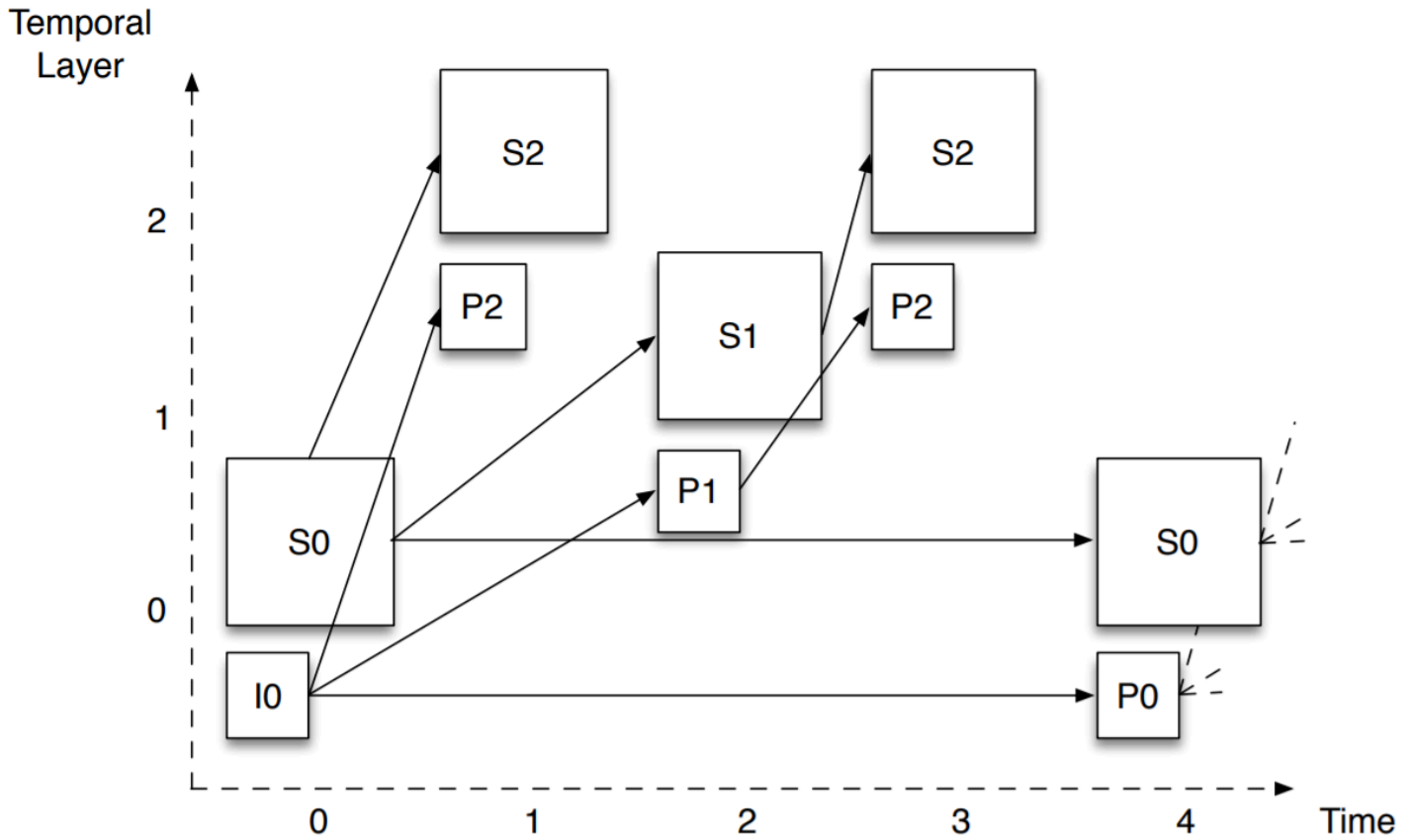


L2T3



S2T1





S2T3

## 6.7.6. Scalability structure semantics

### 6.7.6.1. General

**Note:** The `scalability_structure` is intended for use by intermediate processing entities that may perform selective layer elimination. Its presence allows these entities to know the structure of the video bitstream without have to decode individual frames. Scalability structures should be placed immediately after the sequence header so that these entities are informed of the scalability structure of the video sequence as early as possible.

**`spatial_layers_cnt_minus_1`** indicates the number of spatial layers present in the video sequence minus one.

**`spatial_layer_description_present_flag`** indicates when set to 1 that the `spatial_layer_ref_id` is present for each of the  $(\text{spatial\_layers\_cnt\_minus\_1} + 1)$  layers, or that it is not present when set to 0.

**`spatial_layer_dimensions_present_flag`** indicates when set to 1 that the `spatial_layer_max_width` and `spatial_layer_max_height` parameters are present for each of the  $(\text{spatial\_layers\_cnt\_minus\_1} + 1)$  layers, or that it they are not present when set to 0.

**`temporal_group_description_present_flag`** indicates when set to 1 that the temporal dependency information is present, or that it is not when set to 0.

**scalability\_structure\_reserved\_3bits** must be set to zero and be ignored by decoders.

**spatial\_layer\_max\_width[ i ]** specifies the maximum frame width for the frames with **spatial\_id** equal to *i*. This number must not be larger than **max\_frame\_width\_minus\_1** + 1.

**spatial\_layer\_max\_height[ i ]** specifies the maximum frame height for the frames with **spatial\_id** equal to *i*. This number must not be larger than **max\_frame\_height\_minus\_1** + 1.

**spatial\_layer\_ref\_id[ i ]** specifies the **spatial\_id** value of the frame within the current temporal unit that the frame of layer *i* uses for reference. If no frame within the current temporal unit is used for reference the value must be equal to 255.

**temporal\_group\_size** indicates the number of pictures in a temporal picture group. If the **temporal\_group\_size** is greater than 0, then the scalability structure data allows the inter-picture temporal dependency structure of the video sequence to be specified. If the **temporal\_group\_size** is greater than 0, then for **temporal\_group\_size** pictures in the temporal group, each picture's temporal layer id (**temporal\_id**), switch up points (**temporal\_group\_temporal\_switching\_up\_point\_flag** and **temporal\_group\_spatial\_switching\_up\_point\_flag**), and the reference picture indices (**temporal\_group\_ref\_pic\_diff**) are specified.

The first picture specified in a temporal group must have **temporal\_id** equal to 0.

If the parameter **temporal\_group\_size** is not present or set to 0, then either there is only one temporal layer or there is no fixed inter-picture temporal dependency present going forward in the video sequence.

Note that for a given picture, all frames follow the same inter-picture temporal dependency structure.

However, the frame rate of each layer can be different from each other. The specified dependency structure in the scalability structure data must be for the highest frame rate layer.

**temporal\_group\_temporal\_id[ i ]** specifies the **temporal\_id** value for the *i*-th picture in the temporal group.

**temporal\_group\_temporal\_switching\_up\_point\_flag[ i ]** is set to 1 if subsequent (in decoding order) pictures with a **temporal\_id** higher than **temporal\_group\_temporal\_id[ i ]** do not depend on any picture preceding the current picture (in coding order) with **temporal\_id** higher than **temporal\_group\_temporal\_id[ i ]**.

**Note:** This condition ensures that switching up to a higher frame rate is possible at the current picture.

**temporal\_group\_spatial\_switching\_up\_point\_flag[ i ]** is set to 1 if spatial layers of the current picture in the temporal group (i.e., pictures with a **spatial\_id** higher than zero) do not depend on any picture preceding the current picture in the temporal group.

**temporal\_group\_ref\_cnt[ i ]** indicates the number of reference pictures used by the *i*-th picture in the temporal group.

**temporal\_group\_ref\_pic\_diff[ i ][ j ]** indicates, for the *i*-th picture in the temporal group, the temporal distance between the *i*-th picture and the *j*-th reference picture used by the *i*-th picture. The temporal distance is measured in frames, counting only frames of identical **spatial\_id** values.

**Note:** The scalability structure description does not allow different temporal prediction structures across non-temporal layers (i.e., layers with different spatial\_id values). It also only allows for a single reference picture for inter-layer prediction.

The following sections contain the value of these syntax elements for the predefined modes.

### 6.7.6.2. L1T2 (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	0
Picture	Temporal Group Description	Value
	temporal_group_size	2
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	2
1	temporal_group_temporal_id[1]	1
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[0][0]	1

### 6.7.6.3. L1T3 (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	0
Picture	Temporal Group Description	Value
	temporal_group_size	4
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1

	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	4
1	temporal_group_temporal_id[1]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[1][0]	1
2	temporal_group_temporal_id[2]	1
	temporal_group_temporal_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[2]	1
	temporal_group_ref_pic_diff[2][0]	2
3	temporal_group_temporal_id[3]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[3]	1
	temporal_group_ref_pic_diff[3][0]	1

#### 6.7.6.4. L2T1 / L2T1h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	0
Picture	Temporal Group Description	Value
	temporal_group_size	1
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1



	temporal_group_ref_pic_diff[0][0]	1
--	-----------------------------------	---

#### 6.7.6.5. L2T2 / L2T2h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	0
Picture	Temporal Group Description	Value
	temporal_group_size	2
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	2
1	temporal_group_temporal_id[1]	1
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[1][0]	1

#### 6.7.6.6. L2T3 / L2T3h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	0
Picture	Temporal Group Description	Value
	temporal_group_size	4
0	temporal_group_temporal_id[0]	0

	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	4
1	temporal_group_temporal_id[1]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[1][0]	1
2	temporal_group_temporal_id[2]	1
	temporal_group_temporal_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[2]	1
	temporal_group_ref_pic_diff[2][0]	2
3	temporal_group_temporal_id[3]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[3]	1
	temporal_group_ref_pic_diff[3][0]	1

#### 6.7.6.7. S2T1 / S2T1h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	255
Picture	Temporal Group Description	Value
	temporal_group_size	1
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0

	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	1

#### 6.7.6.8. S2T2 / S2T2h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	255
Picture	Temporal Group Description	Value
	temporal_group_size	2
0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	2
1	temporal_group_temporal_id[1]	1
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[1][0]	1

#### 6.7.6.9. S2T3 / S2T3h (Informative)

Layer	Spatial Layers Description	Value
	spatial_layers_cnt_minus_1	1
0	spatial_layer_ref_id[0]	255
1	spatial_layer_ref_id[1]	255
Picture	Temporal Group Description	Value
	temporal_group_size	4

0	temporal_group_temporal_id[0]	0
	temporal_group_temporal_switching_up_point_flag[0]	1
	temporal_group_spatial_switching_up_point_flag[0]	0
	temporal_group_ref_cnt[0]	1
	temporal_group_ref_pic_diff[0][0]	4
1	temporal_group_temporal_id[1]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[1]	1
	temporal_group_ref_pic_diff[1][0]	1
2	temporal_group_temporal_id[2]	1
	temporal_group_temporal_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[2]	1
	temporal_group_ref_pic_diff[2][0]	2
3	temporal_group_temporal_id[3]	2
	temporal_group_temporal_switching_up_point_flag[1]	1
	temporal_group_spatial_switching_up_point_flag[1]	0
	temporal_group_ref_cnt[3]	1
	temporal_group_ref_pic_diff[3][0]	1

### 6.7.7. Metadata timecode semantics

**counting\_type** specifies the method of dropping values of the `n_frames` syntax element as specified in the table below. `counting_type` should be the same for all pictures in the coded video sequence.

counting_type	Meaning
0	no dropping of <code>n_frames</code> count values and no use of <code>time_offset_value</code>
1	no dropping of <code>n_frames</code> count values
2	dropping of individual zero values of <code>n_frames</code> count
3	dropping of individual values of <code>n_frames</code> count equal to $\text{maxFps} - 1$
4	dropping of the two lowest (value 0 and 1) <code>n_frames</code> counts when <code>seconds_value</code> is equal to 0 and <code>minutes_value</code> is not an integer multiple of 10

counting_type	Meaning
5	dropping of unspecified individual n_frames count values
6	dropping of unspecified numbers of unspecified n_frames count values
7..31	reserved

**full\_timestamp\_flag** equal to 1 indicates that the seconds\_value, minutes\_value, hours\_value syntax elements will be present. full\_timestamp\_flag equal to 0 indicates that there are flags to control the presence of these syntax elements.

When timing\_info\_present\_flag is equal to 1, the contents of the clock timestamp indicate a time of origin, capture, or ideal display. This indicated time is computed as follows:

```

if ( equal_picture_interval ) {
    ticksPerPicture = num_ticks_per_picture_minus_1 + 1
} else {
    ticksPerPicture = 1
}
ss = ( ( hours_value * 60 + minutes_value ) * 60 + seconds_value )
clockTimestamp = ss * time_scale + n_frames * ticksPerPicture + time_offset_value

```

clockTimestamp is in units of clock ticks of a clock with clock frequency equal to time\_scale Hz, relative to some unspecified point in time for which clockTimestamp would be equal to 0.

**discontinuity\_flag** equal to 0 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of timestamp syntax elements in output order can be interpreted as the time difference between the times of origin or capture of the associated frames or fields. discontinuity\_flag equal to 1 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of clock timestamp syntax elements in output order should not be interpreted as the time difference between the times of origin or capture of the associated frames or fields.

When timing\_info\_present\_flag is equal to 1 and discontinuity\_flag is equal to 0, the value of clockTimestamp shall be greater than or equal to the value of clockTimestamp for the previous set of clock timestamp syntax elements in output order.

**cnt\_dropped\_flag** specifies the skipping of one or more values of n\_frames using the counting method specified by counting\_type.

**n\_frames** is used to compute clockTimestamp. When timing\_info\_present\_flag is equal to 1, n\_frames shall be less than maxFps, where maxFps is specified by  $\text{maxFps} = \text{ceil}( \text{time\_scale} / ( 2 * \text{num\_units\_in\_display\_tick} ) )$ .

**seconds\_flag** equal to 1 specifies that seconds\_value and minutes\_flag are present when full\_timestamp\_flag is equal to 0. seconds\_flag equal to 0 specifies that seconds\_value and minutes\_flag are not present.

**seconds\_value** is used to compute clockTimestamp and shall be in the range of 0 to 59. When seconds\_value is not present, its value is inferred to be equal to the value of seconds\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous seconds\_value shall have been present.

**minutes\_flag** equal to 1 specifies that minutes\_value and hours\_flag are present when full\_timestamp\_flag is equal to 0 and seconds\_flag is equal to 1. minutes\_flag equal to 0 specifies that minutes\_value and hours\_flag are not present.

**minutes\_value** specifies the value of mm used to compute clockTimestamp and shall be in the range of 0 to 59, inclusive. When minutes\_value is not present, its value is inferred to be equal to the value of minutes\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous minutes\_value shall have been present.

**hours\_flag** equal to 1 specifies that hours\_value is present when full\_timestamp\_flag is equal to 0 and seconds\_flag is equal to 1 and minutes\_flag is equal to 1.

**hours\_value** is used to compute clockTimestamp and shall be in the range of 0 to 23, inclusive. When hours\_value is not present, its value is inferred to be equal to the value of hours\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous hours\_value shall have been present.

**time\_offset\_length** greater than 0 specifies the length in bits of the time\_offset\_value syntax element. time\_offset\_length equal to 0 specifies that the time\_offset\_value syntax element is not present. time\_offset\_length should be the same for all pictures in the coded video sequence.

**time\_offset\_value** is used to compute clockTimestamp. The number of bits used to represent time\_offset\_value is equal to time\_offset\_length. When time\_offset\_value is not present, its value is inferred to be equal to 0.

## 6.8. Frame header OBU semantics

### 6.8.1. General frame header OBU semantics

It is a requirement of bitstream conformance that a sequence header OBU has been received before a frame header OBU.

**frame\_header\_copy** is a function call that indicates that a copy of the previous frame\_header\_obu should be inserted at this point.

**Note:** Bitstreams may contain several copies of the frame\_header\_obu interspersed with tile\_group\_obu to allow for greater error resilience. However, the copies must contain identical contents to the original frame\_header\_obu.

If obu\_type is equal to OBU\_FRAME\_HEADER, it is a requirement of bitstream conformance that SeenFrameHeader is equal to 0.

If obu\_type is equal to OBU\_REDUNDANT\_FRAME\_HEADER, it is a requirement of bitstream conformance that SeenFrameHeader is equal to 1.

**Note:** These requirements ensure that the first frame header for a frame has obu\_type equal to OBU\_FRAME\_HEADER, while later copies of this frame header (if present) have obu\_type equal to OBU\_REDUNDANT\_FRAME\_HEADER.

**TileNum** is a variable giving the index (zero-based) of the current tile.

**decode\_frame\_wrapup** is a function call that indicates that the decode frame wrapup process specified in [section 7.4](#) should be invoked.

## 6.8.2. Uncompressed header semantics

**show\_existing\_frame** equal to 1, indicates the frame indexed by frame\_to\_show\_map\_idx is to be output; show\_existing\_frame equal to 0 indicates that further processing is required.

If obu\_type is equal to OBU\_FRAME, it is a requirement of bitstream conformance that show\_existing\_frame is equal to 0.

**frame\_to\_show\_map\_idx** specifies the frame to be output. It is only available if show\_existing\_frame is 1.

**display\_frame\_id** provides the frame id number for the frame to output. It is a requirement of bitstream conformance that whenever display\_frame\_id is read, the value matches RefFrameId[ frame\_to\_show\_map\_idx ] (the value of current\_frame\_id at the time that the frame indexed by frame\_to\_show\_map\_idx was stored), and that RefValid[ frame\_to\_show\_map\_idx ] is equal to 1.

It is a requirement of bitstream conformance that the number of bits needed to read display\_frame\_id does not exceed 16. This is equivalent to the constraint that idLen <= 16.

**frame\_type** specifies the type of the frame:

frame_type	Name of frame_type
0	KEY_FRAME
1	INTER_FRAME
2	INTRA_ONLY_FRAME
3	SWITCH_FRAME

**show\_frame** equal to 1 specifies that this frame should be immediately output once decoded. show\_frame equal to 0 specifies that this frame should not be immediately output. (It may be output later if a later uncompressed header uses show\_existing\_frame equal to 1).

**showable\_frame** equal to 1 specifies that the frame may be output using the show\_existing\_frame mechanism. showable\_frame equal to 0 specifies that this frame will not be output using the show\_existing\_frame mechanism.

It is a requirement of bitstream conformance that when show\_existing\_frame is used to show a previous frame, that the value of showable\_frame for the previous frame was equal to 1.

It is a requirement of bitstream conformance that when `show_existing_frame` is used to show a previous frame with `RefFrameType[ frame_to_show_map_idx ]` equal to `KEY_FRAME`, that the frame is output via the `show_existing_frame` mechanism at most once.

**Note:** This requirement also forbids storing a frame with `frame_type` equal to `KEY_FRAME` into multiple reference frames and then using `show_existing_frame` for each reference frame.

`error_resilient_mode` equal to 1 indicates that error resilient mode is enabled; `error_resilient_mode` equal to 0 indicates that error resilient mode is disabled.

**Note:** Error resilient mode allows the syntax of a frame to be parsed independently of previously decoded frames.

`disable_cdf_update` specifies whether the CDF update in the symbol decoding process should be disabled.

`current_frame_id` specifies the frame id number for the current frame. Frame id numbers are additional information that do not affect the decoding process, but provide decoders with a way of detecting missing reference frames so that appropriate action can be taken.

If `frame_type` is not equal to `KEY_FRAME` or `show_frame` is equal to 0, it is a requirement of bitstream conformance that all of the following conditions are true:

- `current_frame_id` is not equal to `PrevFrameID`,
- `DiffFrameID` is less than  $1 \ll (\text{idLen} - 1)$

where `DiffFrameID` is specified as follows:

- If `current_frame_id` is greater than `PrevFrameID`, `DiffFrameID` is equal to `current_frame_id - PrevFrameID`.
- Otherwise, `DiffFrameID` is equal to  $(1 \ll \text{idLen}) + \text{current\_frame\_id} - \text{PrevFrameID}$ .

`frame_size_override_flag` equal to 0 specifies that the frame size is equal to the size in the sequence header.

`frame_size_override_flag` equal to 1 specifies that the frame size will either be specified as the size of one of the reference frames, or computed from the `frame_width_minus_1` and `frame_height_minus_1` syntax elements.

`order_hint` is used to compute `OrderHint`.

`OrderHint` specifies `OrderHintBits` least significant bits of the expected output order for this frame.

**Note:** There is no requirement that `OrderHint` should reflect the true output order. As a guideline, the motion vector prediction is expected to be more accurate if the true output order is used for frames that will be shown later. If a frame is never to be shown (e.g. it has been constructed as an average of several frames for reference purposes), the encoder is free to choose whichever value of `OrderHint` will give the best compression.



**primary\_ref\_frame** specifies which reference frame contains the CDF values and other state that should be loaded at the start of the frame.

**Note:** It is allowed for **primary\_ref\_frame** to be coded as PRIMARY\_REF\_NONE, this will cause default values to be used for the CDF values and other state.

**buffer\_removal\_time\_present\_flag** equal to 1 specifies that **buffer\_removal\_time** is present.

**buffer\_removal\_time\_present\_flag** equal to 0 specifies that **buffer\_removal\_time** is not present.

**buffer\_removal\_time[ opNum ]** specifies the frame removal time in units of DecCT clock ticks counted from the removal time of the last random access point for operating point **opNum**. **buffer\_removal\_time** is signaled as a fixed length unsigned integer with a length in bits given by **buffer\_removal\_time\_length\_minus\_1 + 1**.

**buffer\_removal\_time** is the remainder of a modulo  $1 \ll (\text{buffer\_removal\_time\_length\_minus\_1} + 1)$  counter.

**allow\_screen\_content\_tools** equal to 1 indicates that intra blocks may use palette encoding;

**allow\_screen\_content\_tools** equal to 0 indicates that palette encoding is never used.

**allow\_intrabc** equal to 1 indicates that intra block copy may be used in this frame. **allow\_intrabc** equal to 0 indicates that intra block copy is not allowed in this frame.

**Note:** intra block copy is only allowed in intra frames, and disables all loop filtering. **force\_integer\_mv** will be equal to 1 for intra frames, so only integer offsets are allowed in block copy mode.

**force\_integer\_mv** equal to 1 specifies that motion vectors will always be integers. **force\_integer\_mv** equal to 0 specifies that motion vectors can contain fractional bits.

**ref\_order\_hint[ i ]** specifies the expected output order hint for each reference frame.

**Note:** The values in the **ref\_order\_hint** array are provided to allow implementations to gracefully handle cases when some frames have been lost.

**Note:** When scalability is used, the values in **RefOrderHint** during the decode process may depend on the selected operating point.

**refresh\_frame\_flags** contains a bitmask that specifies which reference frame slots will be updated with the current frame after it is decoded.

If **frame\_type** is equal to INTRA\_ONLY\_FRAME, it is a requirement of bitstream conformance that **refresh\_frame\_flags** is not equal to 0xff.

**Note:** This restriction encourages encoders to correctly label random access points (by forcing `frame_type` to be equal to `KEY_FRAME` when an intra frame is used to reset the decoding process).

See [section 7.20](#) for details of the frame update process.

`frame_refs_short_signaling` equal to 1 indicates that only two reference frames are explicitly signaled.

`frame_refs_short_signaling` equal to 0 indicates that all reference frames are explicitly signaled.

`last_frame_idx` specifies the reference frame to use for `LAST_FRAME`.

`gold_frame_idx` specifies the reference frame to use for `GOLDEN_FRAME`.

`set_frame_refs` is a function call that indicates the conceptual point where the `ref_frame_idx` values are computed (in the case when `frame_refs_short_signaling` is equal to 1, these syntax elements are computed instead of being explicitly signaled). When this function is called, the set frame refs process specified in [section 7.8](#) is invoked.

`ref_frame_idx[ i ]` specifies which reference frames are used by inter frames. It is a requirement of bitstream conformance that `RefValid[ ref_frame_idx[ i ] ]` is equal to 1, and that the selected reference frames match the current frame in bit depth, profile, chroma subsampling, and color space.

**Note:** Syntax elements indicate a reference (such as `LAST_FRAME`, `ALTREF_FRAME`). These references are looked up in the `ref_frame_idx` array to find which reference frame should be used during inter prediction. There is no requirement that the values in `ref_frame_idx` should be distinct.

`RefFrameSignBias` specifies the intended direction of the motion vector in time for each reference frame. A sign bias equal to 0 indicates that the reference frame is a forwards reference (i.e. the reference frame is expected to be output before the current frame); a sign bias equal to 1 indicates that the reference frame is a backwards reference.

**Note:** The sign bias is just an indication that can improve the accuracy of motion vector prediction and is not constrained to reflect the actual output order of pictures.

`delta_frame_id_minus_1` is used to calculate `DeltaFrameId`.

`DeltaFrameId` specifies the distance to the frame id for the reference frame.

`RefFrameId[ i ]` specifies the frame id for each reference frame.

`expectedFrameId[ i ]` specifies the frame id for each frame used for reference. It is a requirement of bitstream conformance that whenever `expectedFrameId[ i ]` is calculated, the value matches `RefFrameId[ ref_frame_idx[ i ] ]` (this contains the value of `current_frame_id` at the time that the frame indexed by `ref_frame_idx` was stored).

`allow_high_precision_mv` equal to 0 specifies that motion vectors are specified to quarter pel precision;

`allow_high_precision_mv` equal to 1 specifies that motion vectors are specified to eighth pel precision.

**is\_motion\_mode\_switchable** equal to 0 specifies that only the SIMPLE motion mode will be used.

**use\_ref\_frame\_mvs** equal to 1 specifies that motion vector information from a previous frame can be used when decoding the current frame. **use\_ref\_frame\_mvs** equal to 0 specifies that this information will not be used.

**disable\_frame\_end\_update\_cdf** equal to 1 indicates that the end of frame CDF update is disabled; **disable\_frame\_end\_update\_cdf** equal to 0 indicates that the end of frame CDF update is enabled.

**Note:** It can be useful to disable the CDF update because it means the next frame can start to be decoded as soon as the frame headers of the current frame have been processed.

**motion\_field\_estimation** is a function call which indicates that the motion field estimation process in [section 7.9](#) should be invoked.

**OrderHints** specifies the expected output order for each reference frame.

**CodedLossless** is a variable that is equal to 1 when all segments use lossless encoding. This indicates that the frame is fully lossless at the coded resolution of FrameWidth by FrameHeight. In this case, the loop filter and CDEF filter are disabled.

It is a requirement of bitstream conformance that **delta\_q\_present** is equal to 0 when **CodedLossless** is equal to 1.

**AllLossless** is a variable that is equal to 1 when **CodedLossless** is equal to 1 and FrameWidth is equal to UpscaledWidth. This indicates that the frame is fully lossless at the upscaled resolution. In this case, the loop filter, CDEF filter, and loop restoration are disabled.

**allow\_warped\_motion** equal to 1 indicates that the syntax element **motion\_mode** may be present. **allow\_warped\_motion** equal to 0 indicates that the syntax element **motion\_mode** will not be present (this means that LOCALWARP cannot be signaled if **allow\_warped\_motion** is equal to 0).

**reduced\_tx\_set** equal to 1 specifies that the frame is restricted to a reduced subset of the full set of transform types.

**setup\_past\_independence** is a function call that indicates that this frame can be decoded without dependence on previous coded frames. When this function is invoked the following takes place:

- **FeatureData**[ i ][ j ] and **FeatureEnabled**[ i ][ j ] are set equal to 0 for i = 0..MAX\_SEGMENTS-1 and j = 0..SEG\_LVL\_MAX-1.
- **PrevSegmentIds**[ row ][ col ] is set equal to 0 for row = 0..MiRows-1 and col = 0..MiCols-1.
- **GmType**[ ref ] is set equal to IDENTITY for ref = LAST\_FRAME..ALTREF\_FRAME.
- **PrevGmParams**[ ref ][ i ] is set equal to ( ( i % 3 == 2 ) ? 1 << WARPEDMODEL\_PREC\_BITS : 0 ) for ref = LAST\_FRAME..ALTREF\_FRAME, for i = 0..5.
- **loop\_filter\_delta\_enabled** is set equal to 1.

- `loop_filter_ref_deltas[ INTRA_FRAME ]` is set equal to 1.
- `loop_filter_ref_deltas[ LAST_FRAME ]` is set equal to 0.
- `loop_filter_ref_deltas[ LAST2_FRAME ]` is set equal to 0.
- `loop_filter_ref_deltas[ LAST3_FRAME ]` is set equal to 0.
- `loop_filter_ref_deltas[ BWDREF_FRAME ]` is set equal to 0.
- `loop_filter_ref_deltas[ GOLDEN_FRAME ]` is set equal to -1.
- `loop_filter_ref_deltas[ ALTREF_FRAME ]` is set equal to -1.
- `loop_filter_ref_deltas[ ALTREF2_FRAME ]` is set equal to -1.
- `loop_filter_mode_deltas[ i ]` is set equal to 0 for  $i = 0..1$ .

**`init_non_coeff_cdfs`** is a function call that indicates that the CDF tables which are not used in the `coeff( )` syntax structure should be initialised. When this function is invoked, the following steps apply:

- `YModeCdf` is set to a copy of `Default_Y_Mode_Cdf`
- `UVModeCflNotAllowedCdf` is set to a copy of `Default_Uv_Mode_Cfl_Not_Allowed_Cdf`
- `UVModeCflAllowedCdf` is set to a copy of `Default_Uv_Mode_Cfl_Allowed_Cdf`
- `AngleDeltaCdf` is set to a copy of `Default_Angle_Delta_Cdf`
- `IntrabcCdf` is set to a copy of `Default_Intrabc_Cdf`
- `PartitionW8Cdf` is set to a copy of `Default_Partition_W8_Cdf`
- `PartitionW16Cdf` is set to a copy of `Default_Partition_W16_Cdf`
- `PartitionW32Cdf` is set to a copy of `Default_Partition_W32_Cdf`
- `PartitionW64Cdf` is set to a copy of `Default_Partition_W64_Cdf`
- `PartitionW128Cdf` is set to a copy of `Default_Partition_W128_Cdf`
- `SegmentIdCdf` is set to a copy of `Default_Segment_Id_Cdf`
- `SegmentIdPredictedCdf` is set to a copy of `Default_Segment_Id_Predicted_Cdf`
- `Tx8x8Cdf` is set to a copy of `Default_Tx_8x8_Cdf`
- `Tx16x16Cdf` is set to a copy of `Default_Tx_16x16_Cdf`
- `Tx32x32Cdf` is set to a copy of `Default_Tx_32x32_Cdf`

- Tx64x64Cdf is set to a copy of Default\_Tx\_64x64\_Cdf
- TxmSplitCdf is set to a copy of Default\_Txfrm\_Split\_Cdf
- FilterIntraModeCdf is set to a copy of Default\_Filter\_Intra\_Mode\_Cdf
- FilterIntraCdf is set to a copy of Default\_Filter\_Intra\_Cdf
- InterpFilterCdf is set to a copy of Default\_Interp\_Filter\_Cdf
- MotionModeCdf is set to a copy of Default\_Motion\_Mode\_Cdf
- NewMvCdf is set to a copy of Default\_New\_Mv\_Cdf
- ZeroMvCdf is set to a copy of Default\_Zero\_Mv\_Cdf
- RefMvCdf is set to a copy of Default\_Ref\_Mv\_Cdf
- CompoundModeCdf is set to a copy of Default\_Compound\_Mode\_Cdf
- DrlModeCdf is set to a copy of Default\_Drl\_Mode\_Cdf
- IsInterCdf is set to a copy of Default\_Is\_Inter\_Cdf
- CompModeCdf is set to a copy of Default\_Comp\_Mode\_Cdf
- SkipModeCdf is set to a copy of Default\_Skip\_Mode\_Cdf
- SkipCdf is set to a copy of Default\_Skip\_Cdf
- CompRefCdf is set to a copy of Default\_Comp\_Ref\_Cdf
- CompBwdRefCdf is set to a copy of Default\_Comp\_Bwd\_Ref\_Cdf
- SingleRefCdf is set to a copy of Default\_Single\_Ref\_Cdf
- MvJointCdf[ i ] is set to a copy of Default\_Mv\_Joint\_Cdf for  $i = 0..MV\_CONTEXTS-1$
- MvClassCdf[ i ] is set to a copy of Default\_Mv\_Class\_Cdf for  $i = 0..MV\_CONTEXTS-1$
- MvClass0BitCdf[ i ][ comp ] is set to a copy of Default\_Mv\_Class0\_Bit\_Cdf for  $i = 0..MV\_CONTEXTS-1$  and  $comp = 0..1$
- MvFrCdf[ i ] is set to a copy of Default\_Mv\_Fr\_Cdf for  $i = 0..MV\_CONTEXTS-1$
- MvClass0FrCdf[ i ] is set to a copy of Default\_Mv\_Class0\_Fr\_Cdf for  $i = 0..MV\_CONTEXTS-1$
- MvClass0HpCdf[ i ][ comp ] is set to a copy of Default\_Mv\_Class0\_Hp\_Cdf for  $i = 0..MV\_CONTEXTS-1$  and  $comp = 0..1$

- $MvSignCdf[i][comp]$  is set to a copy of `Default_Mv_Sign_Cdf` for  $i = 0..MV\_CONTEXTS-1$  and  $comp = 0..1$
- $MvBitCdf[i][comp]$  is set to a copy of `Default_Mv_Bit_Cdf` for  $i = 0..MV\_CONTEXTS-1$  and  $comp = 0..1$
- $MvHpCdf[i][comp]$  is set to a copy of `Default_Mv_Hp_Cdf` for  $i = 0..MV\_CONTEXTS-1$  and  $comp = 0..1$
- `PaletteYModeCdf` is set to a copy of `Default_Palette_Y_Mode_Cdf`
- `PaletteUVModeCdf` is set to a copy of `Default_Palette_Uv_Mode_Cdf`
- `PaletteYSizeCdf` is set to a copy of `Default_Palette_Y_Size_Cdf`
- `PaletteUVSizeCdf` is set to a copy of `Default_Palette_Uv_Size_Cdf`
- `PaletteSize2YColorCdf` is set to a copy of `Default_Palette_Size_2_Y_Color_Cdf`
- `PaletteSize2UVColorCdf` is set to a copy of `Default_Palette_Size_2_Uv_Color_Cdf`
- `PaletteSize3YColorCdf` is set to a copy of `Default_Palette_Size_3_Y_Color_Cdf`
- `PaletteSize3UVColorCdf` is set to a copy of `Default_Palette_Size_3_Uv_Color_Cdf`
- `PaletteSize4YColorCdf` is set to a copy of `Default_Palette_Size_4_Y_Color_Cdf`
- `PaletteSize4UVColorCdf` is set to a copy of `Default_Palette_Size_4_Uv_Color_Cdf`
- `PaletteSize5YColorCdf` is set to a copy of `Default_Palette_Size_5_Y_Color_Cdf`
- `PaletteSize5UVColorCdf` is set to a copy of `Default_Palette_Size_5_Uv_Color_Cdf`
- `PaletteSize6YColorCdf` is set to a copy of `Default_Palette_Size_6_Y_Color_Cdf`
- `PaletteSize6UVColorCdf` is set to a copy of `Default_Palette_Size_6_Uv_Color_Cdf`
- `PaletteSize7YColorCdf` is set to a copy of `Default_Palette_Size_7_Y_Color_Cdf`
- `PaletteSize7UVColorCdf` is set to a copy of `Default_Palette_Size_7_Uv_Color_Cdf`
- `PaletteSize8YColorCdf` is set to a copy of `Default_Palette_Size_8_Y_Color_Cdf`
- `PaletteSize8UVColorCdf` is set to a copy of `Default_Palette_Size_8_Uv_Color_Cdf`
- `DeltaQCdf` is set to a copy of `Default_Delta_Q_Cdf`
- `DeltaLFCdf` is set to a copy of `Default_Delta_Lf_Cdf`
- $\Delta LFMultiCdf[i]$  is set to a copy of `Default_Delta_Lf_Cdf` for  $i = 0..FRAME\_LF\_COUNT-1$
- `IntraTxTypeSet1Cdf` is set to a copy of `Default_Intra_Tx_Type_Set1_Cdf`

- IntraTxTypeSet2Cdf is set to a copy of Default\_Intra\_Tx\_Type\_Set2\_Cdf
- InterTxTypeSet1Cdf is set to a copy of Default\_Inter\_Tx\_Type\_Set1\_Cdf
- InterTxTypeSet2Cdf is set to a copy of Default\_Inter\_Tx\_Type\_Set2\_Cdf
- InterTxTypeSet3Cdf is set to a copy of Default\_Inter\_Tx\_Type\_Set3\_Cdf
- UseObmcCdf is set to a copy of Default\_Use\_Obmc\_Cdf
- InterIntraCdf is set to a copy of Default\_Inter\_Intra\_Cdf
- CompRefTypeCdf is set to a copy of Default\_Comp\_Ref\_Type\_Cdf
- CflSignCdf is set to a copy of Default\_Cfl\_Sign\_Cdf
- UniCompRefCdf is set to a copy of Default\_Uni\_Comp\_Ref\_Cdf
- WedgeInterIntraCdf is set to a copy of Default\_Wedge\_Inter\_Intra\_Cdf
- CompGroupIdxCdf is set to a copy of Default\_Comp\_Group\_Idx\_Cdf
- CompoundIdxCdf is set to a copy of Default\_Compound\_Idx\_Cdf
- CompoundTypeCdf is set to a copy of Default\_Compound\_Type\_Cdf
- InterIntraModeCdf is set to a copy of Default\_Inter\_Intra\_Mode\_Cdf
- WedgeIndexCdf is set to a copy of Default\_Wedge\_Index\_Cdf
- CflAlphaCdf is set to a copy of Default\_Cfl\_Alpha\_Cdf
- UseWienerCdf is set to a copy of Default\_Use\_Wiener\_Cdf
- UseSgrprojCdf is set to a copy of Default\_Use\_Sgrproj\_Cdf
- RestorationTypeCdf is set to a copy of Default\_Restoration\_Type\_Cdf

**init\_coeff\_cdfs( )** is a function call that indicates that the CDF tables used in the `coeff( )` syntax structure should be initialised. When this function is invoked, the following steps apply:

- The variable `idx` is derived as follows:
  - If `base_q_idx` is less than or equal to 20, `idx` is set equal to 0.
  - Otherwise, if `base_q_idx` is less than or equal to 60, `idx` is set equal to 1.
  - Otherwise, if `base_q_idx` is less than or equal to 120, `idx` is set equal to 2.
  - Otherwise, `idx` is set equal to 3.

- The cumulative distribution function arrays are reset to default values as follows:

- TxbSkipCdf is set to a copy of Default\_Txb\_Skip\_Cdf[ idx ].
- EobPt16Cdf is set to a copy of Default\_Eob\_Pt\_16\_Cdf[ idx ].
- EobPt32Cdf is set to a copy of Default\_Eob\_Pt\_32\_Cdf[ idx ].
- EobPt64Cdf is set to a copy of Default\_Eob\_Pt\_64\_Cdf[ idx ].
- EobPt128Cdf is set to a copy of Default\_Eob\_Pt\_128\_Cdf[ idx ].
- EobPt256Cdf is set to a copy of Default\_Eob\_Pt\_256\_Cdf[ idx ].
- EobPt512Cdf is set to a copy of Default\_Eob\_Pt\_512\_Cdf[ idx ].
- EobPt1024Cdf is set to a copy of Default\_Eob\_Pt\_1024\_Cdf[ idx ].
- EobExtraCdf is set to a copy of Default\_Eob\_Extra\_Cdf[ idx ].
- DcSignCdf is set to a copy of Default\_Dc\_Sign\_Cdf[ idx ].
- CoeffBaseEobCdf is set to a copy of Default\_Coeff\_Base\_Eob\_Cdf[ idx ].
- CoeffBaseCdf is set to a copy of Default\_Coeff\_Base\_Cdf[ idx ].
- CoeffBrCdf is set to a copy of Default\_Coeff\_Br\_Cdf[ idx ].

**load\_cdfs( ctx )** is a function call that indicates that the CDF tables are loaded from frame context number ctx in the range 0 to (NUM\_REF\_FRAMES - 1). When this function is invoked, a copy of each CDF array mentioned in the semantics for init\_coeff\_cdfs and init\_non\_coeff\_cdfs is loaded from an area of memory indexed by ctx. (The memory contents of these frame contexts have been initialized by previous calls to save\_cdfs). Once the CDF arrays have been loaded, the last entry in each array, representing the symbol count for that context, is set to 0.

**load\_previous( )** is a function call that indicates that information from a previous frame may be loaded for use in decoding the current frame. When this function is invoked the following ordered steps apply:

1. The variable prevFrame is set equal to ref\_frame\_idx[ primary\_ref\_frame ].
2. PrevGmParams is set equal to SavedGmParams[ prevFrame ].
3. The function load\_loop\_filter\_params( prevFrame ) specified in [section 7.21](#) is invoked.
4. The function load\_segmentation\_params( prevFrame ) specified in [section 7.21](#) is invoked.

**load\_previous\_segment\_ids( )** is a function call that indicates that a segment map from a previous frame may be loaded for use in decoding the current frame. When this function is invoked the segment map contained in PrevSegmentIds is set as follows:



1. The variable `prevFrame` is set equal to `ref_frame_idx[ primary_ref_frame ]`.
2. If `segmentation_enabled` is equal to 1, `RefMiCols[ prevFrame ]` is equal to `MiCols`, and `RefMiRows[ prevFrame ]` is equal to `MiRows`, `PrevSegmentIds[ row ][ col ]` is set equal to `SavedSegmentIds[ prevFrame ][ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.

Otherwise, `PrevSegmentIds[ row ][ col ]` is set equal to 0 for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.

### 6.8.3. Reference frame marking semantics

**RefValid** is an array which is indexed by a reference picture slot number. A value of 1 in the array signifies that the corresponding reference picture slot is valid for use as a reference picture, while a value of 0 signifies that the corresponding reference picture slot is not valid for use as a reference picture.

**Note:** `RefValid` is only used to define valid bitstreams when `frame_id_numbers_present_flag` is equal to 1. Frames are marked as invalid when they are too far in the past to be referenced by the frame id mechanism.

### 6.8.4. Frame size semantics

`frame_width_minus_1` plus one is the width of the frame in luma samples.

`frame_height_minus_1` plus one is the height of the frame in luma samples.

It is a requirement of bitstream conformance that `frame_width_minus_1` is less than or equal to `max_frame_width_minus_1`.

It is a requirement of bitstream conformance that `frame_height_minus_1` is less than or equal to `max_frame_height_minus_1`.

If `FramelIntra` is equal to 0 (indicating that this frame may use inter prediction), the requirements described in the frame size with refs semantics of [section 6.7.5](#) must also be satisfied.

### 6.8.5. Render size semantics

The render size is provided as a hint to the application about the desired display size. It has no effect on the decoding process.

`render_and_frame_size_different` equal to 0 means that the render width and height are inferred from the frame width and height. `render_and_frame_size_different` equal to 1 means that the render width and height are explicitly coded.

**Note:** It is allowed for the bitstream to explicitly code the render dimensions in the bitstream even if they are an exact match for the frame dimensions.

`render_width_minus_1` plus one is the render width of the frame in luma samples.

`render_height_minus_1` plus one is the render height of the frame in luma samples.

## 6.8.6. Frame size with refs semantics

For inter frames, the frame size is either set equal to the size of a reference frame, or can be sent explicitly.

**found\_ref** equal to 1 indicates that the frame dimensions can be inferred from reference frame  $i$  where  $i$  is the loop counter in the syntax parsing process for `frame_size_with_refs`. **found\_ref** equal to 0 indicates that the frame dimensions are not inferred from reference frame  $i$ .

Once the `FrameWidth` and `FrameHeight` have been computed for an inter frame, it is a requirement of bitstream conformance that for all values of  $i$  in the range  $0..(\text{REFS\_PER\_FRAME} - 1)$ , all the following conditions are true:

- $2 * \text{FrameWidth} \geq \text{RefUpscaledWidth}[\text{ref\_frame\_idx}[i]]$
- $2 * \text{FrameHeight} \geq \text{RefFrameHeight}[\text{ref\_frame\_idx}[i]]$
- $\text{FrameWidth} \leq 16 * \text{RefUpscaledWidth}[\text{ref\_frame\_idx}[i]]$
- $\text{FrameHeight} \leq 16 * \text{RefFrameHeight}[\text{ref\_frame\_idx}[i]]$

**Note:** This is a requirement even if all the blocks in an inter frame are coded using intra prediction.

## 6.8.7. Superres params semantics

**use\_superres** equal to 0 indicates that no upscaling is needed. **use\_superres** equal to 1 indicates that upscaling is needed.

**coded\_denom** is used to compute the amount of upscaling.

**SuperresDenom** is the denominator of a fraction that specifies the ratio between the superblock width before and after upscaling. The numerator of this fraction is equal to the constant `SUPERRES_NUM`.

## 6.8.8. Compute image size semantics

**MiCols** is the number of 4x4 block columns in the frame.

**MiRows** is the number of 4x4 block rows in the frame.

## 6.8.9. Interpolation filter semantics

**is\_filter\_switchable** equal to 1 indicates that the filter selection is signaled at the block level; **is\_filter\_switchable** equal to 0 indicates that the filter selection is signaled at the frame level.

**interpolation\_filter** specifies the filter selection used for performing inter prediction:

<b>interpolation_filter</b>	<b>Name of interpolation_filter</b>
0	EIGHTTAP
1	EIGHTTAP_SMOOTH
2	EIGHTTAP_SHARP

interpolation_filter	Name of interpolation_filter
3	BILINEAR
4	SWITCHABLE

## 6.8.10. Loop filter semantics

**loop\_filter\_level** is an array containing loop filter strength values. Different loop filter strength values from the array are used depending on the image plane being filtered, and the edge direction (vertical or horizontal) being filtered.

**loop\_filter\_sharpness** indicates the sharpness level. The **loop\_filter\_level** and **loop\_filter\_sharpness** together determine when a block edge is filtered, and by how much the filtering can change the sample values.

The loop filter process is described in [section 7.14](#).

**loop\_filter\_delta\_enabled** equal to 1 means that the filter level depends on the mode and reference frame used to predict a block. **loop\_filter\_delta\_enabled** equal to 0 means that the filter level does not depend on the mode and reference frame.

**loop\_filter\_delta\_update** equal to 1 means that additional syntax elements are present that specify which mode and reference frame deltas are to be updated. **loop\_filter\_delta\_update** equal to 0 means that these syntax elements are not present.

**update\_ref\_delta** equal to 1 means that the syntax element **loop\_filter\_ref\_delta** is present; **update\_ref\_delta** equal to 0 means that this syntax element is not present.

**loop\_filter\_ref\_deltas** contains the adjustment needed for the filter level based on the chosen reference frame. If this syntax element is not present, it maintains its previous value.

**update\_mode\_delta** equal to 1 means that the syntax element **loop\_filter\_mode\_deltas** is present; **update\_mode\_delta** equal to 0 means that this syntax element is not present.

**loop\_filter\_mode\_deltas** contains the adjustment needed for the filter level based on the chosen mode. If this syntax element is not present in the, it maintains its previous value.

**Note:** The previous values for **loop\_filter\_mode\_deltas** and **loop\_filter\_ref\_deltas** are initially set by the **setup\_past\_independence** function and can be subsequently modified by these syntax elements being coded in a previous frame.

## 6.8.11. Quantization params semantics

The residual is specified via decoded coefficients which are adjusted by one of four quantization parameters before the inverse transform is applied. The choice depends on the plane (Y or UV) and coefficient position (DC/AC coefficient). The Dequantization process is specified in [section 7.12](#).

**base\_q\_idx** indicates the base frame qindex. This is used for Y AC coefficients and as the base value for the other quantizers.

**DeltaQYDc** indicates the Y DC quantizer relative to base\_q\_idx.

**diff\_uv\_delta** equal to 1 indicates that the U and V delta quantizer values are coded separately. **diff\_uv\_delta** equal to 0 indicates that the U and V delta quantizer values share a common value.

**DeltaQUdC** indicates the U DC quantizer relative to base\_q\_idx.

**DeltaQUAc** indicates the U AC quantizer relative to base\_q\_idx.

**DeltaQVDc** indicates the V DC quantizer relative to base\_q\_idx.

**DeltaQVAc** indicates the V AC quantizer relative to base\_q\_idx.

**using\_qmatrix** specifies that the quantizer matrix will be used to compute quantizers.

**qm\_y** specifies the level in the quantizer matrix that should be used for luma plane decoding.

**qm\_u** specifies the level in the quantizer matrix that should be used for chroma U plane decoding.

**qm\_v** specifies the level in the quantizer matrix that should be used for chroma V plane decoding.

## 6.8.12. Delta quantizer semantics

**delta\_coded** specifies that the delta\_q syntax element is present.

**delta\_q** specifies an offset (relative to base\_q\_idx) for a particular quantization parameter.

## 6.8.13. Segmentation params semantics

AV1 provides a means of segmenting the image and then applying various adjustments at the segment level.

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

1. A quantizer (absolute value or delta).
2. A loop filter strength (absolute value or delta).
3. A prediction reference frame.
4. 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 as described in the semantics for setup\_past\_independence.

The segment affiliation (the segmentation map) is stored at the resolution of 4x4 blocks. If no explicit update is coded for a block's segment affiliation, then it persists from frame to frame (until reset by a call to setup\_past\_independence).

**SegIdPreSkip** equal to 1 indicates that the segment id will be read before the skip syntax element. **SegIdPreSkip** equal to 0 indicates that the skip syntax element will be read first.

**LastActiveSegId** indicates the highest numbered segment id that has some enabled feature. This is used when decoding the segment id to only decode choices corresponding to used segments.

**segmentation\_enabled** equal to 1 indicates that this frame makes use of the segmentation tool; **segmentation\_enabled** equal to 0 indicates that the frame does not use segmentation.

**segmentation\_update\_map** equal to 1 indicates that the segmentation map are updated during the decoding of this frame. **segmentation\_update\_map** equal to 0 means that the segmentation map from the previous frame is used.

**segmentation\_temporal\_update** equal to 1 indicates that the updates to the segmentation map are coded relative to the existing segmentation map. **segmentation\_temporal\_update** equal to 0 indicates that the new segmentation map is coded without reference to the existing segmentation map.

**segmentation\_update\_data** equal to 1 indicates that new parameters are about to be specified for each segment. **segmentation\_update\_data** equal to 0 indicates that the segmentation parameters should keep their existing values.

**feature\_enabled** equal to 0 indicates that the corresponding feature is unused and has value equal to 0. **feature\_enabled** equal to 1 indicates that the feature value is coded.

**feature\_value** specifies the feature data for a segment feature.

## 6.8.14. Tile info semantics

**uniform\_tile\_spacing\_flag** equal to 1 means that the tiles are uniformly spaced across the frame. (In other words, all tiles are the same size except for the ones at the right and bottom edge which can be smaller.) **uniform\_tile\_spacing\_flag** equal to 0 means that the tile sizes are coded.

**increment\_tile\_cols\_log2** is used to compute **TileColsLog2**.

**TileColsLog2** specifies the base 2 logarithm of the desired number of tiles across the frame.

**TileCols** specifies the number of tiles across the frame. It is a requirement of bitstream conformance that **TileCols** is less than or equal to **MAX\_TILE\_COLS**.

**increment\_tile\_rows\_log2** is used to compute **TileRowsLog2**.

**TileRowsLog2** specifies the base 2 logarithm of the desired number of tiles down the frame.

**Note:** For small frame sizes the actual number of tiles in the frame may be smaller than the desired number because the tile size is rounded up to a multiple of the maximum superblock size.

**TileRows** specifies the number of tiles down the frame. It is a requirement of bitstream conformance that **TileRows** is less than or equal to **MAX\_TILE\_ROWS**.

**tileWidthSb** is used to specify the width of each tile in units of superblocks. It is a requirement of bitstream conformance that `tileWidthSb` is less than `maxTileWidthSb`.

**tileHeightSb** is used to specify the height of each tile in units of superblocks. It is a requirement of bitstream conformance that `tileWidthSb * tileHeightSb` is less than `maxTileAreaSb`.

If `uniform_tile_spacing_flag` is equal to 0, it is a requirement of bitstream conformance that `startSb` is equal to `sbCols` when the loop writing `MiColStarts` exits.

If `uniform_tile_spacing_flag` is equal to 0, it is a requirement of bitstream conformance that `startSb` is equal to `sbRows` when the loop writing `MiRowStarts` exits.

**Note:** The requirements on `startSb` ensure that the sizes of each tile add up to the full size of the frame when measured in superblocks.

**MiColStarts** is an array specifying the start column (in units of 4x4 luma samples) for each tile across the image.

**MiRowStarts** is an array specifying the start row (in units of 4x4 luma samples) for each tile down the image.

**width\_in\_sbs\_minus\_1** specifies the width of a tile minus 1 in units of superblocks.

**height\_in\_sbs\_minus\_1** specifies the height of a tile minus 1 in units of superblocks.

**maxTileHeightSb** specifies the maximum height (in units of superblocks) that can be used for a tile (to avoid making tiles with too much area).

**context\_update\_tile\_id** specifies which tile to use for the CDF update.

**tile\_size\_bytes\_minus\_1** is used to compute `TileSizeBytes`.

**TileSizeBytes** specifies the number of bytes needed to code each tile size.

## 6.8.15. Quantizer index delta parameters semantics

**delta\_q\_present** specifies whether quantizer index delta values are present.

**delta\_q\_res** specifies the left shift which should be applied to decoded quantizer index delta values.

## 6.8.16. Loop filter delta parameters semantics

**delta\_lf\_present** specifies whether loop filter delta values are present.

**delta\_lf\_res** specifies the left shift which should be applied to decoded loop filter delta values.

**delta\_lf\_multi** equal to 1 specifies that separate loop filter deltas are sent for horizontal luma edges, vertical luma edges, the U edges, and the V edges. **delta\_lf\_multi** equal to 0 specifies that the same loop filter delta is used for all edges.

## 6.8.17. Global motion params semantics

**is\_global** specifies whether global motion parameters are present for a particular reference frame.

**is\_rot\_zoom** specifies whether a particular reference frame uses rotation and zoom global motion.

**is\_translation** specifies whether a particular reference frame uses translation global motion.

## 6.8.18. Global param semantics

**absBits** is used to compute the range of values that can be used for `gm_params[ref][idx]`. The values allowed are in the range  $-(1 \ll \text{absBits})$  to  $(1 \ll \text{absBits})$ .

**precBits** specifies the number of fractional bits used for representing `gm_params[ref][idx]`. All global motion parameters are stored in the model with `WARPEDMODEL_PREC_BITS` fractional bits, but the parameters are encoded with less precision.

## 6.8.19. Decode subexp semantics

**subexp\_final\_bits** provide the final bits that are read once the appropriate range has been determined.

**subexp\_more\_bits** equal to 0 specifies that the parameter is in the range  $m_k$  to  $m_k+a-1$ . **subexp\_more\_bits** equal to 1 specifies that the parameter is greater than  $m_k+a-1$ .

**subexp\_bits** specifies the value of the parameter minus  $m_k$ .

## 6.8.20. Film grain params semantics

**apply\_grain** equal to 1 specifies that film grain should be added to this frame. **apply\_grain** equal to 0 specifies that film grain should not be added.

**reset\_grain\_params()** is a function call that indicates that all the syntax elements read in `film_grain_params` should be set equal to 0.

**grain\_seed** specifies the starting value for the pseudo-random numbers used during film grain synthesis.

**update\_grain** equal to 1 means that a new set of parameters should be sent. **update\_grain** equal to 0 means that the previous set of parameters should be used.

**film\_grain\_params\_ref\_idx** indicates which reference frame contains the film grain parameters to be used for this frame.

It is a requirement of bitstream conformance that `film_grain_params_ref_idx` is equal to `ref_frame_idx[j]` for some value of `j` in the range 0 to `REFS_PER_FRAME - 1`.

**Note:** This requirement means that film grain can only be predicted from the frames that the current frame is using as reference frames.

**load\_grain\_params(idx)** is a function call that indicates that all the syntax elements read in `film_grain_params` should be set equal to the values stored in an area of memory indexed by `idx`.

**tempGrainSeed** is a temporary variable that is used to avoid losing the value of `grain_seed` when `load_grain_params` is called. When `update_grain` is equal to 0, a previous set of parameters should be used for everything except `grain_seed`.

**num\_y\_points** specifies the number of points for the piece-wise linear scaling function of the luma component.

It is a requirement of bitstream conformance that `num_y_points` is less than or equal to 14.

**point\_y\_value[ i ]** represents the x (luma value) coordinate for the i-th point of the piecewise linear scaling function for luma component. The values are signaled on the scale of 0..255. (In case of 10 bit video, these values correspond to luma values divided by 4. In case of 12 bit video, these values correspond to luma values divided by 16.)

If `i` is greater than 0, it is a requirement of bitstream conformance that `point_y_value[ i ]` is greater than `point_y_value[ i - 1 ]` (this ensures the x coordinates are specified in increasing order).

**point\_y\_scaling[ i ]** represents the scaling (output) value for the i-th point of the piecewise linear scaling function for luma component.

**chroma\_scaling\_from\_luma** specifies that the chroma scaling is inferred from the luma scaling.

**num\_cb\_points** specifies the number of points for the piece-wise linear scaling function of the cb component.

It is a requirement of bitstream conformance that `num_cb_points` is less than or equal to 10.

**Note:** When `chroma_scaling_from_luma` is equal to 1, it is still allowed for `num_y_points` to take values up to 14. This means that the chroma scaling also needs to support up to 14 points.

**point\_cb\_value[ i ]** represents the x coordinate for the i-th point of the piece-wise linear scaling function for cb component. The values are signaled on the scale of 0..255.

If `i` is greater than 0, it is a requirement of bitstream conformance that `point_cb_value[ i ]` is greater than `point_cb_value[ i - 1 ]`.

**point\_cb\_scaling[ i ]** represents the scaling (output) value for the i-th point of the piecewise linear scaling function for cb component.

**num\_cr\_points** specifies represents the number of points for the piece-wise linear scaling function of the cr component.

It is a requirement of bitstream conformance that `num_cr_points` is less than or equal to 10.

If `subsampling_x` is equal to 1 and `subsampling_y` is equal to 1 and `num_cb_points` is equal to 0, it is a requirement of bitstream conformance that `num_cr_points` is equal to 0.

If `subsampling_x` is equal to 1 and `subsampling_y` is equal to 1 and `num_cb_points` is not equal to 0, it is a requirement of bitstream conformance that `num_cr_points` is not equal to 0.



**Note:** These requirements ensure that for 4:2:0 chroma subsampling, film grain noise will be applied to both chroma components, or to neither. There is no restriction for 4:2:2 or 4:4:4 chroma subsampling.

**point\_cr\_value[ i ]** represents the x coordinate for the i-th point of the piece-wise linear scaling function for cr component. The values are signaled on the scale of 0..255.

If i is greater than 0, it is a requirement of bitstream conformance that point\_cr\_value[ i ] is greater than point\_cr\_value[ i - 1 ].

**point\_cr\_scaling[ i ]** represents the scaling (output) value for the i-th point of the piecewise linear scaling function for cr component.

**grain\_scaling\_minus\_8** represents the shift – 8 applied to the values of the chroma component. The grain\_scaling\_minus\_8 can take values of 0..3 and determines the range and quantization step of the standard deviation of film grain.

**ar\_coeff\_lag** specifies the number of auto-regressive coefficients for luma and chroma.

**ar\_coeffs\_y\_plus\_128[ i ]** specifies auto-regressive coefficients used for the Y plane.

**ar\_coeffs\_cb\_plus\_128[ i ]** specifies auto-regressive coefficients used for the U plane.

**ar\_coeffs\_cr\_plus\_128[ i ]** specifies auto-regressive coefficients used for the V plane.

**ar\_coeff\_shift\_minus\_6** specifies the range of the auto-regressive coefficients. Values of 0, 1, 2, and 3 correspond to the ranges for auto-regressive coefficients of [-2, 2), [-1, 1), [-0.5, 0.5) and [-0.25, 0.25) respectively.

**grain\_scale\_shift** specifies how much the Gaussian random numbers should be scaled down during the grain synthesis process.

**cb\_mult** represents a multiplier for the cb component used in derivation of the input index to the cb component scaling function.

**cb\_luma\_mult** represents a multiplier for the average luma component used in derivation of the input index to the cb component scaling function.

**cb\_offset** represents an offset used in derivation of the input index to the cb component scaling function.

**cr\_mult** represents a multiplier for the cr component used in derivation of the input index to the cr component scaling function.

**cr\_luma\_mult** represents a multiplier for the average luma component used in derivation of the input index to the cr component scaling function.

**cr\_offset** represents an offset used in derivation of the input index to the cr component scaling function.

**overlap\_flag** equal to 1 indicates that the overlap between film grain blocks shall be applied. **overlap\_flag** equal to 0 indicates that the overlap between film grain blocks shall not be applied.

**clip\_to\_restricted\_range** equal to 1 indicates that clipping to the restricted (studio) range shall be applied to the sample values after adding the film grain (see the semantics for **color\_range** for an explanation of studio swing).

**clip\_to\_restricted\_range** equal to 0 indicates that clipping to the full range shall be applied to the sample values after adding the film grain.

## 6.8.21. TX mode semantics

**tx\_mode\_select** is used to compute TxMode.

**TxMode** specifies how the transform size is determined:

TxMode	Name of TxMode
0	ONLY_4X4
1	TX_MODE_LARGEST
2	TX_MODE_SELECT

For **tx\_mode** equal to TX\_MODE\_LARGEST, the inverse transform will use the largest transform size that fits inside the block.

For **tx\_mode** equal to ONLY\_4X4, the inverse transform will use only 4x4 transforms.

For **tx\_mode** equal to TX\_MODE\_SELECT, the choice of transform size is specified explicitly for each block.

## 6.8.22. Skip mode params semantics

**SkipModeFrame[ list ]** specifies the frames to use for compound prediction when **skip\_mode** is equal to 1.

**skip\_mode\_present** equal to 1 specifies that the syntax element **skip\_mode** will be present. **skip\_mode\_present** equal to 0 specifies that **skip\_mode** will not be used for this frame.

**Note:** Skip mode tries to use the closest forward and backward references (as measured by values in the RefOrderHint array). If no backward reference is found, then the second closest forward reference is used. If no forward reference is found, then skip mode is disabled. (Forward prediction is when a frame is used for reference that is considered to be output before the current frame, backward prediction is when a frame is used that has not yet been output.)

## 6.8.23. Frame reference mode semantics

**reference\_select** equal to 1 specifies that the mode info for inter blocks contains the syntax element **comp\_mode** that indicates whether to use single or compound reference prediction. **reference\_select** equal to 0 specifies that all inter blocks will use single prediction.

## 6.8.24. Temporal point info semantics

**frame\_presentation\_time** specifies the presentation time of the frame in clock ticks DispCT counted from the removal time of the last random access point for the operating point that is being decoded. The syntax element is signaled as a fixed length unsigned integer with a length in bits given by `frame_presentation_time_length_minus_1 + 1`. The `frame_presentation_time` is the remainder of a modulo  $1 \ll (\text{frame\_presentation\_time\_length\_minus\_1} + 1)$  counter.

## 6.9. Frame OBU semantics

A frame OBU consists of a frame header OBU and a tile group OBU packed into a single OBU.

**Note:** The intention is to provide a more compact way of coding the common use case where the frame header is immediately followed by tile group data.

## 6.10. Tile group OBU semantics

### 6.10.1. General tile group OBU semantics

**NumTiles** specifies the total number of tiles in the frame.

**tile\_start\_and\_end\_present\_flag** specifies whether `tg_start` and `tg_end` are present. If `tg_start` and `tg_end` are not present, this tile group covers the entire frame.

If `obu_type` is equal to `OBU_FRAME`, it is a requirement of bitstream conformance that the value of `tile_start_and_end_present_flag` is equal to 0.

**tg\_start** specifies the zero-based index of the first tile in the current tile group.

It is a requirement of bitstream conformance that the value of `tg_start` is equal to the value of `TileNum` at the point that `tile_group_obu` is invoked.

**tg\_end** specifies the zero-based index of the last tile in the current tile group.

It is a requirement of bitstream conformance that the value of `tg_end` is greater than or equal to `tg_start`.

It is a requirement of bitstream conformance that the value of `tg_end` for the last tile group in each frame is equal to `NumTiles - 1`.

**Note:** These requirements ensure that conceptually all tile groups are present and received in order for the purposes of specifying the decode process.

**frame\_end\_update\_cdf** is a function call that indicates that the frame CDF arrays are set equal to the saved CDFs. This process is described in [section 7.7](#).

**tile\_size\_minus\_1** is used to compute `tileSize`.

**tileSize** specifies the size in bytes of the next coded tile.

**Note:** This size includes any padding bytes if added by the exit process for the Symbol decoder. The size does not include the bytes used for `tile_size_minus_1` or syntax elements sent before `tile_size_minus_1`. For the last tile in the tile group, `tileSize` is computed instead of being read and includes the OBU trailing bits.

**decode\_frame\_wrapup** is a function call that indicates that the decode frame wrapup process specified in [section 7.4](#) should be invoked.

## 6.10.2. Decode tile semantics

**clear\_left\_context** is a function call that indicates that some arrays used to determine the probabilities are zeroed. When this function is invoked the arrays `LeftLevelContext`, `LeftDcContext`, and `LeftSegPredContext` are set equal to 0.

**Note:** `LeftLevelContext[ plane ][ i ]`, `LeftDcContext[ plane ][ i ]`, and `LeftSegPredContext[ i ]` need to be set to 0 for  $i = 0..MiRows-1$ , for  $plane = 0..2$ .

**clear\_above\_context** is a function call that indicates that some arrays used to determine the probabilities are zeroed. When this function is invoked the arrays `AboveLevelContext`, `AboveDcContext`, and `AboveSegPredContext` are set equal to 0.

**Note:** `AboveLevelContext[ plane ][ i ]`, `AboveDcContext[ plane ][ i ]`, and `AboveSegPredContext[ i ]` need to be set to 0 for  $i = 0..MiCols-1$ , for  $plane = 0..2$ .

**ReadDeltas** specifies whether the current block may read delta values for the quantizer index and loop filter. If the entire superblock is skipped the delta values are not read, otherwise delta values for the quantizer index and loop filter are read on the first block of a superblock. If `delta_q_present` is equal to 0, no delta values are read for the quantizer index. If `delta_lf_present` is equal to 0, no delta values are read for the loop filter.

## 6.10.3. Clear block decoded flags semantics

**BlockDecoded** is an array which stores one boolean value per 4x4 sample block per plane in the current superblock, plus a border of one 4x4 sample block on all sides of the superblock. Except for the borders, a value of 1 in `BlockDecoded` indicates that the corresponding 4x4 sample block has been decoded. The borders are used when computing above-right and below-left availability along the top and left edges of the superblock.

## 6.10.4. Decode partition semantics

**partition** specifies how a block is partitioned:

partition	Name of partition
0	PARTITION_NONE

<b>partition</b>	<b>Name of partition</b>
1	PARTITION_HORZ
2	PARTITION_VERT
3	PARTITION_SPLIT
4	PARTITION_HORZ_A
5	PARTITION_HORZ_B
6	PARTITION_VERT_A
7	PARTITION_VERT_B
8	PARTITION_HORZ_4
9	PARTITION_VERT_4

The variable **subSize** is computed from partition and indicates the size of the component blocks within this block:

<b>subSize</b>	<b>Name of subSize</b>
0	BLOCK_4X4
1	BLOCK_4X8
2	BLOCK_8X4
3	BLOCK_8X8
4	BLOCK_8X16
5	BLOCK_16X8
6	BLOCK_16X16
7	BLOCK_16X32
8	BLOCK_32X16
9	BLOCK_32X32
10	BLOCK_32X64
11	BLOCK_64X32
12	BLOCK_64X64
13	BLOCK_64X128
14	BLOCK_128X64
15	BLOCK_128X128
16	BLOCK_4X16

subSize	Name of subSize
17	BLOCK_16X4
18	BLOCK_8X32
19	BLOCK_32X8
20	BLOCK_16X64
21	BLOCK_64X16

The dimensions of these blocks are given in width, height order (e.g. BLOCK\_8X16 corresponds to a block that is 8 samples wide, and 16 samples high).

It is a requirement of bitstream conformance that `get_plane_residual_size( subSize, 1 )` is not equal to BLOCK\_INVALID every time subSize is computed.

**Note:** This requirement prevents the UV blocks from being too tall or too wide (i.e. having aspect ratios outside the range 1:2 to 2:1).

**split\_or\_vert** is used to compute partition for blocks when only split or vert partitions are allowed because of overlap with the right hand edge of the frame.

**split\_or\_horz** is used to compute partition for blocks when only split or horz partitions are allowed because of overlap with the bottom edge of the frame.

## 6.10.5. Decode block semantics

**MiRow** is a variable holding the vertical location of the block in units of 4x4 luma samples.

**MiCol** is a variable holding the horizontal location of the block in units of 4x4 luma samples.

**MiSize** is a variable holding the size of the block with values having the same interpretation for the variable subSize.

**HasChroma** is a variable that specifies whether chroma information is coded for this block.

Variable **AvailU** is equal to 0 if the information from the block above cannot be used on the luma plane; AvailU is equal to 1 if the information from the block above can be used on the luma plane.

Variable **AvailL** is equal to 0 if the information from the block to the left can not be used on the luma plane; AvailL is equal to 1 if the information from the block to the left can be used on the luma plane.

**Note:** Information from a block in a different tile can be used in some circumstances if the block is above, but not if the block is to the left.

Variables **AvailUChroma** and **AvailLChroma** have the same significance as AvailU and AvailL, but on the chroma planes.

## 6.10.6. Intra frame mode info semantics

This syntax is used when coding an intra block within an intra frame.

**use\_intrabc** equal to 1 specifies that intra block copy should be used for this block. **use\_intrabc** equal to 0 specifies that intra block copy should not be used.

**intra\_frame\_y\_mode** specifies the direction of intra prediction filtering:

<b>intra_frame_y_mode</b>	<b>Name of intra_frame_y_mode</b>
0	DC_PRED
1	V_PRED
2	H_PRED
3	D45_PRED
4	D135_PRED
5	D113_PRED
6	D157_PRED
7	D203_PRED
8	D67_PRED
9	SMOOTH_PRED
10	SMOOTH_V_PRED
11	SMOOTH_H_PRED
12	PAETH_PRED

**uv\_mode** specifies the chrominance intra prediction mode using values with the same interpretation as in the semantics for **intra\_frame\_y\_mode**, with an additional mode UV\_CFL\_PRED.

<b>uv_mode</b>	<b>Name of uv_mode</b>
0	DC_PRED
1	V_PRED
2	H_PRED
3	D45_PRED
4	D135_PRED

uv_mode	Name of uv_mode
5	D113_PRED
6	D157_PRED
7	D203_PRED
8	D67_PRED
9	SMOOTH_PRED
10	SMOOTH_V_PRED
11	SMOOTH_H_PRED
12	PAETH_PRED
13	UV_CFL_PRED

**Note:** Due to the way the uv\_mode syntax element is read, uv\_mode can only be read as UV\_CFL\_PRED when  $\text{Max}(\text{Block\_Width}[\text{MiSize}], \text{Block\_Height}[\text{MiSize}]) \leq 32$ .

### 6.10.7. Intra segment ID semantics

**Lossless** is a variable which, if equal to 1, indicates that the block is coded using a special 4x4 transform designed for encoding frames that are bit-identical with the original frames.

### 6.10.8. Read segment ID semantics

**segment\_id** specifies which segment is associated with the current intra block being decoded. It is first read from the stream, and then postprocessed based on the predicted segment id.

It is a requirement of bitstream conformance that the postprocessed value of segment\_id (i.e. the value returned by neg\_deinterleave) is in the range 0 to LastActiveSegId (inclusive of endpoints).

### 6.10.9. Inter segment ID semantics

**seg\_id\_predicted** equal to 1 specifies that the segment\_id is taken from the segmentation map. seg\_id\_predicted equal to 0 specifies that the syntax element segment\_id is parsed.

**Note:** It is allowed for seg\_id\_predicted to be equal to 0 even if the value coded for the segment\_id is equal to predictedSegmentId.

### 6.10.10. Skip mode semantics

**skip\_mode** equal to 1 indicates that this block will use some default settings (that correspond to compound prediction) and so most of the mode info is skipped. skip\_mode equal to 0 indicates that the mode info is not skipped.



## 6.10.11. Skip semantics

**skip** equal to 0 indicates that there may be some transform coefficients to read for this block; skip equal to 1 indicates that there are no transform coefficients.

## 6.10.12. Quantizer index delta semantics

**delta\_q\_abs** specifies the absolute value of the quantizer index delta value being decoded. If **delta\_q\_abs** is equal to **DELTA\_Q\_SMALL**, the value is encoded using **delta\_q\_rem\_bits** and **delta\_q\_abs\_bits**.

**delta\_q\_rem\_bits** and **delta\_q\_abs\_bits** encode the absolute value of the quantizer index delta value being decoded, where the absolute value of the quantizer index delta value is of the form:

$$(1 \ll \text{delta\_q\_rem\_bits}) + \text{delta\_q\_abs\_bits} + 1$$

**delta\_q\_sign\_bit** equal to 0 indicates that the quantizer index delta value is positive; **delta\_q\_sign\_bit** equal to 1 indicates that the quantizer index delta value is negative.

## 6.10.13. Loop filter delta semantics

**delta\_lf\_abs** specifies the absolute value of the loop filter delta value being decoded. If **delta\_lf\_abs** is equal to **DELTA\_LF\_SMALL**, the value is encoded using **delta\_lf\_rem\_bits** and **delta\_lf\_abs\_bits**.

**delta\_lf\_rem\_bits** and **delta\_lf\_abs\_bits** encode the absolute value of the loop filter delta value being decoded, where the absolute value of the loop filter delta value is of the form:

$$(1 \ll (\text{delta\_lf\_rem\_bits} + 1)) + \text{delta\_lf\_abs\_bits} + 1$$

**delta\_lf\_sign\_bit** equal to 0 indicates that the loop filter delta value is positive; **delta\_lf\_sign\_bit** equal to 1 indicates that the loop filter delta value is negative.

## 6.10.14. CDEF params semantics

**cdef\_damping\_minus\_3** controls the amount of damping in the deringing filter.

**cdef\_bits** specifies the number of bits needed to specify which CDEF filter to apply.

**cdef\_y\_pri\_strength** and **cdef\_uv\_pri\_strength** specify the strength of the primary filter.

**cdef\_y\_sec\_strength** and **cdef\_uv\_sec\_strength** specify the strength of the secondary filter.

## 6.10.15. Loop restoration params semantics

**lr\_type** is used to compute **FrameRestorationType**.

**FrameRestorationType** specifies the type of restoration used for each plane as follows:

<b>lr_type</b>	<b>FrameRestorationType</b>	<b>Name of FrameRestorationType</b>
0	0	RESTORE_NONE
1	3	RESTORE_SWITCHABLE
2	1	RESTORE_WIENER
3	2	RESTORE_SGRPROJ

**UsesLr** indicates if any plane uses loop restoration.

**lr\_unit\_shift** specifies if the luma restoration size should be halved.

**lr\_unit\_extra\_shift** specifies if the luma restoration size should be halved again.

**lr\_uv\_shift** is only present for 4:2:0 formats and specifies if the chroma size should be half the luma size.

**LoopRestorationSize[plane]** specifies the size of loop restoration units in units of samples in the current plane.

## 6.10.16. TX size semantics

**tx\_depth** is used to compute TxSize. tx\_depth is inverted with respect to TxSize, i.e. it specifies how much smaller the transform size should be made than the largest possible transform size for the block.

**TxSize** specifies the transform size to be used for this block:

<b>TxSize</b>	<b>Name of TxSize</b>
0	TX_4X4
1	TX_8X8
2	TX_16X16
3	TX_32X32
4	TX_64X64
5	TX_4X8
6	TX_8X4
7	TX_8X16
8	TX_16X8
9	TX_16X32
10	TX_32X16
11	TX_32X64
12	TX_64X32

<b>TxSize</b>	<b>Name of TxSize</b>
13	TX_4X16
14	TX_16X4
15	TX_8X32
16	TX_32X8
17	TX_16X64
18	TX_64X16

**Note:** TxSize is determined for skipped intra blocks because TxSize controls the granularity of the intra prediction.

## 6.10.17. Block TX size semantics

**InterTxSizes** is an array that holds the transform sizes within inter frames.

**Note:** TxSizes and InterTxSizes contain different values. All the values in TxSizes across a residual block will share the same value, while InterTxSizes can represent several different transform sizes within a residual block.

## 6.10.18. Var TX size semantics

**txfm\_split** equal to 1 specifies that the block should be split into smaller transform sizes. txfm\_split equal to 0 specifies that the block should not be split any more.

## 6.10.19. Transform type semantics

**set** specifies the transform set.

<b>is_inter</b>	<b>set</b>	<b>Name of transform set</b>
Don't care	0	TX_SET_DCTONLY
0	1	TX_SET_INTRA_1
0	2	TX_SET_INTRA_2
1	1	TX_SET_INTER_1
1	2	TX_SET_INTER_2
1	3	TX_SET_INTER_3

The transform sets determine what subset of transform types can be used, according to the following table.

Transform type	TX_SET_DCTONLY	TX_SET_INTRA_1	TX_SET_INTRA_2	TX_SET_INTER_1	TX_SET_INTER_2	TX_SET_INTER_3
DCT_DCT	X	X	X	X	X	X
ADST_DCT		X	X	X	X	
DCT_ADST		X	X	X	X	
ADST_ADST		X	X	X	X	
FLIPADST_DCT				X	X	
DCT_FLIPADST				X	X	
FLIPADST_FLIPADST				X	X	
ADST_FLIPADST				X	X	
FLIPADST_ADST				X	X	
IDTX		X	X	X	X	X
V_DCT		X		X	X	
H_DCT		X		X	X	
V_ADST				X		
H_ADST				X		
V_FLIPADST				X		
H_FLIPADST				X		

**inter\_tx\_type** specifies the transform type for inter blocks.

**intra\_tx\_type** specifies the transform type for intra blocks.

## 6.10.20. Is inter semantics

**is\_inter** equal to 0 specifies that the block is an intra block; **is\_inter** equal to 1 specifies that the block is an inter block.

## 6.10.21. Intra block mode info semantics

This syntax is used when coding an intra block within an inter frame.

**y\_mode** specifies the direction of luminance intra prediction using values with the same interpretation as for **intra\_frame\_y\_mode**.

**uv\_mode** specifies the chrominance intra prediction mode using values with the same interpretation as in the semantics for **intra\_frame\_y\_mode**, with an additional mode **UV\_CFL\_PRED**.

**Note:** Due to the way the **uv\_mode** syntax element is read, **uv\_mode** can only be read as **UV\_CFL\_PRED** when  $\text{Max}(\text{Block\_Width}[\text{MiSize}], \text{Block\_Height}[\text{MiSize}]) \leq 32$ .

## 6.10.22. Inter block mode info semantics

This syntax is used when coding an inter block.

**compound\_mode** specifies how the motion vector used by inter prediction is obtained when using compound prediction. An offset is added to **compound\_mode** to compute YMode as follows:

YMode	Name of YMode
14	NEARESTMV
15	NEARMV
16	GLOBALMV
17	NEWMV
18	NEAREST_NEARESTMV
19	NEAR_NEARMV
20	NEAREST_NEWMV
21	NEW_NEARESTMV
22	NEAR_NEWMV
23	NEW_NEARMV
24	GLOBAL_GLOBALMV
25	NEW_NEWMV

**Note:** The intra modes take values 0..13 so these YMode values start at 14.

**new\_mv** equal to 0 means that a motion vector difference should be read.

**zero\_mv** equal to 0 means that the motion vector should be set equal to default motion for the frame.

**ref\_mv** equal to 0 means that the most likely motion vector should be used (called NEAREST), **ref\_mv** equal to 1 means that the second most likely motion vector should be used (called NEAR).

**interp\_filter** specifies the type of filter used in inter prediction. Values 0..3 are allowed with the same interpretation as for **interpolation\_filter**. One filter type is specified for the vertical filter direction and one for the horizontal filter direction.

**Note:** The syntax element **interpolation\_filter** from the uncompressed header can specify the type of filter to be used for the whole frame. If it is set to SWITCHABLE then the **interp\_filter** syntax element is read from the bitstream for every inter block.

**RefMvldx** specifies which candidate in the RefStackMv should be used.

**drl\_mode** is a bit sent for candidates in the motion vector stack to indicate if they should be used. drl\_mode equal to 0 means to use the current value of idx. drl\_mode equal to 1 says to continue searching. DRL stands for “Dynamic Reference List”.

## 6.10.23. Filter intra mode info semantics

**use\_filter\_intra** is a bit specifying whether or not intra filtering can be used.

**filter\_intra\_mode** specifies the type of intra filtering, and can take on any of the following values:

filter_intra_mode	Name of filter_intra_mode
0	FILTER_DC_PRED
1	FILTER_V_PRED
2	FILTER_H_PRED
3	FILTER_D157_PRED
4	FILTER_PAETH_PRED

## 6.10.24. Ref frames semantics

**comp\_mode** specifies whether single or compound prediction is used:

comp_mode	Name of comp_mode
0	SINGLE_REFERENCE
1	COMPOUND_REFERENCE

**SINGLE\_REFERENCE** indicates that the inter block uses only a single reference frame to generate motion compensated prediction.

**COMPOUND\_REFERENCE** indicates that the inter block uses compound mode.

There are two reference frame groups:

- Group 1: LAST\_FRAME, LAST2\_FRAME, LAST3\_FRAME, and GOLDEN\_FRAME.
- Group 2: BWDREF\_FRAME, ALTREF2\_FRAME, and ALTREF\_FRAME.

**Note:** Encoders are free to assign these references to any of the reference frames (via the ref\_frame\_idx array). For example, there is no requirement of bitstream conformance that LAST\_FRAME should indicate a frame that appears before the current frame in output order. Similarly, encoders can assign multiple references to the same reference frame.

**comp\_ref\_type** is used for compound prediction to specify whether both reference frames come from the same group or not:

comp_ref_type	Name of comp_ref_type	Description
0	UNIDIR_COMP_REFERENCE	Both reference frames from the same group
1	BIDIR_COMP_REFERENCE	One from Group 1 and one from Group 2

**uni\_comp\_ref**, **uni\_comp\_ref\_p1**, and **uni\_comp\_ref\_p2** specify which reference frames are in use when both come from the same group.

**comp\_ref**, **comp\_ref\_p1**, and **comp\_ref\_p2** specify the first reference frame when the two reference frames come from different groups.

**comp\_bwdref** and **comp\_bwdref\_p1** specify the second reference frame when the two reference frames come from different groups.

**single\_ref\_p1**, **single\_ref\_p2**, **single\_ref\_p3**, **single\_ref\_p4**, **single\_ref\_p5**, and **single\_ref\_p6** specify the reference frame when only a single reference frame is in use.

**RefFrame[ 0 ]** specifies which frame is used to compute the predicted samples for this block:

RefFrame[ 0 ]	Name of ref_frame
0	INTRA_FRAME
1	LAST_FRAME
2	LAST2_FRAME
3	LAST3_FRAME
4	GOLDEN_FRAME
5	BWDREF_FRAME
6	ALTREF2_FRAME
7	ALTREF_FRAME

**RefFrame[ 1 ]** specifies which additional frame is used in compound prediction:

RefFrame[ 1 ]	Name of ref_frame
-1	NONE (this block uses single prediction)
0	INTRA_FRAME (this block uses interintra prediction)
1	LAST_FRAME
2	LAST2_FRAME

RefFrame[ 1 ]	Name of ref_frame
3	LAST3_FRAME
4	GOLDEN_FRAME
5	BWDREF_FRAME
6	ALTREF2_FRAME
7	ALTREF_FRAME

**Note:** Not all combinations of RefFrame[0] and RefFrame[1] can be coded.

## 6.10.25. Assign mv semantics

It is a requirement of bitstream conformance that whenever `assign_mv` returns, the function `is_mv_valid(isCompound)` would return 1, where `is_mv_valid` is defined as:



```

is_mv_valid( isCompound ) {
    for ( i = 0; i < 1 + isCompound; i++ ) {
        for ( comp = 0; comp < 2; comp++ ) {
            if ( Abs( Mv[ i ][ comp ] ) >= ( 1 << 14 ) )
                return 0
        }
    }
    if ( !use_intrabc ) {
        return 1
    }
    bw = Block_Width[ MiSize ]
    bh = Block_Height[ MiSize ]
    if ( (Mv[ 0 ][ 0 ] & 7) || (Mv[ 0 ][ 1 ] & 7) ) {
        return 0
    }
    deltaRow = Mv[ 0 ][ 0 ] >> 3
    deltaCol = Mv[ 0 ][ 1 ] >> 3
    srcTopEdge = MiRow * MI_SIZE + deltaRow
    srcLeftEdge = MiCol * MI_SIZE + deltaCol
    srcBottomEdge = srcTopEdge + bh
    srcRightEdge = srcLeftEdge + bw
    if ( HasChroma ) {
        if ( bw < 8 && subsampling_x )
            srcLeftEdge -= 4
        if ( bh < 8 && subsampling_y )
            srcTopEdge -= 4
    }
    if ( srcTopEdge < MiRowStart * MI_SIZE ||
        srcLeftEdge < MiColStart * MI_SIZE ||
        srcBottomEdge > MiRowEnd * MI_SIZE ||
        srcRightEdge > MiColEnd * MI_SIZE ) {
        return 0
    }
    sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64
    sbH = Block_Height[ sbSize ]
    activeSbRow = (MiRow * MI_SIZE) / sbH
    activeSb64Col = (MiCol * MI_SIZE) >> 6
    srcSbRow = (srcBottomEdge - 1) / sbH
    srcSb64Col = (srcRightEdge - 1) >> 6
    totalSb64PerRow = ((MiColEnd - MiColStart - 1) >> 4) + 1
    activeSb64 = activeSbRow * totalSb64PerRow + activeSb64Col
    srcSb64 = srcSbRow * totalSb64PerRow + srcSb64Col
    if ( srcSb64 >= activeSb64 - INTRABC_DELAY_SB64 ) {
        return 0
    }
    gradient = 1 + INTRABC_DELAY_SB64 + use_128x128_superblock
    wfOffset = gradient * (activeSbRow - srcSbRow)
    if ( srcSbRow > activeSbRow ||
        srcSb64Col >= activeSb64Col - INTRABC_DELAY_SB64 + wfOffset ) {
        return 0
    }
}

```

```

    }
    return 1
}

```

**Note:** The purpose of this function is to limit the maximum size of motion vectors and also, if `use_intrabc` is equal to 1, to additionally constrain the motion vector in order that the data is fetched from parts of the tile that have already been decoded, and that are not too close to the current block (in order to make a pipelined decoder implementation feasible).

## 6.10.26. Read motion mode semantics

`use_obmc` equal to 1 means that OBMC should be used. `use_obmc` equal to 0 means that simple translation should be used.

`motion_mode` specifies the type of motion compensation to perform:

<code>motion_mode</code>	Name of <code>motion_mode</code>
0	SIMPLE
1	OBMC
2	LOCALWARP

**Note:** A `motion_mode` equal to SIMPLE is used for blocks requiring global motion.

## 6.10.27. Read inter intra semantics

`interintra` equal to 1 specifies that an inter prediction should be blended with an intra prediction.

`interintra_mode` specifies the type of intra prediction to be used:

<code>interintra_mode</code>	Name of <code>interintra_mode</code>
0	II_DC_PRED
1	II_V_PRED
2	II_H_PRED
3	II_SMOOTH_PRED

**wedge\_interintra** equal to 1 specifies that wedge blending should be used. **wedge\_interintra** equal to 0 specifies that intra blending should be used.

**wedge\_index** is used to derive the direction and offset of the wedge mask used during blending.

## 6.10.28. Read compound type semantics

**comp\_group\_idx** equal to 0 indicates that the **compound\_idx** syntax element should be read. **comp\_group\_idx** equal to 1 indicates that the **compound\_idx** syntax element is not present.

**compound\_idx** equal to 1 indicates that a distance based weighted scheme should be used for blending. **compound\_idx** equal to 0 indicates that the averaging scheme should be used for blending.

**compound\_type** specifies how the two predictions should be blended together:

<b>compound_type</b>	<b>Name of compound_type</b>
0	COMPOUND_WEDGE
1	COMPOUND_DIFFWTD
2	COMPOUND_AVERAGE
3	COMPOUND_INTRA
4	COMPOUND_DISTANCE

**Note:** COMPOUND\_AVERAGE, COMPOUND\_INTRA, and COMPOUND\_DISTANCE cannot be directly signaled with the **compound\_type** syntax element but are inferred from other syntax elements.

**wedge\_index** is used to derive the direction and offset of the wedge mask used during blending.

**wedge\_sign** specifies the sign of the wedge blend.

**mask\_type** specifies the type of mask to be used during blending:

<b>mask_type</b>	<b>Name of mask_type</b>
0	UNIFORM_45
1	UNIFORM_45_INV

## 6.10.29. MV semantics

**MvCtx** is used to determine which CDFs to use for the motion vector syntax elements.

**mv\_joint** specifies which components of the motion vector difference are non-zero:

<b>mv_joint</b>	<b>Name of mv_joint</b>	<b>Changes row</b>	<b>Changes col</b>
0	MV_JOINT_ZERO	No	No
1	MV_JOINT_HNZVZ	No	Yes
2	MV_JOINT_HZVNZ	Yes	No
3	MV_JOINT_HNZVNZ	Yes	Yes

The motion vector difference is added to the PredMv to compute the final motion vector in Mv.

### 6.10.30. MV component semantics

**mv\_sign** equal to 0 means that the motion vector difference is positive; mv\_sign equal to 1 means that the motion vector difference is negative.

**mv\_class** specifies the class of the motion vector difference. A higher class means that the motion vector difference represents a larger update:

<b>mv_class</b>	<b>Name of mv_class</b>
0	MV_CLASS_0
1	MV_CLASS_1
2	MV_CLASS_2
3	MV_CLASS_3
4	MV_CLASS_4
5	MV_CLASS_5
6	MV_CLASS_6
7	MV_CLASS_7
8	MV_CLASS_8
9	MV_CLASS_9
10	MV_CLASS_10

**mv\_class0\_bit** specifies the integer part of the motion vector difference. This is only present for class 0 motion vector differences.

**mv\_class0\_fr** specifies the first 2 fractional bits of the motion vector difference. This is only present for class 0 motion vector differences.

**mv\_class0\_hp** specifies the third fraction bit of the motion vector difference. This is only present for class 0 motion vector differences.

**mv\_bit** specifies bit *i* of the integer part of the motion vector difference.

**mv\_fr** specifies the first 2 fractional bits of the motion vector difference.

**mv\_hp** specifies the third fractional bit of the motion vector difference.

## 6.10.31. Compute prediction semantics

The prediction for inter and interintra blocks is triggered within `compute_prediction`. However, intra prediction is done at the transform block granularity so `predict_intra` is also called from `transform_block`.

**predW** and **predH** are variables containing the smallest size that can be used for inter prediction. (This size may be increased for chroma blocks if not all blocks use inter prediction.)

**predict\_inter** is a function call that indicates the conceptual point where inter prediction happens. When this function is called, the inter prediction process specified in [section 7.11.3](#) is invoked.

**predict\_intra** is a function call that indicates the conceptual point where intra prediction happens. When this function is called, the intra prediction process specified in [section 7.11.2](#) is invoked.

**Note:** The `predict_inter` and `predict_intra` functions do not affect the syntax decode process.

**someUseIntra** is a variable that indicates if some of the blocks corresponding to this residual require intra prediction.

**Note:** The chroma residual block size is always at least 4 in width and height. This means that no transform width or height smaller than 4 is required. As such, a chroma residual may actually cover several luma blocks. If any of these blocks are intra, a single prediction is performed for the entire chroma residual block based on the mode info of the bottom right luma block. However, if all the constituent blocks are inter blocks, a special case is triggered and inter prediction is done using the smaller chroma block size that corresponds to each of the luma blocks.

## 6.10.32. Residual semantics

The residual consists of a number of transform blocks.

If the block is wider or higher than 64 luma samples, then the residual is split into 64 by 64 chunks.

Within each chunk, the transform blocks are either sent in raster order (if `use_inter` is equal to 0 or `LossLess` is equal to 1), or within a recursive transform tree.

## 6.10.33. Transform block semantics

**reconstruct** is a function call that indicates the conceptual point where inverse transform and reconstruction happens. When this function is called, the reconstruction process specified in [section 7.12.3](#) is invoked.

**predict\_palette** is a function call that indicates the conceptual point where palette prediction happens. When this function is called, the palette prediction process specified in [section 7.11.4](#) is invoked.

**predict\_chroma\_from\_luma** is a function call that indicates the conceptual point where predicting chroma from luma happens. When this function is called, the predict chroma from luma process specified in [section 7.11.5](#) is invoked.

**MaxLumaW** and **MaxLumaH** are needed for chroma from luma prediction and store the extent of luma samples that can be used for prediction.

**LoopfilterTxSizes** is an array that stores the transform size for each plane and position for use in loop filtering. `LoopfilterTxSizes[ plane ][ row ][ col ]` stores the transform size where row and col are in units of 4x4 samples.

**Note:** The transform size is always equal for planes 1 and 2.

## 6.10.34. Coefficients semantics

**TxTypes** is an array which stores at a 4x4 luma sample granularity the transform type to be used.

**Note:** The transform type is only read for luma transform blocks, the chroma uses the transform type for a corresponding luma block. Chroma blocks will only use transform types that have been written for the current residual block.

**Quant** is an array storing the quantised coefficients for the current transform block.

**all\_zero** equal to 1 specifies that all coefficients are zero.

**Note:** The transform type is only present when this is a luminance block and **all\_zero** is equal to 0. If **all\_zero** is equal to 1 for a luminance block, the transform type is set to DCT\_DCT.

**eob\_extra** and **eob\_extra\_bit** specify the position of the last non-zero coefficient by being used to compute the variable **eob**.

**eob\_pt\_16**, **eob\_pt\_32**, **eob\_pt\_64**, **eob\_pt\_128**, **eob\_pt\_256**, **eob\_pt\_512**, **eob\_pt\_1024**: syntax elements used to compute **eob**.

**eob** is a variable that indicates the index of the end of block. This index is equal to one plus the index of the last non-zero coefficient.

**coeff\_base\_eob** is a syntax element used to compute the base level of the last non-zero coefficient.

**Note:** The base level is set to **coeff\_base\_eob** plus 1. Since this coefficient is known to be non-zero, only base levels of 1, 2, or 3 can be coded via **coeff\_base\_eob**.

**coeff\_base** specifies the base level of a coefficient (this syntax element is used for all coefficients except the last non-zero coefficient).

**Note:** The base level can take values of 0, 1, 2, or 3. If the base level is less than 3, then it contains the actual level of the coefficient. Otherwise, the syntax element `coeff_br` is used to optionally increase the level.

**dc\_sign** specifies the sign of the DC coefficient.

**sign\_bit** specifies the sign of a non-zero AC coefficient.

**coeff\_br** specifies an increment to the coefficient.

**Note:** Each quantized coefficient can use `coeff_br` to provide up to 4 increments. If an increment equal to less than 3 is coded, it signifies that this was the final increment.

**golomb\_length\_bit** is used to compute the number of extra bits required to code the coefficient.

If length is equal to 20, it is a requirement of bitstream conformance that `golomb_length_bit` is equal to 1.

**golomb\_data\_bit** specifies the value of one of the extra bits.

**AboveLevelContext** and **LeftLevelContext** are arrays that store at a 4 sample granularity the cumulative sum of coefficient levels.

**AboveDcContext** and **LeftDcContext** are arrays that store at a 4 sample granularity 2 bits signaling the sign of the DC coefficient (zero being counted as a separate sign).

## 6.10.35. Intra angle info semantics

**angle\_delta\_y** specifies the offset to be applied to the intra prediction angle specified by the prediction mode in the luma plane, biased by `MAX_ANGLE_DELTA` so as to encode a positive value.

**angle\_delta\_uv** specifies the offset to be applied to the intra prediction angle specified by the prediction mode in the chroma plane biased by `MAX_ANGLE_DELTA` so as to encode a positive value.

**AngleDeltaY** is computed from `angle_delta_y` by removing the `MAX_ANGLE_DELTA` offset to produce the final luma angle offset value, which may be positive or negative.

**AngleDeltaUV** is computed from `angle_delta_uv` by removing the `MAX_ANGLE_DELTA` offset to produce the final chroma angle offset value, which may be positive or negative.

## 6.10.36. Read CFL alphas semantics

**cfl\_alpha\_signs** contains the sign of the alpha values for U and V packed together into a single syntax element with 8 possible values. (The combination of two zero signs is prohibited as it is redundant with DC Intra prediction.)

<b>cfl_alpha_signs</b>	<b>Name of signU</b>	<b>Name of signV</b>
0	CFL_SIGN_ZERO	CFL_SIGN_NEG

<b>cfl_alpha_signs</b>	<b>Name of signU</b>	<b>Name of signV</b>
1	CFL_SIGN_ZERO	CFL_SIGN_POS
2	CFL_SIGN_NEG	CFL_SIGN_ZERO
3	CFL_SIGN_NEG	CFL_SIGN_NEG
4	CFL_SIGN_NEG	CFL_SIGN_POS
5	CFL_SIGN_POS	CFL_SIGN_ZERO
6	CFL_SIGN_POS	CFL_SIGN_NEG
7	CFL_SIGN_POS	CFL_SIGN_POS

**signU** contains the sign of the alpha value for the U component:

<b>signU</b>	<b>Name of signU</b>
0	CFL_SIGN_ZERO
1	CFL_SIGN_NEG
2	CFL_SIGN_POS

**signV** contains the sign of the alpha value for the V component with the same interpretation as for signU.

**cfl\_alpha\_u** contains the absolute value of alpha minus one for the U component.

**cfl\_alpha\_v** contains the absolute value of alpha minus one for the V component.

**CflAlphaU** contains the signed value of the alpha component for the U component.

**CflAlphaV** contains the signed value of the alpha component for the V component.

## 6.10.37. Palette mode info semantics

**has\_palette\_y** is a boolean value specifying whether a palette is encoded for the Y plane.

**has\_palette\_uv** is a boolean value specifying whether a palette is encoded for the UV plane.

**palette\_size\_y\_minus\_2** is used to compute `PaletteSizeY`.

**PaletteSizeY** is a variable holding the Y plane palette size.

**palette\_size\_uv\_minus\_2** is used to compute `PaletteSizeUV`.

**PaletteSizeUV** is a variable holding the UV plane palette size.

**use\_palette\_color\_cache\_y**, if equal to 1, indicates that for a particular palette entry in the luma palette, the cached entry should be used.



**use\_palette\_color\_cache\_u**, if equal to 1, indicates that for a particular palette entry in the U chroma palette, the cached entry should be used.

**palette\_colors\_y** is an array holding the Y plane palette colors.

**palette\_colors\_u** is an array holding the U plane palette colors.

**palette\_colors\_v** is an array holding the V plane palette colors.

**delta\_encode\_palette\_colors\_v**, if equal to 1, indicates that the V chroma palette is encoded using delta encoding.

**palette\_num\_extra\_bits\_y** is used to calculate the number of bits used to store each palette delta value for the luma palette.

**palette\_num\_extra\_bits\_u** is used to calculate the number of bits used to store each palette delta value for the U chroma palette.

**palette\_num\_extra\_bits\_v** is used to calculate the number of bits used to store each palette delta value for the V chroma palette.

**palette\_delta\_y** is a delta value for the luma palette.

**palette\_delta\_u** is a delta value for the U chroma palette.

**palette\_delta\_v** is a delta value for the V chroma palette.

**Note:** Luma and U delta values give a positive offset relative to the previous palette entry in the same plane. V delta values give a signed offset relative to the U palette entries.

**palette\_delta\_sign\_bit\_v**, if equal to 1, indicates that the decoded V chroma palette delta value should be negated.

## 6.10.38. Palette tokens semantics

**color\_index\_map\_y** holds the index in **palette\_colors\_y** for the block's Y plane top left sample.

**color\_index\_map\_uv** holds the index in **palette\_colors\_u** and **palette\_colors\_v** for the block's UV plane top left sample.

**palette\_color\_idx\_y** holds the index in **ColorOrder** for a sample in the block's Y plane.

**palette\_color\_idx\_uv** holds the index in **ColorOrder** for a sample in the block's UV plane.

## 6.10.39. Palette color context semantics

**ColorOrder** is an array holding the mapping from an encoded index to the palette. **ColorOrder** is ranked in order of frequency of occurrence of each color in the neighborhood of the current block, weighted by closeness to the current block.

**ColorContextHash** is a variable derived from the distribution of colors in the neighborhood of the current block, which is used to determine the probability context used to decode `palette_color_idx_y` and `palette_color_idx_uv`.

## 6.10.40. Read CDEF semantics

**cdef\_idx** specifies which CDEF filtering parameters should be used for a particular 64 by 64 block. A value of -1 means that CDEF is disabled for that block.

## 6.10.41. Read loop restoration unit semantics

**use\_wiener** specifies if the Wiener filter should be used.

**use\_sgrproj** specifies if the self guided filter should be used.

**restoration\_type** specifies the restoration filter that should be used with the same interpretation as `FrameRestorationType`.

**lr\_sgr\_set** specifies which set of parameters to use for the self guided filter.

**subexp\_more\_bools** equal to 0 specifies that the parameter is in the range  $mk$  to  $mk+a-1$ . **subexp\_more\_bools** equal to 1 specifies that the parameter is greater than  $mk+a-1$ .

**subexp\_unif\_bools** specifies the value of the parameter minus  $mk$ .

**subexp\_bools** specifies the value of the parameter minus  $mk$ .

## 6.11. Tile list OBU semantics

### 6.11.1. General tile list OBU semantics

**output\_frame\_width\_in\_tiles\_minus\_1** plus one is the width of the output frame, in tile units.

**output\_frame\_height\_in\_tiles\_minus\_1** plus one is the height of the output frame, in tile units.

**tile\_count\_minus\_1** plus one is the number of `tile_list_entry` in the list.

It is a requirement of bitstream conformance that `tile_count_minus_1` is less than or equal to 511.

### 6.11.2. Tile list entry semantics

**anchor\_frame\_idx** is the index into an array `AnchorFrames` of the frames that the tile uses for prediction. The `AnchorFrames` array is provided by external means and may change for each tile list OBU. The process for creating the `AnchorFrames` array is outside of the scope of this specification.

It is a requirement of bitstream conformance that `anchor_frame_idx` is less than or equal to 127.

**anchor\_tile\_row** is the row coordinate of the tile in the frame that it belongs, in tile units.

It is a requirement of bitstream conformance that `anchor_tile_row` is less than `TileRows`.

**anchor\_tile\_col** is the column coordinate of the tile in the frame that it belongs, in tile units.

It is a requirement of bitstream conformance that **anchor\_tile\_col** is less than **TileCols**.

**tile\_data\_size\_minus\_1** plus one is the size of the coded tile data, **coded\_tile\_data**, in bytes.

**coded\_tile\_data** are the **tile\_data\_size\_minus\_1** + 1 bytes of the coded tile.

# 7. Decoding process

## 7.1. Overview

AV1 contains two operating modes:

1. General decoding (input is a sequence of OBUs, output is decoded frames)
2. Large scale tile decoding (input is a tile list OBU plus additional side information, output is a decoded frame)

The general decoding process is specified in [section 7.2](#).

The large scale tile decoding process is specified in [section 7.3](#).

## 7.2. General decoding process

Decoders shall produce output frames that are identical in all respects and have the same output order as those produced by the decoding process specified herein.

The input to this process is a sequence of open bitstream units (OBUs).

The output from this process is a sequence of decoded frames.

For each OBU in turn the syntax elements are extracted as specified in [section 5.3](#).

The syntax tables include function calls indicating when the remaining decode processes are triggered.

## 7.3. Large scale tile decoding process

### 7.3.1. General

The large scale tile decoding process is used to decode a random subset of tiles taken from a number of coded frames. The list of tiles is specified by a tile list OBU. One possible use case for this process is described in Annex D.

**Note:** A decoder is recommended to support decoding of tile list OBUs, but this is not a requirement for decoder conformance.

The inputs to this process are:

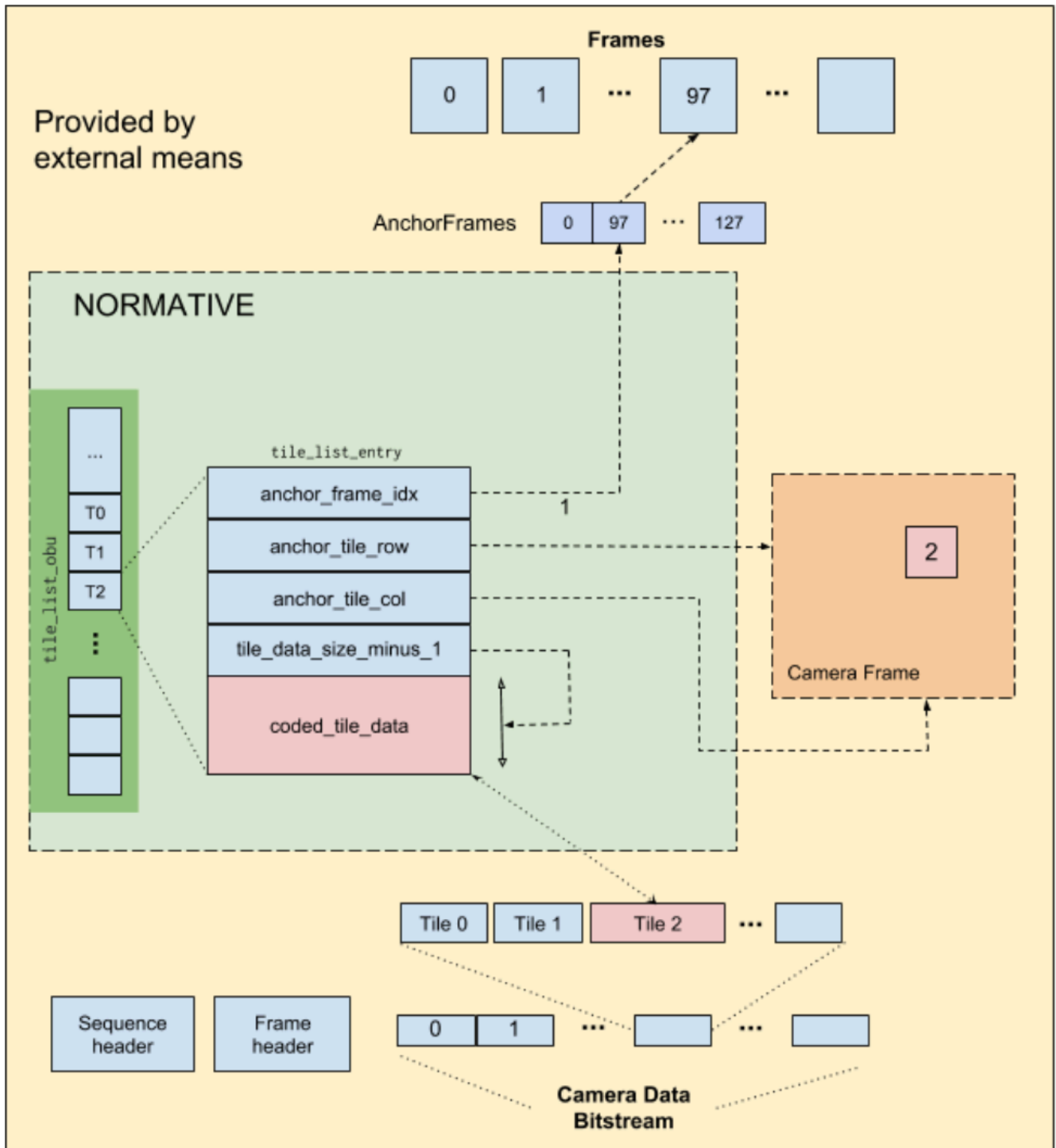
- contents of all syntax elements and variables produced when parsing a sequence header OBU,
- contents of all syntax elements and variables produced when parsing a frame header OBU (including CDF tables optionally loaded from a reference frame),
- an array AnchorFrames containing up to 128 frames,
- a tile list OBU.

The output from this process is:

- an output frame containing decoded tiles in raster order.

**Note:** The syntax elements from the sequence header and frame header may be produced by decoding a sequence header OBU and a frame header OBU, but this is not a requirement of decoder conformance. The AnchorFrames may be produced by decoding an AV1 bitstream, but this is not a requirement of bitstream conformance.

The following figure shows the arrangement of data required to decode a single tile list OBU. Those data shown on a green background are normatively defined in this specification. Data items shown on a yellow background are defined by a process or processes beyond the scope of this specification.



Application level overview

For each tile list entry in the tile list OBU, the following ordered steps are applied:

1. Parse the syntax elements within the `tile_list_entry`

2. Set the bitstream position indicator to point to the the start of the coded\_tile\_data syntax element
3. Set the variable last equal to ref\_frame\_idx[ 0 ]
4. Set FrameStore[ last ] equal to AnchorFrames[ anchor\_frame\_idx ]
5. RefValid[ last ] is set equal to 1.
6. RefUpscaledWidth[ last ] is set equal to UpscaledWidth.
7. RefFrameWidth[ last ] is set equal to FrameWidth.
8. RefFrameHeight[ last ] is set equal to FrameHeight.
9. RefMiCols[ last ] is set equal to MiCols.
10. RefMiRows[ last ] is set equal to MiRows.
11. RefSubsamplingX[ last ] is set equal to subsampling\_x.
12. RefSubsamplingY[ last ] is set equal to subsampling\_y.
13. RefBitDepth[ last ] is set equal to BitDepth.
14. Invoke the decode camera tile process specified in [section 7.3.2](#) and write the decoded tiles into an output frame in raster order, in the order that they occur in the tile list OBU.

The output from this process is the output frame that is built up in the final step above.

The variable outputW is defined as  $( 1 + \text{output\_frame\_width\_in\_tiles\_minus\_1} ) * \text{TileWidth}$ .

The variable outputH is defined as  $( 1 + \text{output\_frame\_height\_in\_tiles\_minus\_1} ) * \text{TileHeight}$ .

The operation of writing a decoded tile (with zero-based index given by the variable tile) into the output frame in raster order is defined as follows:

```

destX = TileWidth * ( tile % (output_frame_width_in_tiles_minus_1 + 1) )
destY = TileHeight * ( tile / (output_frame_width_in_tiles_minus_1 + 1) )
w = TileWidth
h = TileHeight
for ( y = 0; y < h; y++ ) {
    for ( x = 0; x < w; x++ ) {
        OutputFrameY[ y + destY ][ x + destX ] = OutY[ y ][ x ]
    }
}
w = w >> subsampling_x
h = h >> subsampling_y
destX = destX >> subsampling_x
destY = destY >> subsampling_y
for ( y = 0; y < h; y++ ) {
    for ( x = 0; x < w; x++ ) {
        OutputFrameU[ y + destY ][ x + destX ] = OutU[ y ][ x ]
        OutputFrameV[ y + destY ][ x + destX ] = OutV[ y ][ x ]
    }
}

```

OutputFrameY (representing the luma plane of the output frame) is outputW samples across by outputH samples down.

OutputFrameU (representing the U plane of the output frame) is ( outputW >> subsampling\_x ) samples across by ( outputH >> subsampling\_y ) samples down.

OutputFrameV (representing the V plane of the output frame) is ( outputW >> subsampling\_x ) samples across by ( outputH >> subsampling\_y ) samples down.

The bitdepth of each output sample is given by BitDepth.

The output frame may not be fully covered with decoded tiles. The decoder should not modify samples in the output frame outside of the boundaries of the decoded tiles.

Decoders that support large scale tile decoding shall produce output frames that are identical in all respects as those produced by this decoding process.

It is a requirement of bitstream conformance that the following conditions are met:

- enable\_superres is equal to 0
- enable\_order\_hint is equal to 0
- still\_picture is equal to 0
- film\_grain\_params\_present is equal to 0
- timing\_info\_present\_flag is equal to 0
- decoder\_model\_info\_present\_flag is equal to 0



- `initial_display_delay_present_flag` is equal to 0
- `enable_restoration` is equal to 0
- `enable_cdef` is equal to 0
- `mono_chrome` is equal to 0
- `TileHeight` is equal to  $(\text{use\_128x128\_superblock} ? 128 : 64)$  for all tiles (i.e. the tile is exactly one superblock high)
- `TileWidth` is identical for all tiles and is an integer multiple of `TileHeight` (i.e. the tile is an integer number of superblocks wide)
- `FrameWidth` is equal to  $\text{MiCols} * \text{MI\_SIZE}$
- `FrameHeight` is equal to  $\text{MiRows} * \text{MI\_SIZE}$
- `show_existing_frame` is equal to 0
- `frame_type` is equal to `INTER_FRAME`
- `show_frame` is equal to 1
- `error_resilient_mode` is equal to 0
- `disable_cdf_update` is equal to 1
- `disable_frame_end_update_cdf` is equal to 1
- `delta_lf_present` is equal to 0
- `delta_q_present` is equal to 0
- `frame_size_override_flag` is equal to 0
- `refresh_frame_flags` is equal to 0
- `use_ref_frame_mvs` is equal to 0
- `segmentation_temporal_update` is equal to 0
- `reference_select` is equal to 0
- `loop_filter_level[ 0 ]` and `loop_filter_level[ 1 ]` are equal to 0
- $\text{tile\_count\_minus\_1} + 1$  is less than or equal to  $(\text{output\_frame\_width\_in\_tiles\_minus\_1} + 1) * (\text{output\_frame\_height\_in\_tiles\_minus\_1} + 1)$ .

## 7.3.2. Decode camera tile process

This process decodes a single tile within a frame.

The output of this process are arrays OutY, OutU, OutV containing the decoded samples for the tile.

**Note:** The decoding process defined here does not invoke the post-processing steps of deblock, cdef, superres, loop restoration and reference frame update. Implementations may choose to implement this process by using the general decode process with these tools disabled.

The process is specified as:

```

CurrentQIndex = base_q_idx
init_symbol( tile_data_size_minus_1 + 1 )
clear_above_context( )
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64
sbSize4 = Num_4x4_Blocks_Wide[ sbSize ]
MiRowStart = MiRowStarts[ anchor_tile_row ]
MiRowEnd = MiRowStarts[ anchor_tile_row + 1 ]
MiColStart = MiColStarts[ anchor_tile_col ]
MiColEnd = MiColStarts[ anchor_tile_col + 1 ]
for ( r = MiRowStart; r < MiRowEnd; r += sbSize4 ) {
    clear_left_context( )
    for ( c = MiColStart; c < MiColEnd; c += sbSize4 ) {
        ReadDeltas = delta_q_present
        clear_block_decoded_flags( c < ( MiColEnd - 1 ) )
        decode_partition( r, c, sbSize )
    }
}
exit_symbol( )
w = (MiColEnd - MiColStart) * MI_SIZE
h = (MiRowEnd - MiRowStart) * MI_SIZE
x0 = MiColStart * MI_SIZE
y0 = MiRowStart * MI_SIZE
subX = subsampling_x
subY = subsampling_y
xC0 = ( MiColStart * MI_SIZE ) >> subX
yC0 = ( MiRowStart * MI_SIZE ) >> subY

```

**Note:** The intention is that the same decoding process for tile data can be used as for the general decoding process.

It is a requirement of bitstream conformance that the following conditions are met whenever the parsing process returns from the read\_ref\_frames syntax:

- RefFrame[ 0 ] = LAST\_FRAME

- `RefFrame[ 1 ] = NONE`

**Note:** It is allowed to use intra blocks, they are not forbidden by this constraint because intra blocks do not invoke the `read_ref_frames` syntax.

Arrays `OutY`, `OutU`, `OutV` (representing the decoded samples for the tile) are specified as:

- The array `OutY` is `w` samples across by `h` samples down and the sample at location `x` samples across and `y` samples down is given by `OutY[ y ][ x ] = CurrFrame[ 0 ][ y0 + y ][ x0 + x ]` with `x = 0..w - 1` and `y = 0..h - 1`.
- The array `OutU` is `(w + subX) >> subX` samples across by `(h + subY) >> subY` samples down and the sample at location `x` samples across and `y` samples down is given by `OutU[ y ][ x ] = CurrFrame[ 1 ][ yC0 + y ][ xC0 + x ]` with `x = 0..(w >> subX) - 1` and `y = 0..(h >> subY) - 1`.
- The array `OutV` is `(w + subX) >> subX` samples across by `(h + subY) >> subY` samples down and the sample at location `x` samples across and `y` samples down is given by `OutV[ y ][ x ] = CurrFrame[ 2 ][ yC0 + y ][ xC0 + x ]` with `x = 0..(w >> subX) - 1` and `y = 0..(h >> subY) - 1`.

The output of this process is arrays `OutY`, `OutU`, `OutV` representing the Y, U, and V samples.

## 7.4. Decode frame wrapup process

This process is triggered by a call to `decode_frame_wrapup` from within the syntax tables.

At this stage, all the tile level decode has been done, and this process performs any frame level decode that is required.

If `show_existing_frame` is equal to 0, the process first performs any post processing filtering by the following ordered steps:

1. If `loop_filter_level[ 0 ]` is not equal to 0 or `loop_filter_level[ 1 ]` is not equal to 0, the loop filter process specified in [section 7.14](#) is invoked (this process modifies the contents of `CurrFrame`).
2. The CDEF process specified in [section 7.15](#) is invoked (this process takes `CurrFrame` and produces `CdefFrame`).
3. The upscaling process specified in [section 7.16](#) is invoked with `CdefFrame` as input and the output is assigned to `UpscaledCdefFrame`.
4. The upscaling process specified in [section 7.16](#) is invoked with `CurrFrame` as input and the output is assigned to `UpscaledCurrFrame`.
5. The loop restoration process specified in [section 7.17](#) is invoked (this process takes `UpscaledCurrFrame` and `UpscaledCdefFrame` and produces `LrFrame`).
6. The motion field motion vector storage process specified in [section 7.19](#) is invoked.

7. If `segmentation_enabled` is equal to 1 and `segmentation_update_map` is equal to 0, `SegmentIds[ row ][ col ]` is set equal to `PrevSegmentIds[ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.

Otherwise (`show_existing_frame` is equal to 1), if `frame_type` is equal to `KEY_FRAME`, the reference frame loading process as specified in [section 7.21](#) is invoked (this process loads frame state from the reference frames into the current frame state variables).

The following ordered steps now apply:

1. The reference frame update process as specified in [section 7.20](#) is invoked (this process saves the current frame state into the reference frames).
2. If `show_frame` is equal to 1 or `show_existing_frame` is equal to 1, the output process as specified in [section 7.18](#) is invoked (this will output the current frame or a saved frame).

**Note:** Although it is specified that all samples in `CurrFrame` are upsampled, at most 2 lines above and below each stripe (defined by `StripeStartY` and `StripeEndY`) will end up being read. Implementations may wish to avoid upscaling the unused lines.

## 7.5. Ordering of OBUs

A bitstream conforming to this specification consists of one or more coded video sequences.

A coded video sequence consists of one or more temporal units. A temporal unit consists of a series of OBUs starting from a temporal delimiter, optional sequence headers, optional metadata OBUs, a sequence of one or more frame headers, each followed by zero or more tile group OBUs as well as optional padding OBUs.

If scalability is not being used (`OperatingPointIdc` equal to 0), then all frames are part of the operating point. The following constraints must hold:

- The first frame header must have `frame_type` equal to `KEY_FRAME` and `show_frame` equal to 1.
- Each temporal unit must have exactly one shown frame.

If scalability is being used (`OperatingPointIdc` not equal to 0), then only a subset of frames are part of the operating point. For each operating point, the following constraints must hold:

- The first frame header that will be decoded must have `frame_type` equal to `KEY_FRAME` and `show_frame` equal to 1.
- Every layer that has a coded frame in a temporal unit must have exactly one shown frame that is the last frame of that layer in the temporal unit.

**Note:** A shown frame is either a frame with `show_frame` equal to 1, or with `show_existing_frame` equal to 1.

A frame header and its associated tile group OBUs within a temporal unit must use the same value of `obu_extension_flag` (i.e., either both include or both not include the optional OBU extension header).

The value of `temporal_id` must be the same in all OBU extension headers that are contained in the same temporal unit.

If a coded video sequence contains at least one enhancement layer (OBUs with `spatial_id` greater than 0 or `temporal_id` greater than 0) then all frame headers and tile group OBUs associated with base (`spatial_id` equals 0 and `temporal_id` equals 0) and enhancement layer (`spatial_id` greater than 0 or `temporal_id` greater than 0) data must include the OBU extension header.

OBUs with spatial level IDs (`spatial_id`) greater than 0 must appear within a temporal unit in increasing order of the spatial level ID values.

The first temporal unit of a coded video sequence must contain one or more sequence header OBUs before the first frame header OBU.

**Note:** There is not a requirement that every temporal unit with a key frame also contains a sequence header, just that the sequence header has been sent before the first key frame. However, note that temporal units without sequence header OBUs are not considered to be random access points.

Sequence header OBUs may appear in any order within a coded video sequence. Within a particular coded video sequence, the contents of `sequence_header_obu` must be bit-identical each time the sequence header appears except for the contents of `operating_parameters_info`. A new coded video sequence is required if the sequence header parameters change. Any sequence header in a bitstream which changes the parameters must be contained in a temporal unit with `temporal_id` equal to zero.

If a temporal unit contains one or more sequence header OBUs, the first appearance of the sequence header OBU must be before the first frame header OBU.

One or more metadata and padding OBUs may appear in any order within an OBU sequence (unless constrained by semantics provided elsewhere in this specification). Specific metadata types may be required or recommended to be placed in specific locations, as identified in their corresponding definitions.

OBU types that are not defined in this specification can be ignored by a decoder.

**Note:** Some applications may choose to use bitstreams that are not fully conformant to the requirements described in this section. For example, a bitstream received in a streaming use case may never contain key frames, but instead rely on gradual intra refresh.

## 7.6. Random access decoding

### 7.6.1. General

In general, random access points are places in a bitstream where decoding can be started.

This section defines the types of random access point that must be supported by all conformant decoders.

The purpose of this section is to define a minimum level of functionality that must be supported, not a maximum. In other words, decoders may choose to support more types of random access point.

The random access points are defined in [section 7.6.2](#).

The conformance requirements are specified in [section 7.6.3](#).

The consequences for encoders are specified in [section 7.6.4](#).

The consequences for decoders are specified in [section 7.6.5](#).

## 7.6.2. Definitions

This section defines the following terms:

- key frame random access point,
- delayed random access point,
- key frame dependent recovery point.

A key frame random access point is defined as being a frame:

- with `frame_type` equal to `KEY_FRAME`
- with `show_frame` equal to 1
- that is contained in a temporal unit that also contains a sequence header OBU

A delayed random access point is defined as being a frame:

- with `frame_type` equal to `KEY_FRAME`
- with `show_frame` equal to 0
- that is contained in a temporal unit that also contains a sequence header OBU

A key frame dependent recovery point is defined as being a frame:

- with `show_existing_frame` equal to 1
- with `frame_to_show_map_idx` specifying a frame to output that was a delayed random access point

## 7.6.3. Conformance requirements

Informally, the requirement for decoder conformance is that decoding can start at any key frame random access point or delayed random access point. The rest of this section makes this requirement more precise.

Starting at a key frame random access point is trivial, because if the earlier temporal units are dropped, the remaining temporal units still constitute a valid bitstream.

Starting at a delayed random access point is harder to define because:

- if all temporal units before the key frame dependent recovery point are dropped, it is impossible to decode (because the relevant delayed random access point has been dropped)
- if all temporal units before the delayed random access point are dropped, it is unclear what should happen for frames between the delayed random access point and the key-frame dependent recovery point (some applications may wish these to be dropped, while others may wish them to be displayed)
- in either case, the remaining temporal units do not constitute a valid standalone bitstream (because it does not start with a shown key frame)

To support the different modes of operation, a conformant decoder is required to be able to decode bitstreams consisting of:

- a temporal unit containing a delayed random access point
- immediately followed by a temporal unit containing the associated key frame dependent recovery point
- followed by optional additional temporal units

This moves the responsibility for dropping the intermediate temporal units (between the delayed random access point and the key frame dependent recovery point) out of the normatively defined decoding process into application specific behavior. This allows applications to choose which behavior to use depending on the use case and capabilities of the specific decoder implementation.

**Note:** In practice, decoder implementations are expected to be able to start decoding bitstreams from a delayed random access point when the intermediate temporal units are still present. The decoder should correctly produce all output frames from the next key frame or key frame dependent recovery point onwards, while the preceding frames are implementation defined. For example: a streaming decoder may choose to decode and display all frames even when the reference frames are not available (tolerating some errors in the output), a low latency decoder may choose to decode and display all frames that are guaranteed to be correct (e.g. an inter frame that only uses inter prediction from the delayed random access point), a media player decoder may choose to decode and display only frames starting from a key frame or key frame dependent recovery point (guaranteeing smooth playback once display starts).

## 7.6.4. Encoder consequences

Random access points introduce no additional conformance requirements on encoders.

Encoders are free to insert any number of random access points.

## 7.6.5. Decoder consequences

The conformance requirement means that conformant decoders must be able to start decoding at a delayed random access point partway through a valid bitstream.

This is almost the same as decoding a bitstream from the start - the only differences are that:

- The first frame has `show_frame` equal to 0.
- If `frame_id_numbers_present_flag` is equal to 1, for the first frame `current_frame_id` should not be compared to `PrevFrameID` (because `PrevFrameID` is uninitialized).

## 7.7. Frame end update CDF process

This process is triggered when the function `frame_end_update_cdf` is called from the tile group syntax table.

The frame CDF arrays are set equal to the saved CDF arrays as follows.

A copy is made of the saved CDF values for each of the CDF arrays mentioned in the semantics for `init_coeff_cdfs` and `init_non_coeff_cdfs`. The name of the destination for the copy is the name of the CDF array with no prefix. The name of the source for the copy is the name of the CDF array prefixed with “Saved”. For example, the array `YModeCdf` will be updated with values equal to the contents of `SavedYModeCdf`.

## 7.8. Set frame refs process

This process is triggered if the function `set_frame_refs` is called while reading the uncompressed header.

The syntax elements in the `ref_frame_idx` array are computed based on:

- the syntax elements `last_frame_idx` and `gold_frame_idx`,
- the values stored within the `RefOrderHint` array (these values represent the least significant bits of the expected output order of the frames).

The reference frames used for the `LAST_FRAME` and `GOLDEN_FRAME` references are sent explicitly and used to set the corresponding entries of `ref_frame_idx` as follows (the other entries are initialized to -1 and will be overwritten later in this process):

```
for ( i = 0; i < REFS_PER_FRAME; i++ )
    ref_frame_idx[ i ] = -1
ref_frame_idx[ LAST_FRAME - LAST_FRAME ] = last_frame_idx
ref_frame_idx[ GOLDEN_FRAME - LAST_FRAME ] = gold_frame_idx
```

An array `usedFrame` marking which reference frames have been used is prepared as follows:



```

for ( i = 0; i < NUM_REF_FRAMES; i++ )
    usedFrame[ i ] = 0
usedFrame[ last_frame_idx ] = 1
usedFrame[ gold_frame_idx ] = 1

```

A variable `curFrameHint` is set equal to  $1 \ll (\text{OrderHintBits} - 1)$ .

An array `shiftedOrderHints` (containing the expected output order shifted such that the current frame has hint equal to `curFrameHint`) is prepared as follows:

```

for ( i = 0; i < NUM_REF_FRAMES; i++ )
    shiftedOrderHints[ i ] = curFrameHint + get_relative_dist( RefOrderHint[ i ], OrderHint )

```

The variable `lastOrderHint` (representing the expected output order for `LAST_FRAME`) is set equal to `shiftedOrderHints[ last_frame_idx ]`.

It is a requirement of bitstream conformance that `lastOrderHint` is strictly less than `curFrameHint`.

The variable `goldOrderHint` (representing the expected output order for `GOLDEN_FRAME`) is set equal to `shiftedOrderHints[ gold_frame_idx ]`.

It is a requirement of bitstream conformance that `goldOrderHint` is strictly less than `curFrameHint`.

The `ALTREF_FRAME` reference is set to be a backward reference to the frame with highest output order as follows:

```

ref = find_latest_backward()
if ( ref >= 0 ) {
    ref_frame_idx[ ALTREF_FRAME - LAST_FRAME ] = ref
    usedFrame[ ref ] = 1
}

```

where `find_latest_backward` is defined as:

```

find_latest_backward() {
    ref = -1
    for ( i = 0; i < NUM_REF_FRAMES; i++ ) {
        hint = shiftedOrderHints[ i ]
        if ( !usedFrame[ i ] &&
            hint >= curFrameHint &&
            ( ref < 0 || hint >= latestOrderHint ) ) {
            ref = i
            latestOrderHint = hint
        }
    }
    return ref
}

```

The BWDREF\_FRAME reference is set to be a backward reference to the closest frame as follows:

```

ref = find_earliest_backward()
if ( ref >= 0 ) {
    ref_frame_idx[ BWDREF_FRAME - LAST_FRAME ] = ref
    usedFrame[ ref ] = 1
}

```

where find\_earliest\_backward is defined as:

```

find_earliest_backward() {
    ref = -1
    for ( i = 0; i < NUM_REF_FRAMES; i++ ) {
        hint = shiftedOrderHints[ i ]
        if ( !usedFrame[ i ] &&
            hint >= curFrameHint &&
            ( ref < 0 || hint < earliestOrderHint ) ) {
            ref = i
            earliestOrderHint = hint
        }
    }
    return ref
}

```

The ALTREF2\_FRAME reference is set to the next closest backward reference as follows:

```

ref = find_earliest_backward()
if ( ref >= 0 ) {
    ref_frame_idx[ ALTREF2_FRAME - LAST_FRAME ] = ref
    usedFrame[ ref ] = 1
}

```

The remaining references are set to be forward references in anti-chronological order as follows:

```

for ( i = 0; i < REFS_PER_FRAME - 2; i++ ) {
    refFrame = Ref_Frame_List[ i ]
    if ( ref_frame_idx[ refFrame - LAST_FRAME ] < 0 ) {
        ref = find_latest_forward()
        if ( ref >= 0 ) {
            ref_frame_idx[ refFrame - LAST_FRAME ] = ref
            usedFrame[ ref ] = 1
        }
    }
}

```

where Ref\_Frame\_List is specified as:

```

Ref_Frame_List[ REFS_PER_FRAME - 2 ] = {
    LAST2_FRAME, LAST3_FRAME, BWDREF_FRAME, ALTREF2_FRAME, ALTREF_FRAME
}

```

and find\_latest\_forward is defined as:

```

find_latest_forward() {
    ref = -1
    for ( i = 0; i < NUM_REF_FRAMES; i++ ) {
        hint = shiftedOrderHints[ i ]
        if ( !usedFrame[ i ] &&
            hint < curFrameHint &&
            ( ref < 0 || hint >= latestOrderHint ) ) {
            ref = i
            latestOrderHint = hint
        }
    }
    return ref
}

```

Finally, any remaining references are set to the reference frame with smallest output order as follows:

```

ref = -1
for ( i = 0; i < NUM_REF_FRAMES; i++ )
    hint = shiftedOrderHints[ i ]
    if ( ref < 0 || hint < earliestOrderHint ) {
        ref = i
        earliestOrderHint = hint
    }
}
for ( i = 0; i < REFS_PER_FRAME; i++ ) {
    if ( ref_frame_idx[ i ] < 0 ) {
        ref_frame_idx[ i ] = ref
    }
}

```

**Note:** Multiple reference frames can share the same value for OrderHint and care needs to be taken to handle this case consistently. The reference implementation uses an equivalent implementation based on sorting the reference frames based on their expected output order, with ties broken based on the reference frame index.

## 7.9. Motion field estimation process

### 7.9.1. General

This process is triggered by a call to `motion_field_estimation` while reading the uncompressed header.

A linear projection model is employed to create a motion field estimation that is able to capture high velocity temporal motion trajectories.

The motion field is estimated based on the saved motion vectors from the reference frames and the relative frame distances.

As the frame distances depend on the frame being referenced, a separate motion field is estimated for each reference frame used by the current frame.

A motion vector (for each reference frame type) is prepared at each location on an 8x8 luma sample grid.

The variable `w8` (representing the width of the motion field in units of 8x8 luma samples) is set equal to `MiCols >> 1`.

The variable `h8` (representing the height of the motion field in units of 8x8 luma samples) is set equal to `MiRows >> 1`.

As the linear projection can create a field with holes, the motion fields are initialized to an invalid motion vector of -32768, -32768 as follows:

```

for ( ref = LAST_FRAME; ref <= ALTREF_FRAME; ref++ )
  for ( y = 0; y < h8 ; y++ )
    for ( x = 0; x < w8; x++ )
      for ( j = 0; j < 2; j++ )
        MotionFieldMvs[ ref ][ y ][ x ][ j ] = -1 << 15

```

The variable `lastIdx` (representing which reference frame is used for `LAST_FRAME`) is set equal to `ref_frame_idx[ 0 ]`.

The variable `curGoldOrderHint` (representing the expected output order for `GOLDEN_FRAME` of the current frame) is set equal to `OrderHints[ GOLDEN_FRAME ]`.

The variable `lastAltOrderHint` (representing the expected output order for `ALTREF_FRAME` of `LAST_FRAME`) is set equal to `SavedOrderHints[ lastIdx ][ ALTREF_FRAME ]`.

The variable `useLast` (representing whether to project the motion vectors from `LAST_FRAME`) is set equal to ( `lastAltOrderHint != curGoldOrderHint` ).

If `useLast` is equal to 1, the projection process in [section 7.9.2](#) is invoked with `src` equal to `LAST_FRAME`, and `dstSign` equal to -1. (The output of this process is discarded.)

The variable `refStamp` (that limits how many reference frames have to be projected) is set equal to `MFMV_STACK_SIZE - 2`.

The variable `useBwd` is set equal to `get_relative_dist( OrderHints[ BWDREF_FRAME ], OrderHint ) > 0`.

If `useBwd` is equal to 1, the following steps apply:

- The projection process in [section 7.9.2](#) is invoked with `src` equal to `BWDREF_FRAME`, and `dstSign` equal to 1, and the output assigned to `projOutput`.
- If `projOutput` is equal to 1, `refStamp` is set equal to `refStamp - 1`.

The variable `useAlt2` is set equal to `get_relative_dist( OrderHints[ ALTREF2_FRAME ], OrderHint ) > 0`.

If `useAlt2` is equal to 1, the following steps apply:

- The projection process in [section 7.9.2](#) is invoked with `src` equal to `ALTREF2_FRAME`, and `dstSign` equal to 1, and the output assigned to `projOutput`.
- If `projOutput` is equal to 1, `refStamp` is set equal to `refStamp - 1`.

The variable `useAlt` is set equal to `get_relative_dist( OrderHints[ ALTREF_FRAME ], OrderHint ) > 0`.

If `useAlt` is equal to 1 and (`refStamp >= 0`), the following steps apply:

- The projection process in [section 7.9.2](#) is invoked with `src` equal to `ALTREF_FRAME`, and `dstSign` equal to 1, and the output assigned to `projOutput`.

- If projOutput is equal to 1, refStamp is set equal to refStamp - 1.

If ( refStamp  $\geq$  0 ), the projection process in [section 7.9.2](#) is invoked with src equal to LAST2\_FRAME, and dstSign equal to -1. (The output of this process is discarded.)

## 7.9.2. Projection process

The inputs to this process are:

- a variable src specifying which reference frame's motion vectors should be projected,
- a variable dstSign specifying a negation multiplier for the motion vector direction.

The process projects the motion vectors from a whole reference frame (which will always be one of the backward reference types) and stores the results in MotionFieldMvs.

The process outputs a single boolean value representing whether the source frame was valid for this operation. If the output is zero, no modification is made to MotionFieldMvs.

The variable srcIdx (representing which reference frame is used) is set equal to ref\_frame\_idx[ src - LAST\_FRAME ].

The variable w8 (representing the width of the motion field in units of 8x8 luma samples) is set equal to MiCols  $\gg$  1.

The variable h8 (representing the height of the motion field in units of 8x8 luma samples) is set equal to MiRows  $\gg$  1.

If RefMiRows[ srcIdx ] is not equal to MiRows, RefMiCols[ srcIdx ] is not equal to MiCols, or RefFrameType[ srcIdx ] is equal to INTRA\_ONLY\_FRAME or KEY\_FRAME, the process exits at this point, with the output set equal to 0.

The process is specified as follows:

```

for ( y8 = 0; y8 < h8; y8++ ) {
  for ( x8 = 0; x8 < w8; x8++ ) {
    row = 2 * y8 + 1
    col = 2 * x8 + 1
    srcRef = SavedRefFrames[ srcIdx ][ row ][ col ]
    if ( srcRef > INTRA_FRAME ) {
      refToCur = get_relative_dist( OrderHints[ src ], OrderHint )
      refOffset = get_relative_dist( OrderHints[ src ], SavedOrderHints[ srcIdx ][ srcRef ] )
      posValid = Abs( refToCur ) <= MAX_FRAME_DISTANCE &&
                  Abs( refOffset ) <= MAX_FRAME_DISTANCE &&
                  refOffset > 0
      if ( posValid ) {
        mv = SavedMvs[ srcIdx ][ row ][ col ]
        projMv = get_mv_projection( mv, refToCur * dstSign, refOffset )
        posValid = get_block_position( x8, y8, dstSign, projMv )
        if ( posValid ) {
          for ( dst = LAST_FRAME; dst <= ALTREF_FRAME; dst++ ) {
            refToDst = get_relative_dist( OrderHint, OrderHints[ dst ] )
            projMv = get_mv_projection( mv, refToDst, refOffset )
            MotionFieldMvs[ dst ][ PosY8 ][ PosX8 ] = projMv
          }
        }
      }
    }
  }
}

```

When the function `get_mv_projection` is called, the get mv projection process specified in [section 7.9.3](#) is invoked and the output assigned to `projMv`.

When the function `get_block_position` is called, the get block position process specified in [section 7.9.4](#) is invoked and the output assigned to `posValid`. This process also sets up the variables `PosY8` and `PosX8` representing the projected location in the motion field.

The process now exits with the output set equal to 1.

### 7.9.3. Get MV projection process

The inputs to this process are:

- a length 2 array `mv` specifying a motion vector,
- a variable numerator specifying the number of frames to be covered by the projected motion vector,
- a variable denominator specifying the number of frames covered by the original motion vector.

The outputs of this process are:

- a length 2 array `projMv` containing the projected motion vector

This process starts with a motion vector `mv` from a previous frame. This motion vector gives the displacement expected when moving a certain number of frames (given by the variable denominator). In order to use the motion vector for predictions using a different reference frame, the length of the motion vector must be scaled.

The variable `clippedDenominator` is set equal to `Min( MAX_FRAME_DISTANCE, denominator )`.

The variable `clippedNumerator` is set equal to `Clip3( -MAX_FRAME_DISTANCE, MAX_FRAME_DISTANCE, numerator )`.

The projected motion vector is specified as follows:

```
for ( i = 0; i < 2; i++ ) {
    scaled = Round2Signed( mv[ i ] * clippedNumerator * Div_Mult[ clippedDenominator ], 14 )
    projMv[ i ] = Clip3( -(1 << 14) + 1, (1 << 14) - 1, scaled )
}
```

where `Div_Mult` is a constant lookup table specified as:

```
Div_Mult[32] = {
    0,    16384, 8192, 5461, 4096, 3276, 2730, 2340, 2048, 1820, 1638,
    1489, 1365, 1260, 1170, 1092, 1024, 963, 910, 862, 819, 780,
    744, 712, 682, 655, 630, 606, 585, 564, 546, 528
}
```

## 7.9.4. Get block position process

The inputs to this process are:

- variables `x8` and `y8` specifying a location in units of 8x8 luma samples,
- a variable `dstSign` specifying a negation multiplier for the motion vector direction,
- a length 2 array `projMv` specifying a projected motion vector.

The process generates global variables `PosX8` and `PosY8` representing the projected location in units of 8x8 luma samples.

The process returns a flag `posValid` that indicates if the position should be used.

**Note:** `posValid` is specified such that only blocks within a certain distance of the current location need to be projected.

The variable `posValid` is set equal to 1.

The variable `PosY8` is set equal to `project(y8, projMv[ 0 ], dstSign, MiRows >> 1, MAX_OFFSET_HEIGHT)`.



The variable PosX8 is set equal to project(x8, projMv[ 1 ], dstSign, MiCols >> 1, MAX\_OFFSET\_WIDTH).

where the function project is specified as follows:

```
project( v8, delta, dstSign, max8, maxOff8 ) {
    base8 = (v8 >> 3) << 3
    if ( delta >= 0 ) {
        offset8 = delta >> ( 3 + 1 + MI_SIZE_LOG2 )
    } else {
        offset8 = -( ( -delta ) >> ( 3 + 1 + MI_SIZE_LOG2 ) )
    }
    v8 += dstSign * offset8
    if ( v8 < 0 ||
        v8 >= max8 ||
        v8 < base8 - maxOff8 ||
        v8 >= base8 + 8 + maxOff8 ) {
        posValid = 0
    }
    return v8
}
```

The project function clears posValid if the resulting position is offset too far.

## 7.10. Motion vector prediction processes

### 7.10.1. General

The following sections define the processes used for predicting the motion vectors.

The entry point to these processes is triggered by the function call to find\_mv\_stack in the inter block mode info syntax described in [section 5.11.23](#). This function call invokes the Find MV Stack Process specified in [section 7.10.2](#).

### 7.10.2. Find MV stack process

This process is triggered by a function call to find\_mv\_stack.

The input to this process is a variable isCompound containing 0 for single prediction, or 1 to signal compound prediction.

This process constructs an array RefStackMv containing motion vector candidates.

The process also prepares the value of the contexts used when decoding inter prediction syntax elements.

The array RefStackMv will be constructed during this process. RefStackMv[ idx ][ list ][ comp ] represents component comp (0 for y or 1 for x) of a motion vector for a particular list (0 or 1) at position idx (0 to MAX\_REF\_MV\_STACK\_SIZE - 1) in the stack. No initialization is needed because each entry is always written before it can be read.

The variable bw4 specifying the width of the block in 4x4 luma samples is set equal to Num\_4x4\_Blocks\_Wide[ MiSize ].

The variable `bh4` specifying the height of the block in 4x4 luma samples is set equal to `Num_4x4_Blocks_High[ MiSize ]`.

The following ordered steps apply:

1. The variable `NumMvFound` (representing the number of motion vector candidates in `RefStackMv`) is set equal to 0.
2. The variable `NewMvCount` (representing the number of candidates found that used NEWMV encoding) is set equal to 0.
3. The setup global mv process specified in [section 7.10.2.1](#) is invoked with the input 0 and the output is assigned to `GlobalMvs[ 0 ]`.
4. If `isCompound` is equal to 1, the setup global mv process specified in [section 7.10.2.1](#) is invoked with the input 1 and the output is assigned to `GlobalMvs[ 1 ]`.
5. The variable `FoundMatch` is set equal to 0.
6. The scan row process in [section 7.10.2.2](#) is invoked with `deltaRow` equal to -1 and `isCompound` as inputs.
7. The variable `foundAboveMatch` is set equal to `FoundMatch`, and `FoundMatch` is set equal to 0.
8. The scan col process in [section 7.10.2.3](#) is invoked with `deltaCol` equal to -1 and `isCompound` as inputs.
9. The variable `foundLeftMatch` is set equal to `FoundMatch`, and `FoundMatch` is set equal to 0.
10. If `Max( bw4, bh4 )` is less than or equal to 16, the scan point process in [section 7.10.2.4](#) is invoked with `deltaRow` equal to -1, `deltaCol` equal to `bw4`, and `isCompound` as inputs.
11. If `FoundMatch` is equal to 1, the variable `foundAboveMatch` is set equal to 1.
12. The variable `CloseMatches` (representing candidates found in the immediate neighborhood) is set equal to `foundAboveMatch + foundLeftMatch`.
13. The variable `numNearest` (representing the number of motion vectors found in the immediate neighborhood) is set equal to `NumMvFound`.
14. The variable `numNew` (representing the number of times a NEWMV candidate was found in the immediate neighborhood) is set equal to `NewMvCount`.
15. If `numNearest` is greater than 0, `WeightStack[ idx ]` is incremented by `REF_CAT_LEVEL` for `idx = 0..(numNearest-1)`.
16. The variable `ZeroMvContext` is set equal to 0.
17. If `use_ref_frame_mvs` is equal to 1, the temporal scan process in [section 7.10.2.5](#) is invoked with `isCompound` as input (the temporal scan process affects `ZeroMvContext`).

18. The scan point process in [section 7.10.2.4](#) is invoked with deltaRow equal to -1, deltaCol equal to -1, and isCompound as inputs.
19. If FoundMatch is equal to 1, the variable foundAboveMatch is set equal to 1.
20. The variable FoundMatch is set equal to 0.
21. The scan row process in [section 7.10.2.2](#) is invoked with deltaRow equal to -3 and isCompound as inputs.
22. If FoundMatch is equal to 1, the variable foundAboveMatch is set equal to 1.
23. The variable FoundMatch is set equal to 0.
24. The scan col process in [section 7.10.2.3](#) is invoked with deltaCol equal to -3 and isCompound as inputs.
25. If FoundMatch is equal to 1, the variable foundLeftMatch is set equal to 1.
26. The variable FoundMatch is set equal to 0.
27. If bh4 is greater than 1, the scan row process in [section 7.10.2.2](#) is invoked with deltaRow equal to -5 and isCompound as inputs.
28. If FoundMatch is equal to 1, the variable foundAboveMatch is set equal to 1.
29. The variable FoundMatch is set equal to 0.
30. If bw4 is greater than 1, the scan col process in [section 7.10.2.3](#) is invoked with deltaCol equal to -5 and isCompound as inputs.
31. If FoundMatch is equal to 1, the variable foundLeftMatch is set equal to 1.
32. The variable TotalMatches (representing all found candidates) is set equal to foundAboveMatch + foundLeftMatch.
33. The sorting process in [section 7.10.2.11](#) is invoked with start equal to 0, end equal to numNearest, and isCompound as input.
34. The sorting process in [section 7.10.2.11](#) is invoked with start equal to numNearest, end equal to NumMvFound, and isCompound as input.
35. If NumMvFound is less than 2, the extra search process in [section 7.10.2.12](#) is invoked with isCompound as input.
36. The context and clamping process in [section 7.10.2.14](#) is invoked with isCompound and numNew as input.

### 7.10.2.1. Setup global MV process

The input to this process is a variable refList specifying which set of motion vectors to predict.

The output is a motion vector mv representing global motion for this block.

The variable `ref` (specifying the reference frame) is set equal to `RefFrame[ refList ]`.

If `ref` is not equal to `INTRA_FRAME`, the variable `typ` (specifying the type of global motion) is set equal to `GmType[ ref ]`.

The variable `bw` (representing the width of the block in units of luma samples) is set equal to `Block_Width[ MiSize ]`.

The variable `bh` (representing the height of the block in units of luma samples) is set equal to `Block_Height[ MiSize ]`.

The output motion vector `mv` is specified by projecting the central luma sample of the block as follows:

```

if ( ref == INTRA_FRAME || typ == IDENTITY ) {
    mv[0] = 0
    mv[1] = 0
} else if ( typ == TRANSLATION ) {
    mv[0] = gm_params[ref][0] >> (WARPEDMODEL_PREC_BITS - 3)
    mv[1] = gm_params[ref][1] >> (WARPEDMODEL_PREC_BITS - 3)
} else {
    x = MiCol * MI_SIZE + bw / 2 - 1
    y = MiRow * MI_SIZE + bh / 2 - 1
    xc = (gm_params[ref][2] - (1 << WARPEDMODEL_PREC_BITS)) * x +
          gm_params[ref][3] * y +
          gm_params[ref][0]
    yc = gm_params[ref][4] * x +
          (gm_params[ref][5] - (1 << WARPEDMODEL_PREC_BITS)) * y +
          gm_params[ref][1]
    if ( allow_high_precision_mv ) {
        mv[0] = Round2Signed(yc, WARPEDMODEL_PREC_BITS - 3)
        mv[1] = Round2Signed(xc, WARPEDMODEL_PREC_BITS - 3)
    } else {
        mv[0] = Round2Signed(yc, WARPEDMODEL_PREC_BITS - 2) * 2
        mv[1] = Round2Signed(xc, WARPEDMODEL_PREC_BITS - 2) * 2
    }
}
lower_mv_precision( mv )

```

where the call to `lower_mv_precision` invokes the lower precision process specified in [section 7.10.2.10](#).

## 7.10.2.2. Scan row process

The inputs to this process are:

- a variable `deltaRow` specifying (in units of 4x4 luma samples) how far above to look for motion vectors,
- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction.

The variable `bw4` specifying the width of the block in 4x4 luma samples is set equal to `Num_4x4_Blocks_Wide[ MiSize ]`.

The variable `end4` specifying the last block to be scanned in horizontal 4x4 luma samples is set equal to `Min( Min( bw4, MiCols - MiCol ), 16 )`.

**Note:** end4 limits the number of locations to be searched for large blocks. There is a similar optimization in that the scan point process for the top-right location is not invoked for large blocks. For example, for a 64 by 64 block, candidates from the row above will be examined at x offsets of -1, 0, 16, 32, 48, 64. (The 0, 16, 32, 48 locations are scanned in this process, while the -1 and 64 are scanned by the scan point process.) However, for a 128 by 64 or 64 by 128 block, candidates from the row above will only be examined at x offsets of -1, 0, 16, 32, 48 because the scan point process for the top-right location is not invoked.

The variable deltaCol is set equal to 0.

The variable useStep16 is set equal to (bw4 >= 16).

**Note:** useStep16 is equal to 1 when the block is 64 luma samples wide or wider. This means only 4 locations will be searched in this case. However, a 32 luma samples wide block may still search 8 locations.

If Abs(deltaRow) is greater than 1, the offset is adjusted as follows:

```
deltaRow += MiRow & 1
deltaCol = 1 - (MiCol & 1)
```

**Note:** These adjustments reduce the number of motion vectors that need to be kept in memory.

A series of motion vector locations is scanned as follows:

```
i = 0
while ( i < end4 ) {
    mvRow = MiRow + deltaRow
    mvCol = MiCol + deltaCol + i
    if ( !is_inside(mvRow,mvCol) )
        break
    len = Min(bw4, Num_4x4_Blocks_Wide[ MiSizes[ mvRow ][ mvCol ] ])
    if ( Abs(deltaRow) > 1 )
        len = Max(2, len)
    if ( useStep16 )
        len = Max(4, len)
    weight = len * 2
    add_ref_mv_candidate( mvRow, mvCol, isCompound, weight)
    i += len
}
```

where the call to add\_ref\_mv\_candidate invokes the process in [section 7.10.2.7](#).

### 7.10.2.3. Scan col process

The inputs to this process are:

- a variable deltaCol specifying (in units of 4x4 luma samples) how far left to look for motion vectors,
- a variable isCompound containing 0 for single prediction, or 1 to signal compound prediction.

The variable bh4 specifying the height of the block in 4x4 luma samples is set equal to Num\_4x4\_Blocks\_High[ MiSize ].

The variable end4 specifying the last block to be scanned in vertical 4x4 luma samples is set equal to Min( Min( bh4, MiRows - MiRow ), 16 ).

The variable deltaRow is set equal to 0.

The variable useStep16 is set equal to (bh4 >= 16).

If Abs(deltaCol) is greater than 1, the offset is adjusted as follows:

```
deltaRow = 1 - (MiRow & 1)
deltaCol += MiCol & 1
```

A series of motion vector locations is scanned as follows:

```
i = 0
while ( i < end4 ) {
    mvRow = MiRow + deltaRow + i
    mvCol = MiCol + deltaCol
    if ( !is_inside(mvRow,mvCol) )
        break
    len = Min(bh4, Num_4x4_Blocks_High[ MiSizes[ mvRow ][ mvCol ] ])
    if ( Abs(deltaCol) > 1 )
        len = Max(2, len)
    if ( useStep16 )
        len = Max(4, len)
    weight = len * 2
    add_ref_mv_candidate( mvRow, mvCol, isCompound, weight )
    i += len
}
```

where the call to add\_ref\_mv\_candidate invokes the process in [section 7.10.2.7](#).

### 7.10.2.4. Scan point process

The inputs to this process are:

- a variable deltaRow specifying (in units of 4x4 luma samples) how far above to look for a motion vector,

- a variable `deltaCol` specifying (in units of 4x4 luma samples) how far left to look for a motion vector,
- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction.

The variable `mvRow` is set equal to `MiRow + deltaRow`.

The variable `mvCol` is set equal to `MiCol + deltaCol`.

The variable `weight` is set equal to 4.

If `is_inside( mvRow, mvCol )` is equal to 1 and `RefFrames[ mvRow ][ mvCol ][ 0 ]` has been written for this frame (this checks that the candidate location has been decoded), the add reference motion vector process in [section 7.10.2.7](#) is invoked with `mvRow`, `mvCol`, `isCompound`, `weight` as inputs.

### 7.10.2.5. Temporal scan process

The input to this process is a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction.

This process scans the motion vectors in a previous frame looking for candidates which use the same reference frame.

The variable `bw4` specifying the width of the block in 4x4 luma samples is set equal to `Num_4x4_Blocks_Wide[ MiSize ]`.

The variable `bh4` specifying the height of the block in 4x4 luma samples is set equal to `Num_4x4_Blocks_High[ MiSize ]`.

The variable `stepW4` is set equal to  $(bw4 \geq 16) ? 4 : 2$ .

The variable `stepH4` is set equal to  $(bh4 \geq 16) ? 4 : 2$ .

The process scans the locations within the block as follows:

```
for ( deltaRow = 0; deltaRow < Min( bh4, 16 ) ; deltaRow += stepH4 ) {
    for ( deltaCol = 0; deltaCol < Min( bw4, 16 ) ; deltaCol += stepW4 ) {
        add_tpl_ref_mv( deltaRow, deltaCol, isCompound )
    }
}
```

where the call to `add_tpl_ref_mv` invokes the temporal sample process in [section 7.10.2.6](#).

The process then scans positions around the block (but still within the same superblock) as follows:

```

allowExtension = ((bh4 >= Num_4x4_Blocks_High[BLOCK_8X8]) &&
                  (bh4 < Num_4x4_Blocks_High[BLOCK_64X64]) &&
                  (bw4 >= Num_4x4_Blocks_Wide[BLOCK_8X8]) &&
                  (bw4 < Num_4x4_Blocks_Wide[BLOCK_64X64]))
if ( allowExtension ) {
    for ( i = 0; i < 3; i++ ) {
        deltaRow = tplSamplePos[ i ][ 0 ]
        deltaCol = tplSamplePos[ i ][ 1 ]
        if ( check_sb_border( deltaRow, deltaCol ) ) {
            add_tpl_ref_mv( deltaRow, deltaCol, isCompound)
        }
    }
}

```

where `tplSamplePos` contains the offsets to search (in units of 4x4 luma samples) and is specified as:

```

tplSamplePos[3][2] = {
    { bh4, -2 }, { bh4, bw4 }, { bh4 - 2, bw4 }
}

```

and `check_sb_border` checks that the position is within the same 64x64 block as follows:

```

check_sb_border( deltaRow, deltaCol ) {
    row = (MiRow & 15) + deltaRow
    col = (MiCol & 15) + deltaCol

    return ( row >= 0 && row < 16 && col >= 0 && col < 16 )
}

```

### 7.10.2.6. Temporal sample process

The inputs to this process are:

- variables `deltaRow` and `deltaCol` specifying (in units of 4x4 luma samples) the offset to the candidate location,
- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction.

This process looks up a motion vector from the motion field and adds it into the stack.

The variable `mvRow` is set equal to  $(\text{MiRow} + \text{deltaRow}) \mid 1$ .

The variable `mvCol` is set equal to  $(\text{MiCol} + \text{deltaCol}) \mid 1$ .

If `is_inside( mvRow, mvCol )` is equal to 0, this process terminates immediately.

The variable `x8` is set equal to  $\text{mvCol} \gg 1$ .



The variable  $y8$  is set equal to  $mvRow \gg 1$ .

( $x8$  and  $y8$  represent the position of the candidate in units of  $8 \times 8$  luma samples.)

The process is specified as follows:

```

if ( deltaRow == 0 && deltaCol == 0 ) {
    ZeroMvContext = 1
}
if ( !isCompound ) {
    candMv = MotionFieldMvs[ RefFrame[ 0 ] ][ y8 ][ x8 ]
    if ( candMv[ 0 ] == -1 << 15 )
        return
    lower_mv_precision( candMv )
    if ( deltaRow == 0 && deltaCol == 0 ) {
        if ( Abs( candMv[ 0 ] - GlobalMvs[ 0 ][ 0 ] ) >= 16 ||
            Abs( candMv[ 1 ] - GlobalMvs[ 0 ][ 1 ] ) >= 16 )
            ZeroMvContext = 1
        else
            ZeroMvContext = 0
    }
    for ( idx = 0; idx < NumMvFound; idx++ ) {
        if ( candMv[ 0 ] == RefStackMv[ idx ][ 0 ][ 0 ] &&
            candMv[ 1 ] == RefStackMv[ idx ][ 0 ][ 1 ] )
            break
    }
    if ( idx < NumMvFound ) {
        WeightStack[ idx ] += 2
    } else if ( NumMvFound < MAX_REF_MV_STACK_SIZE ) {
        RefStackMv[ NumMvFound ][ 0 ] = candMv
        WeightStack[ NumMvFound ] = 2
        NumMvFound += 1
    }
} else {
    candMv0 = MotionFieldMvs[ RefFrame[ 0 ] ][ y8 ][ x8 ]
    if ( candMv0[ 0 ] == -1 << 15 )
        return
    candMv1 = MotionFieldMvs[ RefFrame[ 1 ] ][ y8 ][ x8 ]
    if ( candMv1[ 0 ] == -1 << 15 )
        return
    lower_mv_precision( candMv0 )
    lower_mv_precision( candMv1 )
    if ( deltaRow == 0 && deltaCol == 0 ) {
        if ( Abs( candMv0[ 0 ] - GlobalMvs[ 0 ][ 0 ] ) >= 16 ||
            Abs( candMv0[ 1 ] - GlobalMvs[ 0 ][ 1 ] ) >= 16 ||
            Abs( candMv1[ 0 ] - GlobalMvs[ 1 ][ 0 ] ) >= 16 ||
            Abs( candMv1[ 1 ] - GlobalMvs[ 1 ][ 1 ] ) >= 16 )
            ZeroMvContext = 1
        else
            ZeroMvContext = 0
    }
    for ( idx = 0; idx < NumMvFound; idx++ ) {
        if ( candMv0[ 0 ] == RefStackMv[ idx ][ 0 ][ 0 ] &&
            candMv0[ 1 ] == RefStackMv[ idx ][ 0 ][ 1 ] &&
            candMv1[ 0 ] == RefStackMv[ idx ][ 1 ][ 0 ] &&
            candMv1[ 1 ] == RefStackMv[ idx ][ 1 ][ 1 ] )

```

```

        break
    }
    if ( idx < NumMvFound ) {
        WeightStack[ idx ] += 2
    } else if ( NumMvFound < MAX_REF_MV_STACK_SIZE ) {
        RefStackMv[ NumMvFound ][ 0 ] = candMv0
        RefStackMv[ NumMvFound ][ 1 ] = candMv1
        WeightStack[ NumMvFound ] = 2
        NumMvFound += 1
    }
}

```

where the call to `lower_mv_precision` invokes the lower precision process specified in [section 7.10.2.10](#).

### 7.10.2.7. Add reference motion vector process

The inputs to this process are:

- variables `mvRow` and `mvCol` specifying (in units of 4x4 luma samples) the candidate location,
- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction,
- a variable `weight` specifying the weight attached to this motion vector.

This process examines the candidate to find matching reference frames.

If `IsInters[ mvRow ][ mvCol ]` is equal to 0, this process terminates immediately.

If `isCompound` is equal to 0, the following applies for `candList = 0..1`:

1. If `RefFrames[ mvRow ][ mvCol ][ candList ]` is equal to `RefFrame[ 0 ]`, the search stack process in [section 7.10.2.8](#) is invoked with `mvRow`, `mvCol`, `weight`, and `candList` as inputs.

Otherwise (`isCompound` is equal to 1), the following applies:

1. If `RefFrames[ mvRow ][ mvCol ][ 0 ]` is equal to `RefFrame[ 0 ]` and `RefFrames[ mvRow ][ mvCol ][ 1 ]` is equal to `RefFrame[ 1 ]`, the compound search stack process in [section 7.10.2.9](#) is invoked with `mvRow`, `mvCol`, and `weight` as inputs.

### 7.10.2.8. Search stack process

The inputs to this process are:

- variables `mvRow` and `mvCol` specifying (in units of 4x4 luma samples) the candidate location,
- a variable `candList` specifying which list in the candidate matches our reference frame,
- a variable `weight` proportional to the corresponding block width or height for the candidate motion vector.

This process searches the stack for an exact match with a candidate motion vector. If present, the weight of the candidate motion vector is added to the weight of its counterpart in the stack, otherwise the process adds a motion vector to the stack.

The variable `candMode` is set equal to `YModes[ mvRow ][ mvCol ]`.

The variable `candSize` is set equal to `MiSizes[ mvRow ][ mvCol ]`.

The variable `large` is set equal to  $( \text{Min}( \text{Block\_Width}[ \text{candSize} ], \text{Block\_Height}[ \text{candSize} ] ) \geq 8 )$ .

The candidate motion vector `candMv` is set as follows:

- If  $( \text{candMode} == \text{GLOBALMV} \parallel \text{candMode} == \text{GLOBAL\_GLOBALMV} )$  and  $( \text{GmType}[ \text{RefFrame}[ 0 ] ] > \text{TRANSLATION} )$  and  $( \text{large} == 1 )$ , `candMv` is set equal to `GlobalMvs[ 0 ]`.
- Otherwise, `candMv` is set equal to `Mvs[ mvRow ][ mvCol ][ candList ]`.

The lower precision process specified in [section 7.10.2.10](#) is invoked with `candMv`.

If `has_newmv( candMode )` is equal to 1, `NewMvCount` is set equal to `NewMvCount + 1`.

The variable `FoundMatch` is set equal to 1.

The process depends on whether the candidate motion vector is already in the stack as follows:

- If `candMv` is already equal to `RefStackMv[ idx ][ 0 ]` for some `idx` less than `NumMvFound`, then `WeightStack[ idx ]` is increased by `weight`
- Otherwise, if `NumMvFound` is less than `MAX_REF_MV_STACK_SIZE`, the following ordered steps apply:
  - a. `RefStackMv[ NumMvFound ][ 0 ]` is set equal to `candMv`
  - b. `WeightStack[ NumMvFound ]` is set equal to `weight`
  - c. `NumMvFound` is set equal to `NumMvFound + 1`.
- Otherwise, (`NumMvFound` is greater than or equal to `MAX_REF_MV_STACK_SIZE`), the process has no effect.

### 7.10.2.9. Compound search stack process

The inputs to this process are:

- variables `mvRow` and `mvCol` specifying (in units of 4x4 luma samples) the candidate location,
- a variable `weight` proportional to the corresponding block width or height for the candidate pair of motion vectors.

This process searches the stack for an exact match with a candidate pair of motion vectors. If present, the weight of the candidate pair of motion vectors is added to the weight of its counterpart in the stack, otherwise the process adds the motion vectors to the stack.

The array `candMvs` (containing two motion vectors) is set equal to `Mvs[ mvRow ][ mvCol ]`.

The variable `candMode` is set equal to `YModes[ mvRow ][ mvCol ]`.

The variable `candSize` is set equal to `MiSizes[ mvRow ][ mvCol ]`.

If `candMode` is equal to `GLOBAL_GLOBALMV`, for `refList = 0..1` the following applies:

- If `GmType[ RefFrame[ refList ] ] > TRANSLATION`, `candMvs[ refList ]` is set equal to `GlobalMvs[ refList ]`.

For `i = 0..1`, the lower precision process specified in [section 7.10.2.10](#) is invoked with `candMvs[ i ]`.

The variable `FoundMatch` is set equal to 1.

The process depends on whether the candidate motion vector pair is already in the stack as follows:

- If `candMvs[ 0 ]` is equal to `RefStackMv[ idx ][ 0 ]` and `candMvs[ 1 ]` is equal to `RefStackMv[ idx ][ 1 ]` for some `idx` less than `NumMvFound`, then `WeightStack[ idx ]` is increased by `weight`
- Otherwise, if `NumMvFound` is less than `MAX_REF_MV_STACK_SIZE`, the following ordered steps apply:
  - a. `RefStackMv[ NumMvFound ][ i ]` is set equal to `candMvs[ i ]` for `i = 0..1`
  - b. `WeightStack[ NumMvFound ]` is set equal to `weight`
  - c. `NumMvFound` is set equal to `NumMvFound + 1`.
- Otherwise, (`NumMvFound` is greater than or equal to `MAX_REF_MV_STACK_SIZE`), the process has no effect.

If `has_newmv( candMode )` is equal to 1, `NewMvCount` is set equal to `NewMvCount + 1`.

The function `has_newmv` is defined as:

```
has_newmv( mode ) {
    return (mode == NEWMV ||
           mode == NEW_NEWMV ||
           mode == NEAR_NEWMV ||
           mode == NEW_NEARMV ||
           mode == NEAREST_NEWMV ||
           mode == NEW_NEARESTMV)
}
```

**Note:** It is impossible for `mode` to equal `NEWMV` in this function because it is only called for compound modes.

## 7.10.2.10. Lower precision process

The input to this process is a reference `candMv` to a motion vector array.

This process modifies the contents of the input motion vector to remove the least significant bit when high precision is not allowed, and all three fractional bits when force\_integer\_mv is equal to 1.

If allow\_high\_precision\_mv is equal to 1, this process terminates immediately.

For i = 0..1, the following applies:

```

if ( force_integer_mv ) {
    a = Abs( candMv[ i ] )
    aInt = (a + 3) >> 3
    if ( candMv[ i ] > 0 )
        candMv[ i ] = aInt << 3
    else
        candMv[ i ] = -( aInt << 3 )
} else {
    if ( candMv[ i ] & 1 ) {
        if ( candMv[ i ] > 0 )
            candMv[ i ]--
        else
            candMv[ i ]++
    }
}

```

### 7.10.2.11. Sorting process

The inputs to this process are:

- a variable start representing the first position to be sorted,
- a variable end representing the length of the array,
- a variable isCompound containing 0 for single prediction, or 1 to signal compound prediction.

This process performs a stable sort of part of the stack of motion vectors according to the corresponding weight.

Entries in RefStackMv from start (inclusive) to end (exclusive) are sorted.

The sorting process is specified as:

```

while ( end > start ) {
    newEnd = start
    for ( idx = start + 1; idx < end; idx++ ) {
        if ( WeightStack[ idx - 1 ] < WeightStack[ idx ] ) {
            swap_stack(idx - 1, idx)
            newEnd = idx
        }
    }
    end = newEnd
}

```

When the function `swap_stack` is invoked, the entries at locations `idx` and `idx - 1` should be swapped in `WeightStack` and `RefStackMv` as follows:

```

swap_stack( i, j ) {
    temp = WeightStack[ i ]
    WeightStack[ i ] = WeightStack[ j ]
    WeightStack[ j ] = temp
    for ( list = 0; list < 1 + isCompound; list++ ) {
        for ( comp = 0; comp < 2; comp++ ) {
            temp = RefStackMv[ i ][ list ][ comp ]
            RefStackMv[ i ][ list ][ comp ] = RefStackMv[ j ][ list ][ comp ]
            RefStackMv[ j ][ list ][ comp ] = temp
        }
    }
}

```

### 7.10.2.12. Extra search process

The input to this process is a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction.

This process adds additional motion vectors to `RefStackMv` until it has 2 choices of motion vector by first searching the left and above neighbors for partially matching candidates, and second adding global motion candidates.

When doing single prediction, the motion vectors go straight on the stack.

When doing compound prediction, the motion vectors are added to arrays called `RefIdMvs` (counting matches from the same frame) and `RefDiffMvs` (counting matches from different frames).

The number of entries in these arrays are initialized to zero as follows:

```

for ( list = 0; list < 2; list++ ) {
    RefIdCount[ list ] = 0
    RefDiffCount[ list ] = 0
}

```

A two pass search for the partial matching candidates is specified as:

```

w4 = Min( 16, Num_4x4_Blocks_Wide[ MiSize ] )
h4 = Min( 16, Num_4x4_Blocks_High[ MiSize ] )
w4 = Min( w4, MiCols - MiCol )
h4 = Min( h4, MiRows - MiRow )
num4x4 = Min( w4, h4 )
for ( pass = 0; pass < 2; pass++ ) {
    idx = 0
    while ( idx < num4x4 && NumMvFound < 2 ) {
        if ( pass == 0 ) {
            mvRow = MiRow - 1
            mvCol = MiCol + idx
        } else {
            mvRow = MiRow + idx
            mvCol = MiCol - 1
        }
        if ( !is_inside( mvRow, mvCol ) )
            break
        add_extra_mv_candidate( mvRow, mvCol, isCompound )
        if ( pass == 0 ) {
            idx += Num_4x4_Blocks_Wide[ MiSizes[ mvRow ][ mvCol ] ]
        } else {
            idx += Num_4x4_Blocks_High[ MiSizes[ mvRow ][ mvCol ] ]
        }
    }
}

```

The first pass searches the row above, the second searches the column to the left.

The function call to `add_extra_mv_candidate` invokes the add extra mv candidate process specified in [section 7.10.2.13](#) with `mvRow`, `mvCol`, `isCompound` as inputs.

If `isCompound` is equal to 1, the candidates in the `RefIdMvs` and `RefDiffMvs` arrays are added to the stack as follows (using the temporary array `combinedMvs`):



```

for ( list = 0; list < 2; list++ ) {
    compCount = 0
    for ( idx = 0; idx < RefIdCount[ list ]; idx++ ) {
        combinedMvs[ compCount ][ list ] = RefIdMvs[ list ][ idx ]
        compCount++
    }
    for ( idx = 0; idx < RefDiffCount[ list ] && compCount < 2; idx++ ) {
        combinedMvs[ compCount ][ list ] = RefDiffMvs[ list ][ idx ]
        compCount++
    }
    while ( compCount < 2 ) {
        combinedMvs[ compCount ][ list ] = GlobalMvs[ list ]
        compCount++
    }
}
if ( NumMvFound == 1 ) {
    if ( combinedMvs[ 0 ][ 0 ] == RefStackMv[ 0 ][ 0 ] &&
        combinedMvs[ 0 ][ 1 ] == RefStackMv[ 0 ][ 1 ] ) {
        RefStackMv[ NumMvFound ][ 0 ] = combinedMvs[ 1 ][ 0 ]
        RefStackMv[ NumMvFound ][ 1 ] = combinedMvs[ 1 ][ 1 ]
    } else {
        RefStackMv[ NumMvFound ][ 0 ] = combinedMvs[ 0 ][ 0 ]
        RefStackMv[ NumMvFound ][ 1 ] = combinedMvs[ 0 ][ 1 ]
    }
    WeightStack[ NumMvFound ] = 2
    NumMvFound++
} else {
    for ( idx = 0; idx < 2; idx++ ) {
        RefStackMv[ NumMvFound ][ 0 ] = combinedMvs[ idx ][ 0 ]
        RefStackMv[ NumMvFound ][ 1 ] = combinedMvs[ idx ][ 1 ]
        WeightStack[ NumMvFound ] = 2
        NumMvFound++
    }
}
}

```

If isCompound is equal to 0, the candidates have already been added to RefStackMv, and this process simply extends with global motion candidates as follows:

```

for ( idx = NumMvFound; idx < 2; idx++ ) {
    RefStackMv[ idx ][ 0 ] = GlobalMvs[ 0 ]
}

```

**Note:** For single prediction, NumMvFound is not incremented by the addition of global motion candidates, whereas for compound prediction NumMvFound will always be greater or equal to 2 by this point.

### 7.10.2.13. Add extra MV candidate process

The inputs to this process are:

- variables mvRow and mvCol specifying (in units of 4x4 luma samples) the candidate location,
- a variable isCompound containing 0 for single prediction, or 1 to signal compound prediction.

This process may modify the contents of the global variables RefIdMvs, RefIdCount, RefDiffMvs, RefDiffCount, RefStackMv, WeightStack, and NumMvFound.

This process examines the candidate location to find possible motion vectors as follows:

```

if ( isCompound ) {
  for ( candList = 0; candList < 2; candList++ ) {
    candRef = RefFrames[ mvRow ][ mvCol ][ candList ]
    if ( candRef > INTRA_FRAME ) {
      for ( list = 0; list < 2; list++ ) {
        candMv = Mvs[ mvRow ][ mvCol ][ candList ]
        if ( candRef == RefFrame[ list ] && RefIdCount[ list ] < 2 ) {
          RefIdMvs[ list ][ RefIdCount[ list ] ] = candMv
          RefIdCount[ list ]++
        } else if ( RefDiffCount[ list ] < 2 ) {
          if ( RefFrameSignBias[ candRef ] != RefFrameSignBias[ RefFrame[ list ] ] ) {
            candMv[ 0 ] *= -1
            candMv[ 1 ] *= -1
          }
          RefDiffMvs[ list ][ RefDiffCount[ list ] ] = candMv
          RefDiffCount[ list ]++
        }
      }
    }
  }
} else {
  for ( candList = 0; candList < 2; candList++ ) {
    candRef = RefFrames[ mvRow ][ mvCol ][ candList ]
    if ( candRef > INTRA_FRAME ) {
      candMv = Mvs[ mvRow ][ mvCol ][ candList ]
      if ( RefFrameSignBias[ candRef ] != RefFrameSignBias[ RefFrame[ 0 ] ] ) {
        candMv[ 0 ] *= -1
        candMv[ 1 ] *= -1
      }
      for ( idx = 0; idx < NumMvFound; idx++ ) {
        if ( candMv == RefStackMv[ idx ][ 0 ] )
          break
      }
      if ( idx == NumMvFound ) {
        RefStackMv[ idx ][ 0 ] = candMv
        WeightStack[ idx ] = 2
        NumMvFound++
      }
    }
  }
}
}

```

#### 7.10.2.14. Context and clamping process

The inputs to this process are:

- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction,
- a variable `numNew` specifying the number of NEWMV candidates found in the immediate neighborhood.

This process computes contexts to be used when decoding syntax elements, and clamps the candidates in RefStackMv.

The variable bw (representing the width of the block in units of luma samples) is set equal to Block\_Width[ MiSize ].

The variable bh (representing the height of the block in units of luma samples) is set equal to Block\_Height[ MiSize ].

**Note:** It only matters whether numNew is zero or non-zero because the value is clipped at 1 when it is used. Implementations may therefore choose to implement numNew and NewMvCount as a boolean instead of a counter.

The variable numLists specifying the number of reference frames used for this block is set equal to ( isCompound ? 2 : 1 ).

The array DrlCtxStack is set as follows:

```
for ( idx = 0; idx < NumMvFound ; idx++ ) {
    z = 0
    if ( idx + 1 < NumMvFound ) {
        w0 = WeightStack[ idx ]
        w1 = WeightStack[ idx + 1 ]
        if ( w0 >= REF_CAT_LEVEL ) {
            if ( w1 < REF_CAT_LEVEL ) {
                z = 1
            }
        } else {
            z = 2
        }
    }
    DrlCtxStack[ idx ] = z
}
```

The motion vectors are clamped as follows:

```
for ( list = 0; list < numLists; list++ ) {
    for ( idx = 0; idx < NumMvFound ; idx++ ) {
        refMv = RefStackMv[ idx ][ list ]
        refMv[ 0 ] = clamp_mv_row( refMv[ 0 ], MV_BORDER + bh * 8)
        refMv[ 1 ] = clamp_mv_col( refMv[ 1 ], MV_BORDER + bw * 8)
        RefStackMv[ idx ][ list ] = refMv
    }
}
```

The variables RefMvContext and NewMvContext are set as follows:

```

if ( CloseMatches == 0 ) {
    NewMvContext = Min( TotalMatches, 1 )      // 0,1
    RefMvContext = TotalMatches
} else if ( CloseMatches == 1 ) {
    NewMvContext = 3 - Min( numNew, 1 )        // 2,3
    RefMvContext = 2 + TotalMatches
} else {
    NewMvContext = 5 - Min( numNew, 1 )        // 4,5
    RefMvContext = 5
}

```

### 7.10.3. Has overlappable candidates process

This process is triggered by a call to `has_overlappable_candidates`.

It returns 1 to indicate that the block has neighbors suitable for use by overlapped motion compensation, or 0 otherwise.

The process looks to see if there are any inter blocks to the left or above.

The check is only made at 8x8 granularity.

The process is specified as:

```

has_overlappable_candidates( ) {
    if ( AvailU ) {
        w4 = Num_4x4_Blocks_Wide[ MiSize ]
        for ( x4 = MiCol; x4 < Min( MiCols, MiCol + w4 ); x4 += 2 ) {
            if ( RefFrames[ MiRow - 1 ][ x4 | 1 ][ 0 ] > INTRA_FRAME )
                return 1
        }
    }
    if ( AvailL ) {
        h4 = Num_4x4_Blocks_High[ MiSize ]
        for ( y4 = MiRow; y4 < Min( MiRows, MiRow + h4 ); y4 += 2 ) {
            if ( RefFrames[ y4 | 1 ][ MiCol - 1 ][ 0 ] > INTRA_FRAME )
                return 1
        }
    }
    return 0
}

```

### 7.10.4. Find warp samples process

#### 7.10.4.1. General

This process is triggered when the `find_warp_samples` function is invoked.

The process examines the neighboring inter predicted blocks and estimates a local warp transformation based on the motion vectors.

The process produces a variable NumSamples containing the number of valid candidates found, and an array CandList containing sorted candidates.

The variables NumSamples and NumSamplesScanned are both set equal to 0.

**Note:** NumSamplesScanned counts the number of distinct candidates found by the add sample process - even if the motion vectors are too large. NumSamples counts the number of distinct valid candidates found by the add sample process (i.e. only counting cases where the motion vector is small enough to be considered valid). As a special case, if no small motion vectors are found, then the process returns the first large motion vector found (by setting NumSamples to 1).

The variable w4 specifying the width of the block in 4x4 luma samples is set equal to Num\_4x4\_Blocks\_Wide[ MiSize ].

The variable h4 specifying the height of the block in 4x4 luma samples is set equal to Num\_4x4\_Blocks\_High[ MiSize ].

The process is specified as:

```

doTopLeft = 1
doTopRight = 1
if ( AvailU ) {
    srcSize = MiSizes[ MiRow - 1 ][ MiCol ]
    srcW = Num_4x4_Blocks_Wide[ srcSize ]
    if ( w4 <= srcW ) {
        colOffset = -(MiCol & (srcW - 1))
        if ( colOffset < 0 )
            doTopLeft = 0
        if ( colOffset + srcW > w4 )
            doTopRight = 0
        add_sample( -1, 0 )
    } else {
        for ( i = 0; i < Min( w4, MiCols - MiCol ); i += miStep ) {
            srcSize = MiSizes[ MiRow - 1 ][ MiCol + i ]
            srcW = Num_4x4_Blocks_Wide[ srcSize ]
            miStep = Min(w4, srcW)
            add_sample( -1, i )
        }
    }
}
if ( AvailL ) {
    srcSize = MiSizes[ MiRow ][ MiCol - 1 ]
    srcH = Num_4x4_Blocks_High[ srcSize ]
    if ( h4 <= srcH ) {
        rowOffset = -(MiRow & (srcH - 1))
        if ( rowOffset < 0 )
            doTopLeft = 0
        add_sample( 0, -1 )
    } else {
        for ( i = 0; i < Min( h4, MiRows - MiRow ); i += miStep ) {
            srcSize = MiSizes[ MiRow + i ][ MiCol - 1 ]
            srcH = Num_4x4_Blocks_High[ srcSize ]
            miStep = Min(h4, srcH)
            add_sample( i, -1 )
        }
    }
}
if ( doTopLeft ) {
    add_sample( -1, -1 )
}
if ( doTopRight ) {
    if ( Max( w4, h4 ) <= 16 ) {
        add_sample( -1, w4 )
    }
}
if ( NumSamples == 0 && NumSamplesScanned > 0 )
    NumSamples = 1

```

where the call to `add_sample` specifies that the add sample process in [section 7.10.4.2](#) should be invoked.

## 7.10.4.2. Add sample process

The inputs to this process are:

- a variable `deltaRow` specifying (in units of 4x4 luma samples) how far above to look for a motion vector,
- a variable `deltaCol` specifying (in units of 4x4 luma samples) how far left to look for a motion vector.

The output of this process is to add a new sample to the list of candidates if it is a valid candidate and has not been seen before.

If `NumSamplesScanned` is greater than or equal to `LEAST_SQUARES_SAMPLES_MAX`, this process immediately exits.

The variable `mvRow` is set equal to `MiRow + deltaRow`.

The variable `mvCol` is set equal to `MiCol + deltaCol`.

If `is_inside( mvRow, mvCol )` is equal to 0, then this process immediately returns.

If `RefFrames[ mvRow ][ mvCol ][ 0 ]` has not been written for this frame, then this process immediately returns.

If `RefFrames[ mvRow ][ mvCol ][ 0 ]` is not equal to `RefFrame[ 0 ]`, then this process immediately returns.

If `RefFrames[ mvRow ][ mvCol ][ 1 ]` is not equal to `NONE`, then this process immediately returns.

The variable `candSz` is set equal to `MiSizes[ mvRow ][ mvCol ]`.

The variable `candW4` is set equal to `Num_4x4_Blocks_Wide[ candSz ]`.

The variable `candH4` is set equal to `Num_4x4_Blocks_High[ candSz ]`.

The variable `candRow` is set equal to `mvRow & ~(candH4 - 1)`.

The variable `candCol` is set equal to `mvCol & ~(candW4 - 1)`.

The variable `midY` is set equal to `candRow * 4 + candH4 * 2 - 1`.

The variable `midX` is set equal to `candCol * 4 + candW4 * 2 - 1`.

The variable `threshold` is set equal to `Clip3( 16, 112, Max( Block_Width[ MiSize ], Block_Height[ MiSize ] ) )`.

The variable `mvDiffRow` is set equal to `Abs( Mvs[ candRow ][ candCol ][ 0 ][ 0 ] - Mv[ 0 ][ 0 ] )`.

The variable `mvDiffCol` is set equal to `Abs( Mvs[ candRow ][ candCol ][ 0 ][ 1 ] - Mv[ 0 ][ 1 ] )`.

The variable `valid` is set equal to `( ( mvDiffRow + mvDiffCol ) <= threshold )`.

**Note:** `candRow` and `candCol` give the top-left position of the candidate block in units of 4x4 blocks. `midX` and `midY` give the central position of the candidate block in units of luma samples.



A candidate array (representing source and destination locations in units of 1/8 luma samples) is specified as:

```
cand[ 0 ] = midY * 8
cand[ 1 ] = midX * 8
cand[ 2 ] = midY * 8 + Mvs[ candRow ][ candCol ][ 0 ][ 0 ]
cand[ 3 ] = midX * 8 + Mvs[ candRow ][ candCol ][ 0 ][ 1 ]
```

The following ordered steps apply:

1. NumSamplesScanned is increased by 1.
2. If valid is equal to 0 and NumSamplesScanned is greater than 1, the process exits.
3. CandList[ NumSamples ][ j ] is set equal to cand[ j ] for j=0..3.
4. If valid is equal to 1, NumSamples is increased by 1.

## 7.11. Prediction processes

### 7.11.1. General

The following sections define the processes used for predicting the sample values.

These processes are triggered at points defined by function calls to predict\_intra, predict\_inter, predict\_chroma\_from\_luma, and predict\_palette in the residual syntax table described in [section 5.11.34](#).

### 7.11.2. Intra prediction process

#### 7.11.2.1. General

The intra prediction process is invoked for intra coded blocks to predict a part of the block corresponding to a transform block. When the transform size is smaller than the block size, this process can be invoked multiple times within a single block for the same plane, and the invocations are in raster order within the block.

This process is triggered by a call to predict\_intra.

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the current transform block,
- a variable haveLeft that is equal to 1 if there are valid samples to the left of this transform block,
- a variable haveAbove that is equal to 1 if there are valid samples above this transform block,

- a variable `haveAboveRight` that is equal to 1 if there are valid samples above the transform block to the right of this transform block,
- a variable `haveBelowLeft` that is equal to 1 if there are valid samples to the left of the transform block below this transform block,
- a variable `mode` specifying the type of intra prediction to apply,
- a variable `log2W` specifying the base 2 logarithm of the width of the region to be predicted,
- a variable `log2H` specifying the base 2 logarithm of the height of the region to be predicted.

The process makes use of the already reconstructed samples in the current frame `CurrFrame` to form a prediction for the current block.

The outputs of this process are intra predicted samples in the current frame `CurrFrame`.

The variable `w` is set equal to  $1 \ll \log2W$ .

The variable `h` is set equal to  $1 \ll \log2H$ .

The variable `maxX` is set equal to  $(\text{MiCols} * \text{MI\_SIZE}) - 1$ .

The variable `maxY` is set equal to  $(\text{MiRows} * \text{MI\_SIZE}) - 1$ .

If `plane` is greater than 0, then:

- `maxX` is set equal to  $(\text{MiCols} * \text{MI\_SIZE}) \gg \text{subsampling\_x} - 1$ .
- `maxY` is set equal to  $(\text{MiRows} * \text{MI\_SIZE}) \gg \text{subsampling\_y} - 1$ .

The array `AboveRow[i]` for  $i = 0..w + h - 1$  is derived as follows:

- If `haveAbove` is equal to 0 and `haveLeft` is equal to 1, `AboveRow[i]` is set equal to `CurrFrame[plane][y][x - 1]`.
- Otherwise, if `haveAbove` is equal to 0 and `haveLeft` is equal to 0, `AboveRow[i]` is set equal to  $(1 \ll (\text{BitDepth} - 1)) - 1$ .
- Otherwise, the following applies:
  - The variable `aboveLimit` is set equal to  $\text{Min}(\text{maxX}, x + (\text{haveAboveRight} ? 2 * w : w) - 1)$ .
  - `AboveRow[i]` is set equal to `CurrFrame[plane][y-1][Min(aboveLimit, x+i)]`.

The array `LeftCol[i]` for  $i = 0..w + h - 1$  is derived as follows:

- If `haveLeft` is equal to 0 and `haveAbove` is equal to 1, `LeftCol[i]` is set equal to `CurrFrame[plane][y - 1][x]`.

- Otherwise, if haveLeft is equal to 0 and haveAbove is equal to 0, LeftCol[ i ] is set equal to  $(1 \ll (\text{BitDepth} - 1)) + 1$ .
- Otherwise, the following applies:
  - The variable leftLimit is set equal to  $\text{Min}(\text{maxY}, y + (\text{haveBelowLeft} ? 2 * h : h) - 1)$ .
  - LeftCol[ i ] is set equal to CurrFrame[ plane ][ Min(leftLimit, y+i) ][ x-1 ].

The array AboveRow[ i ] for i = -1 is specified by:

- If haveAbove is equal to 1 and haveLeft is equal to 1, AboveRow[ -1 ] is set equal to CurrFrame[ plane ][ y-1 ][ x-1 ].
- Otherwise if haveAbove is equal to 1, AboveRow[ -1 ] is set equal to CurrFrame [ plane ][ y - 1 ][ x ].
- Otherwise if haveLeft is equal to 1, AboveRow[ -1 ] is set equal to CurrFrame [ plane ][ y ][ x - 1 ].
- Otherwise, AboveRow[ -1 ] is set equal to  $1 \ll (\text{BitDepth} - 1)$ .

The array LeftCol[ i ] for i = -1 is set equal to AboveRow[ -1 ].

A 2D array named pred containing the intra predicted samples is constructed as follows:

- If plane is equal to 0 and use\_filter\_intra is true, the recursive intra prediction process specified in [section 7.11.2.3](#) is invoked with w and h as inputs, and the output is assigned to pred.
- Otherwise, if is\_directional\_mode( mode ) is true, the directional intra prediction process specified in [section 7.11.2.4](#) is invoked with plane, x, y, haveLeft, haveAbove, mode, w, h, maxX, maxY as inputs and the output is assigned to pred.
- Otherwise if mode is equal to SMOOTH\_PRED or SMOOTH\_V\_PRED or SMOOTH\_H\_PRED, the smooth intra prediction process specified in [section 7.11.2.6](#) is invoked with mode, log2W, log2H, w, and h as inputs, and the output is assigned to pred.
- Otherwise if mode is equal to DC\_PRED, the DC intra prediction process specified in [section 7.11.2.5](#) is invoked with haveLeft, haveAbove, log2W, log2H, w, and h as inputs and the output is assigned to pred.
- Otherwise (mode is equal to PAETH\_PRED), the basic intra prediction process specified in [section 7.11.2.2](#) is invoked with mode, w, and h as inputs, and the output is assigned to pred.

The current frame is updated as follows:

- CurrFrame[ plane ][ y + i ][ x + j ] is set equal to pred[ i ][ j ] for i = 0..h-1 and j = 0..w-1.

## 7.11.2.2. Basic intra prediction process

The inputs to this process are:

- a variable  $w$  specifying the width of the region to be predicted,
- a variable  $h$  specifying the height of the region to be predicted.

The output of this process is a 2D array named `pred` containing the intra predicted samples.

The process generates filtered samples from the samples in `LeftCol` and `AboveRow` as follows:

- The following ordered steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
  1. The variable `base` is set equal to `AboveRow[ j ] + LeftCol[ i ] - AboveRow[ -1 ]`.
  2. The variable `pLeft` is set equal to `Abs( base - LeftCol[ i ] )`.
  3. The variable `pTop` is set equal to `Abs( base - AboveRow[ j ] )`.
  4. The variable `pTopLeft` is set equal to `Abs( base - AboveRow[ -1 ] )`.
  5. If `pLeft <= pTop` and `pLeft <= pTopLeft`, `pred[ i ][ j ]` is set equal to `LeftCol[ i ]`.
  6. Otherwise, if `pTop <= pTopLeft`, `pred[ i ][ j ]` is set equal to `AboveRow[ j ]`.
  7. Otherwise, `pred[ i ][ j ]` is set equal to `AboveRow[ -1 ]`.

The output of the process is the array `pred`.

### 7.11.2.3. Recursive intra prediction process

The inputs to this process are:

- a variable  $w$  specifying the width of the region to be predicted,
- a variable  $h$  specifying the height of the region to be predicted.

The output of this process is a 2D array named `pred` containing the intra predicted samples.

For each block of  $4 \times 2$  samples, this process first prepares an array `p` of 7 neighboring samples, and then produces the output block by filtering this array.

The variable `w4` is set equal to  $w \gg 2$ .

The variable `h2` is set equal to  $h \gg 1$ .

The following steps apply for  $i2 = 0..h2-1$ , for  $j4 = 0..w4-1$ :

- The array `p` is derived as follows for  $i = 0..6$ :
  - If  $i$  is less than 5, `p[ i ]` is derived as follows:
    - If  $i2$  is equal to 0, `p[ i ]` is set equal to `AboveRow[ ( j4 << 2 ) + i - 1 ]`.

- Otherwise, if  $j_4$  is equal to 0 and  $i$  is equal to 0,  $p[i]$  is set equal to  $\text{LeftCol}[(i_2 \ll 1) - 1]$ .
- Otherwise,  $p[i]$  is set equal to  $\text{pred}[(i_2 \ll 1) - 1][(j_4 \ll 2) + i - 1]$ .
- Otherwise ( $i$  is greater than or equal to 5),  $p[i]$  is derived as follows:
  - If  $j_4$  is equal to 0,  $p[i]$  is set equal to  $\text{LeftCol}[(i_2 \ll 1) + i - 5]$ .
  - Otherwise ( $j_4$  is not equal to 0),  $p[i]$  is set equal to  $\text{pred}[(i_2 \ll 1) + i - 5][(j_4 \ll 2) - 1]$ .
- The following steps apply for  $i_1 = 0..1$ ,  $j_1 = 0..3$ :
  - The variable  $pr$  is set equal to 0.
  - The variable  $pr$  is incremented by  $\text{Intra\_Filter\_Taps}[\text{filter\_intra\_mode}][(i_1 \ll 2) + j_1][i] * p[i]$  for  $i = 0..6$ .
  - $\text{pred}[(i_2 \ll 1) + i_1][(j_4 \ll 2) + j_1]$  is set equal to  $\text{Clip1}(\text{Round2Signed}(pr, \text{INTRA\_FILTER\_SCALE\_BITS}))$ .

The output of the process is the array  $\text{pred}$ .

#### 7.11.2.4. Directional intra prediction process

The inputs to this process are:

- a variable  $\text{plane}$  specifying which plane is being predicted,
- variables  $x$  and  $y$  specifying the location of the top left sample in the  $\text{CurrFrame}[\text{plane}]$  array of the current transform block,
- a variable  $\text{haveLeft}$  that is equal to 1 if there are valid samples to the left of this transform block,
- a variable  $\text{haveAbove}$  that is equal to 1 if there are valid samples above this transform block,
- a variable  $\text{mode}$  specifying the type of intra prediction to apply,
- a variable  $w$  specifying the width of the region to be predicted,
- a variable  $h$  specifying the height of the region to be predicted,
- a variable  $\text{maxX}$  specifying the largest valid  $x$  coordinate for the current plane,
- a variable  $\text{maxY}$  specifying the largest valid  $y$  coordinate for the current plane.

The output of this process is a 2D array named  $\text{pred}$  containing the intra predicted samples.

The process uses a directional filter to generate filtered samples from the samples in  $\text{LeftCol}$  and  $\text{AboveRow}$ .

The following ordered steps apply:

1. The variable angleDelta is derived as follows:
  - If plane is equal to 0, angleDelta is set equal to AngleDeltaY.
  - Otherwise (plane is not equal to 0), angleDelta is set equal to AngleDeltaUV.
2. The variable pAngle is set equal to ( Mode\_To\_Angle[ mode ] + angleDelta \* ANGLE\_STEP ).
3. The variables upsampleAbove and upsampleLeft are set equal to 0.
4. If enable\_intra\_edge\_filter is equal to 1, the following applies:
  - If pAngle is not equal to 90 and pAngle is not equal to 180, the following applies:
    - If (pAngle > 90) and (pAngle < 180) and (w + h) >= 24, the filter corner process specified in [section 7.11.2.7](#) is invoked and the output assigned to both LeftCol[ -1 ] and AboveRow[ -1 ].
    - The intra filter type process specified in [section 7.11.2.8](#) is invoked with the input variable plane and the output assigned to filterType.
    - If haveAbove is equal to 1, the following steps apply:
      - The intra edge filter strength selection process specified in [section 7.11.2.9](#) is invoked with w, h, filterType, and pAngle - 90 as inputs, and the output assigned to the variable strength.
      - The variable numPx is set equal to  $\text{Min}(w, (\text{maxX} - x + 1)) + (\text{pAngle} < 90 ? h : 0) + 1$ .
      - The intra edge filter process specified in [section 7.11.2.12](#) is invoked with the parameters numPx, strength, and 0 as inputs.
    - If haveLeft is equal to 1, the following steps apply:
      - The intra edge filter strength selection process specified in [section 7.11.2.9](#) is invoked with w, h, filterType, and pAngle - 180 as inputs, and the output assigned to the variable strength.
      - The variable numPx is set equal to  $\text{Min}(h, (\text{maxY} - y + 1)) + (\text{pAngle} > 180 ? w : 0) + 1$ .
      - The intra edge filter process specified in [section 7.11.2.12](#) is invoked with the parameters numPx, strength, and 1 as inputs.
  - The intra edge upsample selection process specified in [section 7.11.2.10](#) is invoked with w, h, filterType, and pAngle - 90 as inputs, and the output assigned to the variable upsampleAbove.
  - The variable numPx is set equal to  $(w + (\text{pAngle} < 90 ? h : 0))$ .

- If upsampleAbove is equal to 1, the intra edge upsample process specified in [section 7.11.2.11](#) is invoked with the parameters numPx and 0 as inputs.
- The intra edge upsample selection process specified in [section 7.11.2.10](#) is invoked with w, h, filterType, and pAngle - 180 as inputs, and the output assigned to the variable upsampleLeft.
- The variable numPx is set equal to  $(h + (pAngle < 180 ? w : 0))$ .
- If upsampleLeft is equal to 1, the intra edge upsample process specified in [section 7.11.2.11](#) is invoked with the parameters numPx and 1 as inputs.

5. The variable dx is derived as follows:

- If pAngle is less than 90, dx is set equal to Dr\_Intra\_Derivative[ pAngle ].
- Otherwise, if pAngle is greater than 90 and less than 180, dx is set equal to Dr\_Intra\_Derivative[ 180 - pAngle ].
- Otherwise, dx is undefined.

6. The variable dy is derived as follows:

- If pAngle is greater than 90 and less than 180, dy is set equal to Dr\_Intra\_Derivative[ pAngle - 90 ].
- Otherwise, if pAngle is greater than 180, dy is set equal to Dr\_Intra\_Derivative[ 270 - pAngle ].
- Otherwise, dy is undefined.

7. If pAngle is less than 90, the following steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :

- The variable idx is set equal to  $(i + 1) * dx$ .
- The variable base is set equal to  $(idx \gg (6 - upsampleAbove)) + (j \ll upsampleAbove)$ .
- The variable shift is set equal to  $((idx \ll upsampleAbove) \gg 1) \& 0x1F$ .
- The variable maxBaseX is set equal to  $(w + h - 1) \ll upsampleAbove$ .
- If base is less than maxBaseX,  $pred[i][j]$  is set equal to  $Clip1( Round2( AboveRow[ base ] * (32 - shift) + AboveRow[ base + 1 ] * shift, 5 ))$ .
- Otherwise (base is greater than or equal to maxBaseX),  $pred[i][j]$  is set equal to AboveRow[ maxBaseX ].

8. Otherwise, if pAngle is greater than 90 and pAngle is less than 180, the following steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :

- The variable idx is set equal to  $(j \ll 6) - (i + 1) * dx$ .

- The variable base is set equal to  $\text{idx} \gg (6 - \text{upsampleAbove})$ .
  - If base is greater than or equal to  $-(1 \ll \text{upsampleAbove})$ , the following steps apply:
    - The variable shift is set equal to  $((\text{idx} \ll \text{upsampleAbove}) \gg 1) \& 0x1F$ .
    - $\text{pred}[i][j]$  is set equal to  $\text{Clip1}(\text{Round2}(\text{AboveRow}[\text{base}] * (32 - \text{shift}) + \text{AboveRow}[\text{base} + 1] * \text{shift}, 5))$ .
  - Otherwise (base is less than  $-(1 \ll \text{upsampleAbove})$ ), the following steps apply:
    - The variable idx is set equal to  $(i \ll 6) - (j + 1) * \text{dy}$ .
    - The variable base is set equal to  $\text{idx} \gg (6 - \text{upsampleLeft})$ .
    - The variable shift is set equal to  $((\text{idx} \ll \text{upsampleLeft}) \gg 1) \& 0x1F$ .
    - $\text{pred}[i][j]$  is set equal to  $\text{Clip1}(\text{Round2}(\text{LeftCol}[\text{base}] * (32 - \text{shift}) + \text{LeftCol}[\text{base} + 1] * \text{shift}, 5))$ .
9. Otherwise, if pAngle is greater than 180, the following steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
- The variable idx is set equal to  $(j + 1) * \text{dy}$ .
  - The variable base is set equal to  $(\text{idx} \gg (6 - \text{upsampleLeft})) + (i \ll \text{upsampleLeft})$ .
  - The variable shift is set equal to  $((\text{idx} \ll \text{upsampleLeft}) \gg 1) \& 0x1F$ .
  - $\text{pred}[i][j]$  is set equal to  $\text{Clip1}(\text{Round2}(\text{LeftCol}[\text{base}] * (32 - \text{shift}) + \text{LeftCol}[\text{base} + 1] * \text{shift}, 5))$ .
10. Otherwise, if pAngle is equal to 90,  $\text{pred}[i][j]$  is set equal to  $\text{AboveRow}[j]$  with  $j = 0..w-1$  and  $i = 0..h-1$  (each row of the block is filled with a copy of AboveRow).
11. Otherwise, if pAngle is equal to 180,  $\text{pred}[i][j]$  is set equal to  $\text{LeftCol}[i]$  with  $j = 0..w-1$  and  $i = 0..h-1$  (each column of the block is filled with a copy of LeftCol).

The output of the process is the array pred.

### 7.11.2.5. DC intra prediction process

The inputs to this process are:

- a variable haveLeft that is equal to 1 if there are valid samples to the left of this transform block,
- a variable haveAbove that is equal to 1 if there are valid samples above this transform block,
- a variable log2W specifying the base 2 logarithm of the width of the region to be predicted,
- a variable log2H specifying the base 2 logarithm of the height of the region to be predicted,



- a variable *w* specifying the width of the region to be predicted,
- a variable *h* specifying the height of the region to be predicted.

The output of this process is a 2D array named *pred* containing the intra predicted samples.

The process averages the available edge samples in *LeftCol* and *AboveRow* to generate the prediction as follows:

- If *haveLeft* is equal to 1 and *haveAbove* is equal to 1, *pred[i][j]* is set equal to *avg* with *i* = 0..*h*-1 and *j* = 0..*w*-1. The variable *avg* (the average of the samples in union of *AboveRow* and *LeftCol*) is specified as follows:

```
sum = 0
for ( k = 0; k < h; k++ )
    sum += LeftCol[ k ]
for ( k = 0; k < w; k++ )
    sum += AboveRow[ k ]

sum += ( w + h ) >> 1
avg = sum / ( w + h )
```

**Note:** The reference code shows how the division by (*w*+*h*) can be implemented with multiplication and shift operations.

- Otherwise if *haveLeft* is equal to 1 and *haveAbove* is equal to 0, *pred[i][j]* is set equal to *leftAvg* with *i* = 0..*h*-1 and *j* = 0..*w*-1. The variable *leftAvg* is specified as follows:

```
sum = 0
for ( k = 0; k < h; k++ ) {
    sum += LeftCol[ k ]
}
leftAvg = Clip1( ( sum + ( h >> 1 ) ) >> log2H )
```

- Otherwise if *haveLeft* is equal to 0 and *haveAbove* is equal to 1, *pred[i][j]* is set equal to *aboveAvg* with *i* = 0..*h*-1 and *j* = 0..*w*-1. The variable *aboveAvg* is specified as follows:

```
sum = 0
for ( k = 0; k < w; k++ ) {
    sum += AboveRow[ k ]
}
aboveAvg = Clip1( ( sum + ( w >> 1 ) ) >> log2W )
```

- Otherwise (*haveLeft* is equal to 0 and *haveAbove* is equal to 0), *pred[i][j]* is set equal to  $1 \ll (\text{BitDepth} - 1)$  with *i* = 0..*h*-1 and *j* = 0..*w*-1.

The output of the process is the array `pred`.

### 7.11.2.6. Smooth intra prediction process

The inputs to this process are:

- a variable `mode` specifying the type of intra prediction to apply,
- a variable `log2W` specifying the base 2 logarithm of the width of the region to be predicted,
- a variable `log2H` specifying the base 2 logarithm of the height of the region to be predicted,
- a variable `w` specifying the width of the region to be predicted,
- a variable `h` specifying the height of the region to be predicted.

The output of this process is a 2D array named `pred` containing the intra predicted samples.

The process uses linear interpolation to generate filtered samples from the samples in `LeftCol` and `AboveRow` as follows:

- If `mode` is equal to `SMOOTH_PRED`, the following ordered steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
  1. The array `smWeightsX` is set dependent on the value of `log2W` according to the following table:

<b>log2W</b>	<b>smWeightsX</b>
2	<code>Sm_Weights_Tx_4x4</code>
3	<code>Sm_Weights_Tx_8x8</code>
4	<code>Sm_Weights_Tx_16x16</code>
5	<code>Sm_Weights_Tx_32x32</code>
6	<code>Sm_Weights_Tx_64x64</code>

2. The array `smWeightsY` is set dependent on the value of `log2H` according to the following table:

<b>log2H</b>	<b>smWeightsY</b>
2	<code>Sm_Weights_Tx_4x4</code>
3	<code>Sm_Weights_Tx_8x8</code>
4	<code>Sm_Weights_Tx_16x16</code>
5	<code>Sm_Weights_Tx_32x32</code>
6	<code>Sm_Weights_Tx_64x64</code>

3. The variable `smoothPred` is set as follows:

```

smoothPred =  smWeightsY[ i ] * AboveRow[ j ] +
               ( 256 - smWeightsY[ i ] ) * LeftCol[ h - 1 ] +
               smWeightsX[ j ] * LeftCol[ i ] +
               ( 256 - smWeightsX[ j ] ) * AboveRow[ w - 1 ]

```

4.  $\text{pred}[i][j]$  is set equal to  $\text{Round2}(\text{smoothPred}, 9)$ .
- Otherwise if mode is equal to SMOOTH\_V\_PRED, the following ordered steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
    1. The array  $\text{smWeights}$  is set dependent on the value of  $\log_2 H$  according to the following table:

<b>log2H</b>	<b>smWeights</b>
2	Sm_Weights_Tx_4x4
3	Sm_Weights_Tx_8x8
4	Sm_Weights_Tx_16x16
5	Sm_Weights_Tx_32x32
6	Sm_Weights_Tx_64x64

2. The variable  $\text{smoothPred}$  is set as follows:

```

smoothPred =  smWeights[ i ] * AboveRow[ j ] +
               ( 256 - smWeights[ i ] ) * LeftCol[ h - 1 ]

```

3.  $\text{pred}[i][j]$  is set equal to  $\text{Round2}(\text{smoothPred}, 8)$ .
- Otherwise (mode is equal to SMOOTH\_H\_PRED), the following ordered steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
    1. The array  $\text{smWeights}$  is set dependent on the value of  $\log_2 W$  according to the following table:

<b>log2W</b>	<b>smWeights</b>
2	Sm_Weights_Tx_4x4
3	Sm_Weights_Tx_8x8
4	Sm_Weights_Tx_16x16
5	Sm_Weights_Tx_32x32
6	Sm_Weights_Tx_64x64

2. The variable  $\text{smoothPred}$  is set as follows:

```
smoothPred = smWeights[ j ] * LeftCol[ i ] +
              ( 256 - smWeights[ j ] ) * AboveRow[ w - 1 ]
```

3. `pred[ i ][ j ]` is set equal to `Round2( smoothPred, 8 )`.

The output of the process is the array `pred`.

### 7.11.2.7. Filter corner process

This process uses a three tap filter to compute the value to be used for the top-left corner.

The variable `s` is set equal to `LeftCol[ 0 ] * 5 + AboveRow[ -1 ] * 6 + AboveRow[ 0 ] * 5`.

The output of this process is `Round2(s, 4)`.

### 7.11.2.8. Intra filter type process

The input to this process is a variable `plane` specifying the color plane being processed.

The output of this process is a variable `filterType` that is set to 1 if either the block above or to the left uses a smooth prediction mode.

The process is specified as follows:

```

get_filter_type( plane ) {
    aboveSmooth = 0
    leftSmooth = 0
    if ( ( plane == 0 ) ? AvailU : AvailUChroma ) {
        r = MiRow - 1
        c = MiCol
        if ( plane > 0 ) {
            if ( subsampling_x && !( MiCol & 1 ) )
                c++
            if ( subsampling_y && ( MiRow & 1 ) )
                r--
        }
        aboveSmooth = is_smooth( r, c, plane )
    }
    if ( ( plane == 0 ) ? AvailL : AvailLChroma ) {
        r = MiRow
        c = MiCol - 1
        if ( plane > 0 ) {
            if ( subsampling_x && ( MiCol & 1 ) )
                c--
            if ( subsampling_y && !( MiRow & 1 ) )
                r++
        }
        leftSmooth = is_smooth( r, c, plane )
    }
    return aboveSmooth || leftSmooth
}

```

where the function `is_smooth` indicates if a prediction mode is one of the smooth intra modes and is specified as:

```

is_smooth( row, col, plane ) {
    if ( plane == 0 ) {
        mode = YModes[ row ][ col ]
    } else {
        if ( RefFrames[ row ][ col ][ 0 ] > INTRA_FRAME )
            return 0
        mode = UVModes[ row ][ col ]
    }
    return (mode == SMOOTH_PRED || mode == SMOOTH_V_PRED || mode == SMOOTH_H_PRED)
}

```

### 7.11.2.9. Intra edge filter strength selection process

The inputs to this process are:

- a variable `w` containing the width of the transform in samples,
- a variable `h` containing the height of the transform in samples,

- a variable `filterType` that is 0 or 1 that controls the strength of filtering,
- a variable `delta` containing an angle difference in degrees.

The output is an intra edge filter strength from 0 to 3 inclusive.

The variable `d` is set equal to `Abs( delta )`.

The variable `blkWh` (containing the sum of the dimensions) is set equal to `w + h`.

The output variable `strength` is specified as follows:

```

strength = 0
if ( filterType == 0 ) {
    if ( blkWh <= 8 ) {
        if ( d >= 56 ) strength = 1
    } else if ( blkWh <= 12 ) {
        if ( d >= 40 ) strength = 1
    } else if ( blkWh <= 16 ) {
        if ( d >= 40 ) strength = 1
    } else if ( blkWh <= 24 ) {
        if ( d >= 8 ) strength = 1
        if ( d >= 16 ) strength = 2
        if ( d >= 32 ) strength = 3
    } else if ( blkWh <= 32 ) {
        strength = 1
        if ( d >= 4 ) strength = 2
        if ( d >= 32 ) strength = 3
    } else {
        strength = 3
    }
} else {
    if ( blkWh <= 8 ) {
        if ( d >= 40 ) strength = 1
        if ( d >= 64 ) strength = 2
    } else if ( blkWh <= 16 ) {
        if ( d >= 20 ) strength = 1
        if ( d >= 48 ) strength = 2
    } else if ( blkWh <= 24 ) {
        if ( d >= 4 ) strength = 3
    } else {
        strength = 3
    }
}

```

### 7.11.2.10. Intra edge upsample selection process

The inputs to this process are:

- a variable `w` containing the width of the transform in samples,

- a variable `h` containing the height of the transform in samples,
- a variable `filterType` that is 0 or 1 that controls the strength of filtering,
- a variable `delta` containing an angle difference in degrees.

The output is a flag `useUpsample` that is true if upsampling should be applied to the edge.

The variable `d` is set equal to `Abs( delta )`.

The variable `blkWh` (containing the sum of the dimensions) is set equal to `w + h`.

The output variable `useUpsample` is specified as follows:

```
if ( d <= 0 || d >= 40 ) {
    useUpsample = 0
} else if ( filterType == 0 ) {
    useUpsample = (blkWh <= 16)
} else {
    useUpsample = (blkWh <= 8)
}
```

### 7.11.2.11. Intra edge upsample process

The inputs to this process are:

- a variable `numPx` specifying the number of samples to filter,
- a variable `dir` containing 0 when filtering the above samples, and 1 when filtering the left samples.

The output of this process are upsampled samples in the `AboveRow` and `LeftCol` arrays.

The variable `buf` is set depending on `dir`:

- If `dir` is equal to 0, `buf` is set equal to a reference to `AboveRow`.
- Otherwise (`dir` is equal to 1), `buf` is set equal to a reference to `LeftCol`.

**Note:** `buf` is a reference to either `AboveRow` or `LeftCol`. “reference” indicates that modifying values in `buf` modifies values in the original array.

When the process starts, entries -1 to `numPx-1` are valid in `buf` and contain the original values. When the process completes, entries -2 to `2*numPx-2` are valid in `buf` and contain the upsampled values.

An array `dup` of length `numPx+3` is generated by extending `buf` by one sample at the start and end as follows:

```

dup[ 0 ] = buf[ -1 ]
for ( i = -1; i < numPx; i++ ) {
    dup[ i + 2 ] = buf[ i ]
}
dup[ numPx + 2 ] = buf[ numPx - 1 ]

```

The upsampling process (modifying values in buf) is specified as follows:

```

buf[-2] = dup[0]
for ( i = 0; i < numPx; i++ ) {
    s = -dup[i] + (9 * dup[i + 1]) + (9 * dup[i + 2]) - dup[i + 3]
    s = Clip1( Round2(s, 4) )
    buf[ 2 * i - 1 ] = s
    buf[ 2 * i ] = dup[i + 2]
}

```

## 7.11.2.12. Intra edge filter process

The inputs to this process are:

- a size sz (sz will always be less than or equal to 129),
- a filter strength strength between 0 and 3 inclusive,
- an edge direction left (when equal to 1, it specifies a vertical edge; when equal to 0, it specifies a horizontal edge).

The process filters the LeftCol (if left is equal to 1) or AboveRow (if left is equal to 0) arrays.

If strength is equal to 0, the process returns without doing anything.

The array edge is derived by setting edge[ i ] equal to ( left ? LeftCol[ i - 1 ] : AboveRow[ i - 1 ] ) for i = 0..sz-1.

Otherwise (strength is not equal to 0), the following ordered steps apply for i = 1..sz-1:

1. The variable s is set equal to 0.
2. The following steps now apply for j = 0..INTRA\_EDGE\_TAPS-1.
  - a. The variable k is set equal to Clip3( 0, sz - 1, i - 2 + j ).
  - b. The variable s is incremented by Intra\_Edge\_Kernel[ strength - 1 ][ j ] \* edge[ k ].
3. If left is equal to 1, LeftCol[ i - 1 ] is set equal to ( s + 8 ) >> 4.
4. If left is equal to 0, AboveRow[ i - 1 ] is set equal to ( s + 8 ) >> 4.



The array `Intra_Edge_Kernel` is specified as follows:

```
Intra_Edge_Kernel[INTRA_EDGE_KERNELS][INTRA_EDGE_TAPS] = {
  { 0, 4, 8, 4, 0 },
  { 0, 5, 6, 5, 0 },
  { 2, 4, 4, 4, 2 }
}
```

## 7.11.3. Inter prediction process

### 7.11.3.1. General

The inter prediction process is invoked for inter coded blocks and interintra blocks. The inputs to this process are:

- a variable `plane` specifying which plane is being predicted,
- variables `x` and `y` specifying the location of the top left sample in the `CurrFrame[ plane ]` array of the region to be predicted,
- variables `w` and `h` specifying the width and height of the region to be predicted,
- variables `candRow` and `candCol` specifying the location (in units of 4x4 blocks) of the motion vector information to be used.

The outputs of this process are predicted samples in the current frame `CurrFrame`.

This process is triggered by a function call to `predict_inter`.

The variable `isCompound` is set equal to `RefFrames[ candRow ][ candCol ][ 1 ] > INTRA_FRAME`.

The prediction arrays are formed by the following ordered steps:

1. The rounding variables derivation process specified in [section 7.11.3.2](#) is invoked with the variable `isCompound` as input.
2. If `plane` is equal to 0 and `motion_mode` is equal to `LOCALWARP`, the warp estimation process in [section 7.11.3.8](#) is invoked.
3. If `plane` is equal to 0 and `motion_mode` is equal to `LOCALWARP` and `LocalValid` is equal to 1, the setup shear process specified in [section 7.11.3.6](#) is invoked with `LocalWarpParams` as input, and the output `warpValid` is assigned to `LocalValid` (the other outputs are discarded).
4. The variable `refList` is set equal to 0.
5. The variable `refFrame` is set equal to `RefFrames[ candRow ][ candCol ][ refList ]`.

6. If (YMode == GLOBALMV || YMode == GLOBAL\_GLOBALMV) and GmType[ refFrame ] > TRANSLATION, the setup shear process specified in [section 7.11.3.6](#) is invoked with gm\_params[ refFrame ] as input, and the output warpValid is assigned to globalValid (the other outputs are discarded).
7. The variable useWarp (a value of 1 indicates local warping, 2 indicates global warping) is derived as follows:
  - If  $w < 8$  or  $h < 8$ , useWarp is set equal to 0.
  - Otherwise, if force\_integer\_mv is equal to 1, useWarp is set equal to 0.
  - Otherwise, if motion\_mode is equal to LOCALWARP and LocalValid is equal to 1, useWarp is set equal to 1.
  - Otherwise, if all of the following are true, useWarp is set equal to 2.
    - (YMode == GLOBALMV || YMode == GLOBAL\_GLOBALMV).
    - GmType[ refFrame ] > TRANSLATION.
    - is\_scaled( refFrame ) is equal to 0.
    - globalValid is equal to 1.
  - Otherwise, useWarp is set equal to 0.
8. The motion vector array mv is set equal to Mvs[ candRow ][ candCol ][ refList ].
9. The variable refIdx specifying which reference frame is being used is set as follows:
  - If use\_intrabc is equal to 0, refIdx is set equal to ref\_frame\_idx[ refFrame - LAST\_FRAME ].
  - Otherwise (use\_intrabc is equal to 1), refIdx is set equal to -1 and RefFrameWidth[ -1 ] is set equal to FrameWidth, RefFrameHeight[ -1 ] is set equal to FrameHeight, and RefUpscaledWidth[ -1 ] is set equal to UpscaledWidth. (These values ensure that the motion vector scaling has no effect.)
10. The motion vector scaling process in [section 7.11.3.3](#) is invoked with plane, refIdx, x, y, mv as inputs and the output being the initial location startX, startY, and the step sizes stepX, stepY.
11. If use\_intrabc is equal to 1, RefFrameWidth[ -1 ] is set equal to MiCols \* MI\_SIZE, RefFrameHeight[ -1 ] is set equal to MiRows \* MI\_SIZE, and RefUpscaledWidth[ -1 ] is set equal to MiCols \* MI\_SIZE. (These values are needed to avoid intrabc prediction being cropped to the frame boundaries.)
12. If useWarp is not equal to 0, the block warp process in [section 7.11.3.5](#) is invoked with useWarp, plane, refList, x, y, i8, j8, w, h as inputs and the output is merged into the 2D array preds[ refList ] for  $i8 = 0..(h-1) \gg 3$  and for  $j8 = 0..(w-1) \gg 3$ . (Each invocation fills in a block of output of size w by h at x offset  $j8 * 8$  and y offset  $i8 * 8$ .)
13. If useWarp is equal to 0, the block inter prediction process in [section 7.11.3.4](#) is invoked with plane, refIdx, startX, startY, stepX, stepY, w, h, candRow, candCol as inputs and the output is assigned to the 2D array preds[ refList ].

14. If `isCompound` is equal to 1, then the variable `refList` is set equal to 1 and steps 5 to 13 are repeated to form the prediction for the second reference.

An array named `Mask` is prepared as follows:

- If `compound_type` is equal to `COMPOUND_WEDGE` and `plane` is equal to 0, the wedge mask process in [section 7.11.3.11](#) is invoked with `w`, `h` as inputs.
- Otherwise if `compound_type` is equal to `COMPOUND_INTRA`, the intra mode variant mask process in [section 7.11.3.13](#) is invoked with `preds`, `plane`, `x`, `y`, `w`, `h` as inputs.
- Otherwise if `compound_type` is equal to `COMPOUND_DIFFWTD` and `plane` is equal to 0, the difference weight mask process in [section 7.11.3.12](#) is invoked with `preds`, `w`, `h` as inputs.
- Otherwise, no mask array is needed.

If `compound_type` is equal to `COMPOUND_DISTANCE`, the distance weights process in [section 7.11.3.15](#) is invoked with `candRow` and `candCol` as inputs.

The inter predicted samples are then derived as follows:

- If `isCompound` is equal to 0 and `IsInterIntra` is equal to 0, `CurrFrame[ plane ][ y + i ][ x + j ]` is set equal to `Clip1( preds[ 0 ][ i ][ j ] )` for `i = 0..h-1` and `j = 0..w-1`.
- Otherwise if `compound_type` is equal to `COMPOUND_AVERAGE`, `CurrFrame[ plane ][ y + i ][ x + j ]` is set equal to `Clip1( Round2( preds[ 0 ][ i ][ j ] + preds[ 1 ][ i ][ j ], 1 + InterPostRound ) )` for `i = 0..h-1` and `j = 0..w-1`.
- Otherwise if `compound_type` is equal to `COMPOUND_DISTANCE`, `CurrFrame[ plane ][ y + i ][ x + j ]` is set equal to `Clip1( Round2( FwdWeight * preds[ 0 ][ i ][ j ] + BckWeight * preds[ 1 ][ i ][ j ], 4 + InterPostRound ) )` for `i = 0..h-1` and `j = 0..w-1`.
- Otherwise, the mask blend process in [section 7.11.3.14](#) is invoked with `preds`, `plane`, `x`, `y`, `w`, `h` as inputs.

If `motion_mode` is equal to `OBMC`, the overlapped motion compensation in [section 7.11.3.9](#) is invoked with `plane`, `x`, `y`, `w`, `h` as inputs.

### 7.11.3.2. Rounding variables derivation process

The input to this process is a variable `isCompound`.

The rounding variables `InterRound0`, `InterRound1`, and `InterPostRound` are derived as follows:

- `InterRound0` (representing the amount to round by after horizontal filtering) is set equal to 3.
- `InterRound1` (representing the amount to round by after vertical filtering) is set equal to `( isCompound ? 7 : 11 )`.
- If `BitDepth` is equal to 12, `InterRound0` is set equal to `InterRound0 + 2`.
- If `BitDepth` is equal to 12 and `isCompound` is equal to 0, `InterRound1` is set equal to `InterRound1 - 2`.

- InterPostRound (representing the amount to round by at the end of the prediction process) is set equal to  $2 * \text{FILTER\_BITS} - (\text{InterRound0} + \text{InterRound1})$ .

**Note:** The rounding is chosen to ensure that the output of the horizontal filter always fits within 16 bits.

### 7.11.3.3. Motion vector scaling process

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- a variable refIdx specifying which reference frame is being used,
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- a variable mv specifying the clamped motion vector (in units of 1/8 th of a luma sample, i.e. with 3 fractional bits).

The outputs of this process are the variables startX and startY giving the reference block location in units of 1/1024 th of a sample, and variables xStep and yStep giving the step size in units of 1/1024 th of a sample.

This process is responsible for computing the sampling locations in the reference frame based on the motion vector. The sampling locations are also adjusted to compensate for any difference in the size of the reference frame compared to the current frame.

**Note:** When intra block copy is being used, refIdx will be equal to -1 to signal prediction from the frame currently being decoded. The arrays RefFrameWidth, RefFrameHeight, and RefUpscaledWidth include values at index -1 giving the dimensions of the current frame.

It is a requirement of bitstream conformance that all the following conditions are satisfied:

- $2 * \text{FrameWidth} \geq \text{RefUpscaledWidth}[\text{refIdx}]$
- $2 * \text{FrameHeight} \geq \text{RefFrameHeight}[\text{refIdx}]$
- $\text{FrameWidth} \leq 16 * \text{RefUpscaledWidth}[\text{refIdx}]$
- $\text{FrameHeight} \leq 16 * \text{RefFrameHeight}[\text{refIdx}]$

A variable xScale is set equal to  $((\text{RefUpscaledWidth}[\text{refIdx}] \ll \text{REF\_SCALE\_SHIFT}) + (\text{FrameWidth} / 2)) / \text{FrameWidth}$ .

A variable yScale is set equal to  $((\text{RefFrameHeight}[\text{refIdx}] \ll \text{REF\_SCALE\_SHIFT}) + (\text{FrameHeight} / 2)) / \text{FrameHeight}$ .

(xScale and yScale specify the size of the reference frame relative to the current frame in units where  $(1 \ll 14)$  is equivalent to both frames having the same size.)

The variables subX and subY are set equal to the subsampling for the current plane as follows:

- If plane is equal to 0, subX is set equal to 0 and subY is set equal to 0.
- Otherwise, subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The variable halfSample (representing half the size of a sample in units of 1/16 th of a sample) is set equal to  $(1 \ll (\text{SUBPEL\_BITS} - 1))$ .

The variable origX is set equal to  $((x \ll \text{SUBPEL\_BITS}) + ((2 * mv[1]) \gg \text{subX}) + \text{halfSample})$ .

The variable origY is set equal to  $((y \ll \text{SUBPEL\_BITS}) + ((2 * mv[0]) \gg \text{subY}) + \text{halfSample})$ .

(origX and origY specify the location of the centre of the sample at the top-left corner of the reference block in the current frame's coordinate system in units of 1/16 th of a sample, i.e. with SUBPEL\_BITS=4 fractional bits.)

The variable baseX is set equal to  $(\text{origX} * \text{xScale} - (\text{halfSample} \ll \text{REF\_SCALE\_SHIFT}))$ .

The variable baseY is set equal to  $(\text{origY} * \text{yScale} - (\text{halfSample} \ll \text{REF\_SCALE\_SHIFT}))$ .

(baseX and baseY specify the location of the top-left corner of the block in the reference frame in the reference frame's coordinate system with 18 fractional bits.)

The variable off (containing a rounding offset for the filter tap selection) is set equal to  $((1 \ll (\text{SCALE\_SUBPEL\_BITS} - \text{SUBPEL\_BITS})) / 2)$ .

The output variable startX is set equal to  $(\text{Round2Signed}(\text{baseX}, \text{REF\_SCALE\_SHIFT} + \text{SUBPEL\_BITS} - \text{SCALE\_SUBPEL\_BITS}) + \text{off})$ .

The output variable startY is set equal to  $(\text{Round2Signed}(\text{baseY}, \text{REF\_SCALE\_SHIFT} + \text{SUBPEL\_BITS} - \text{SCALE\_SUBPEL\_BITS}) + \text{off})$ .

(startX and startY specify the location of the top-left corner of the block in the reference frame in the reference frame's coordinate system with SCALE\_SUBPEL\_BITS=10 fractional bits.)

The output variable stepX is set equal to  $\text{Round2Signed}(\text{xScale}, \text{REF\_SCALE\_SHIFT} - \text{SCALE\_SUBPEL\_BITS})$ .

The output variable stepY is set equal to  $\text{Round2Signed}(\text{yScale}, \text{REF\_SCALE\_SHIFT} - \text{SCALE\_SUBPEL\_BITS})$ .

(stepX and stepY are the size of one current frame sample in the reference frame's coordinate system with 10 fractional bits.)

### 7.11.3.4. Block inter prediction process

The inputs to this process are:

- a variable plane,
- a variable refIdx specifying which reference frame is being used (or -1 for intra block copy),
- variables x and y giving the block location with in units of 1/1024 th of a sample,
- variables xStep and yStep giving the step size in units of 1/1024 th of a sample,
- variables w and h giving the width and height of the block in units of samples,
- variables candRow and candCol specifying the location (in units of 4x4 blocks) of the motion vector information to be used.

The output from this process is the 2D array named pred containing inter predicted samples.

A variable ref specifying the reference frame contents is set as follows:

- If refIdx is equal to -1, ref is set equal to CurrFrame.
- Otherwise (refIdx is greater than or equal to 0), ref is set equal to FrameStore[ refIdx ].

The variables subX and subY are set equal to the subsampling for the current plane as follows:

- If plane is equal to 0, subX is set equal to 0 and subY is set equal to 0.
- Otherwise, subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The variable lastX is set equal to  $(\text{RefUpscaledWidth}[\text{refIdx}] + \text{subX}) \gg \text{subX} - 1$ .

The variable lastY is set equal to  $(\text{RefFrameHeight}[\text{refIdx}] + \text{subY}) \gg \text{subY} - 1$ .

(lastX and lastY specify the coordinates of the bottom right sample of the reference plane.)

The variable intermediateHeight specifying the height required for the intermediate array is set equal to  $((h - 1) * \text{yStep} + (1 \ll \text{SCALE\_SUBPEL\_BITS}) - 1) \gg \text{SCALE\_SUBPEL\_BITS} + 8$ .

The sub-sample interpolation is effected via two one-dimensional convolutions. First a horizontal filter is used to build up a temporary array, and then this array is vertically filtered to obtain the final prediction. The fractional parts of the motion vectors determine the filtering process. If the fractional part is zero, then the filtering is equivalent to a straight sample copy.

The filtering is applied as follows:

- The array intermediate is specified as follows:

```

interpFilter = InterpFilters[ candRow ][ candCol ][ 1 ]
if ( w <= 4 ) {
    if ( interpFilter == EIGHTTAP || interpFilter == EIGHTTAP_SHARP ) {
        interpFilter = 4
    } else if ( interpFilter == EIGHTTAP_SMOOTH ) {
        interpFilter = 5
    }
}
for ( r = 0; r < intermediateHeight; r++ ) {
    for ( c = 0; c < w; c++ ) {
        s = 0
        p = x + xStep * c
        for ( t = 0; t < 8; t++ )
            s += Subpel_Filters[ interpFilter ][ (p >> 6) & SUBPEL_MASK ][ t ] *
                ref[ plane ][ Clip3( 0, lastY, (y >> 10) + r - 3 ) ]
                [ Clip3( 0, lastX, (p >> 10) + t - 3 ) ]
        intermediate[ r ][ c ] = Round2(s, InterRound0)
    }
}

```

- The array pred is specified as follows:

```

interpFilter = InterpFilters[ candRow ][ candCol ][ 0 ]
if ( h <= 4 ) {
    if ( interpFilter == EIGHTTAP || interpFilter == EIGHTTAP_SHARP ) {
        interpFilter = 4
    } else if ( interpFilter == EIGHTTAP_SMOOTH ) {
        interpFilter = 5
    }
}
for ( r = 0; r < h; r++ ) {
    for ( c = 0; c < w; c++ ) {
        s = 0
        p = (y & 1023) + yStep * r
        for ( t = 0; t < 8; t++ )
            s += Subpel_Filters[ interpFilter ][ (p >> 6) & SUBPEL_MASK ][ t ] *
                intermediate[ (p >> 10) + t ][ c ]
        pred[ r ][ c ] = Round2(s, InterRound1)
    }
}

```

where the constant array Subpel\_Filters is specified as:

```

Subpel_Filters[ 6 ][ 16 ][ 8 ] = {
{
{ 0, 0, 0, 128, 0, 0, 0, 0 },
{ 0, 2, -6, 126, 8, -2, 0, 0 },
{ 0, 2, -10, 122, 18, -4, 0, 0 },
{ 0, 2, -12, 116, 28, -8, 2, 0 },
{ 0, 2, -14, 110, 38, -10, 2, 0 },
{ 0, 2, -14, 102, 48, -12, 2, 0 },
{ 0, 2, -16, 94, 58, -12, 2, 0 },
{ 0, 2, -14, 84, 66, -12, 2, 0 },
{ 0, 2, -14, 76, 76, -14, 2, 0 },
{ 0, 2, -12, 66, 84, -14, 2, 0 },
{ 0, 2, -12, 58, 94, -16, 2, 0 },
{ 0, 2, -12, 48, 102, -14, 2, 0 },
{ 0, 2, -10, 38, 110, -14, 2, 0 },
{ 0, 2, -8, 28, 116, -12, 2, 0 },
{ 0, 0, -4, 18, 122, -10, 2, 0 },
{ 0, 0, -2, 8, 126, -6, 2, 0 }
},
{
{ 0, 0, 0, 128, 0, 0, 0, 0 },
{ 0, 2, 28, 62, 34, 2, 0, 0 },
{ 0, 0, 26, 62, 36, 4, 0, 0 },
{ 0, 0, 22, 62, 40, 4, 0, 0 },
{ 0, 0, 20, 60, 42, 6, 0, 0 },
{ 0, 0, 18, 58, 44, 8, 0, 0 },
{ 0, 0, 16, 56, 46, 10, 0, 0 },
{ 0, -2, 16, 54, 48, 12, 0, 0 },
{ 0, -2, 14, 52, 52, 14, -2, 0 },
{ 0, 0, 12, 48, 54, 16, -2, 0 },
{ 0, 0, 10, 46, 56, 16, 0, 0 },
{ 0, 0, 8, 44, 58, 18, 0, 0 },
{ 0, 0, 6, 42, 60, 20, 0, 0 },
{ 0, 0, 4, 40, 62, 22, 0, 0 },
{ 0, 0, 4, 36, 62, 26, 0, 0 },
{ 0, 0, 2, 34, 62, 28, 2, 0 }
},
{
{ 0, 0, 0, 128, 0, 0, 0, 0 },
{ -2, 2, -6, 126, 8, -2, 2, 0 },
{ -2, 6, -12, 124, 16, -6, 4, -2 },
{ -2, 8, -18, 120, 26, -10, 6, -2 },
{ -4, 10, -22, 116, 38, -14, 6, -2 },
{ -4, 10, -22, 108, 48, -18, 8, -2 },
{ -4, 10, -24, 100, 60, -20, 8, -2 },
{ -4, 10, -24, 90, 70, -22, 10, -2 },
{ -4, 12, -24, 80, 80, -24, 12, -4 },
{ -2, 10, -22, 70, 90, -24, 10, -4 },
{ -2, 8, -20, 60, 100, -24, 10, -4 },
{ -2, 8, -18, 48, 108, -22, 10, -4 },

```



```

    { -2, 6, -14, 38, 116, -22, 10, -4 },
    { -2, 6, -10, 26, 120, -18, 8, -2 },
    { -2, 4, -6, 16, 124, -12, 6, -2 },
    { 0, 2, -2, 8, 126, -6, 2, -2 }
  },
  {
    { 0, 0, 0, 128, 0, 0, 0, 0 },
    { 0, 0, 0, 120, 8, 0, 0, 0 },
    { 0, 0, 0, 112, 16, 0, 0, 0 },
    { 0, 0, 0, 104, 24, 0, 0, 0 },
    { 0, 0, 0, 96, 32, 0, 0, 0 },
    { 0, 0, 0, 88, 40, 0, 0, 0 },
    { 0, 0, 0, 80, 48, 0, 0, 0 },
    { 0, 0, 0, 72, 56, 0, 0, 0 },
    { 0, 0, 0, 64, 64, 0, 0, 0 },
    { 0, 0, 0, 56, 72, 0, 0, 0 },
    { 0, 0, 0, 48, 80, 0, 0, 0 },
    { 0, 0, 0, 40, 88, 0, 0, 0 },
    { 0, 0, 0, 32, 96, 0, 0, 0 },
    { 0, 0, 0, 24, 104, 0, 0, 0 },
    { 0, 0, 0, 16, 112, 0, 0, 0 },
    { 0, 0, 0, 8, 120, 0, 0, 0 }
  },
  {
    { 0, 0, 0, 128, 0, 0, 0, 0 },
    { 0, 0, -4, 126, 8, -2, 0, 0 },
    { 0, 0, -8, 122, 18, -4, 0, 0 },
    { 0, 0, -10, 116, 28, -6, 0, 0 },
    { 0, 0, -12, 110, 38, -8, 0, 0 },
    { 0, 0, -12, 102, 48, -10, 0, 0 },
    { 0, 0, -14, 94, 58, -10, 0, 0 },
    { 0, 0, -12, 84, 66, -10, 0, 0 },
    { 0, 0, -12, 76, 76, -12, 0, 0 },
    { 0, 0, -10, 66, 84, -12, 0, 0 },
    { 0, 0, -10, 58, 94, -14, 0, 0 },
    { 0, 0, -10, 48, 102, -12, 0, 0 },
    { 0, 0, -8, 38, 110, -12, 0, 0 },
    { 0, 0, -6, 28, 116, -10, 0, 0 },
    { 0, 0, -4, 18, 122, -8, 0, 0 },
    { 0, 0, -2, 8, 126, -4, 0, 0 }
  },
  {
    { 0, 0, 0, 128, 0, 0, 0, 0 },
    { 0, 0, 30, 62, 34, 2, 0, 0 },
    { 0, 0, 26, 62, 36, 4, 0, 0 },
    { 0, 0, 22, 62, 40, 4, 0, 0 },
    { 0, 0, 20, 60, 42, 6, 0, 0 },
    { 0, 0, 18, 58, 44, 8, 0, 0 },
    { 0, 0, 16, 56, 46, 10, 0, 0 },
    { 0, 0, 14, 54, 48, 12, 0, 0 },
  }

```

```

    { 0, 0, 12, 52, 52, 12, 0, 0 },
    { 0, 0, 12, 48, 54, 14, 0, 0 },
    { 0, 0, 10, 46, 56, 16, 0, 0 },
    { 0, 0, 8, 44, 58, 18, 0, 0 },
    { 0, 0, 6, 42, 60, 20, 0, 0 },
    { 0, 0, 4, 40, 62, 22, 0, 0 },
    { 0, 0, 4, 36, 62, 26, 0, 0 },
    { 0, 0, 2, 34, 62, 30, 0, 0 }
  }
}

```

**Note:** All the values in Subpel\_Filters are even. The last two filter types are used for small blocks and only have four filter taps. The filter at index 4 has a four tap version of the EIGHTTAP filter. The filter at index 5 has a four tap version of the EIGHTAP\_SMOOTH filter.

### 7.11.3.5. Block warp process

The inputs to this process are:

- a variable useWarp (equal to 1 for local warp, or 2 for global warp),
- a variable plane,
- a variable refList specifying that the process should predict from RefFrame[ refList ],
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- variables i8 and j8 specifying the offset (in units of 8 samples) relative to the top left sample,
- variables w and h giving the width and height of the block in units of samples.

The output from this process is the 2D array named pred containing warped inter predicted samples.

The process only updates a section of the pred array. The size of the updated section is 8x8 samples, clipped to the size of the block. Variables i8 and j8 give the location of the section to update.

A variable refIdx specifying which reference frame is being used is set equal to ref\_frame\_idx[ RefFrame[ refList ] - LAST\_FRAME ].

A variable ref specifying the reference frame contents is set equal to FrameStore[ refIdx ].

The variables subX and subY are set equal to the subsampling for the current plane as follows:

- If plane is equal to 0, subX is set equal to 0 and subY is set equal to 0.
- Otherwise, subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The variable lastX is set equal to  $(\text{RefUpscaledWidth}[\text{refIdx}] + \text{subX}) \gg \text{subX} - 1$ .

The variable lastY is set equal to  $(\text{RefFrameHeight}[\text{refIdx}] + \text{subY}) \gg \text{subY} - 1$ .

(lastX and lastY specify the coordinates of the bottom right sample of the reference plane.)

The variable srcX is set equal to  $(x + j8 * 8 + 4) \ll \text{subX}$ .

The variable srcY is set equal to  $(y + i8 * 8 + 4) \ll \text{subY}$ .

(srcX and srcY specify a location in the luma plane that will be projected using the warp parameters.)

The array warpParams is specified as follows:

- If useWarp is equal to 1, warpParams is set equal to LocalWarpParams.
- Otherwise (useWarp is equal to 2), warpParams is set equal to  $\text{gm\_params}[\text{RefFrame}[\text{refList}]]$ .

The variable dstX is set equal to  $\text{warpParams}[2] * \text{srcX} + \text{warpParams}[3] * \text{srcY} + \text{warpParams}[0]$ .

The variable dstY is set equal to  $\text{warpParams}[4] * \text{srcX} + \text{warpParams}[5] * \text{srcY} + \text{warpParams}[1]$ .

(dstX and dstY specify the destination location in the luma plane using WARPEDMODEL\_PREC\_BITS bits of precision).

The setup shear process specified in [section 7.11.3.6](#) is invoked with warpParams as input, and the outputs are assigned to warpValid, alpha, beta, gamma, and delta. (warpValid will always be equal to 1 at this point.)

The sub-sample interpolation is effected via two one-dimensional convolutions. First a horizontal filter is used to build up an intermediate array, and then this array is vertically filtered to obtain the final prediction.

The filtering is applied as follows:

- The array intermediate is specified as follows:

```

x4 = dstX >> subX
y4 = dstY >> subY
ix4 = x4 >> WARPEDMODEL_PREC_BITS
sx4 = x4 & ((1 << WARPEDMODEL_PREC_BITS) - 1)
iy4 = y4 >> WARPEDMODEL_PREC_BITS
sy4 = y4 & ((1 << WARPEDMODEL_PREC_BITS) - 1)

for ( i1 = -7; i1 < 8; i1++ ) {
    for ( i2 = -4; i2 < 4; i2++ ) {
        sx = sx4 + alpha * i2 + beta * i1
        offs = Round2(sx, WARPEDDIFF_PREC_BITS) + WARPEDPIXEL_PREC_SHIFTS
        s = 0
        for ( i3 = 0; i3 < 8; i3++ ) {
            s += Warped_Filters[ offs ][ i3 ] *
                ref[ plane ][ Clip3( 0, lastY, iy4 + i1 ) ]
                    [ Clip3( 0, lastX, ix4 + i2 - 3 + i3 ) ]
        }
        intermediate[(i1 + 7)][(i2 + 4)] = Round2(s, InterRound0)
    }
}

```

- The array pred is specified as follows:

```

for ( i1 = -4; i1 < Min(4, h - i8 * 8 - 4); i1++ ) {
    for ( i2 = -4; i2 < Min(4, w - j8 * 8 - 4); i2++ ) {
        sy = sy4 + gamma * i2 + delta * i1
        offs = Round2(sy, WARPEDDIFF_PREC_BITS) + WARPEDPIXEL_PREC_SHIFTS
        s = 0
        for ( i3 = 0; i3 < 8; i3++ ) {
            s += Warped_Filters[offs][i3] *
                intermediate[(i1 + i3 + 4)][(i2 + 4)]
        }
        pred[ i8 * 8 + i1 + 4 ][ j8 * 8 + i2 + 4 ] = Round2(s, InterRound1)
    }
}

```

where the constant array Warped\_Filters is specified as:

```

Warped_Filters[WARPEDPIXEL_PREC_SHIFTS * 3 + 1][8] = {
    { 0, 0, 127, 1, 0, 0, 0, 0 }, { 0, -1, 127, 2, 0, 0, 0, 0 },
    { 1, -3, 127, 4, -1, 0, 0, 0 }, { 1, -4, 126, 6, -2, 1, 0, 0 },
    { 1, -5, 126, 8, -3, 1, 0, 0 }, { 1, -6, 125, 11, -4, 1, 0, 0 },
    { 1, -7, 124, 13, -4, 1, 0, 0 }, { 2, -8, 123, 15, -5, 1, 0, 0 },
    { 2, -9, 122, 18, -6, 1, 0, 0 }, { 2, -10, 121, 20, -6, 1, 0, 0 },
    { 2, -11, 120, 22, -7, 2, 0, 0 }, { 2, -12, 119, 25, -8, 2, 0, 0 },
    { 3, -13, 117, 27, -8, 2, 0, 0 }, { 3, -13, 116, 29, -9, 2, 0, 0 },
    { 3, -14, 114, 32, -10, 3, 0, 0 }, { 3, -15, 113, 35, -10, 2, 0, 0 },
    { 3, -15, 111, 37, -11, 3, 0, 0 }, { 3, -16, 109, 40, -11, 3, 0, 0 },
    { 3, -16, 108, 42, -12, 3, 0, 0 }, { 4, -17, 106, 45, -13, 3, 0, 0 },
    { 4, -17, 104, 47, -13, 3, 0, 0 }, { 4, -17, 102, 50, -14, 3, 0, 0 },
    { 4, -17, 100, 52, -14, 3, 0, 0 }, { 4, -18, 98, 55, -15, 4, 0, 0 },
    { 4, -18, 96, 58, -15, 3, 0, 0 }, { 4, -18, 94, 60, -16, 4, 0, 0 },
    { 4, -18, 91, 63, -16, 4, 0, 0 }, { 4, -18, 89, 65, -16, 4, 0, 0 },
    { 4, -18, 87, 68, -17, 4, 0, 0 }, { 4, -18, 85, 70, -17, 4, 0, 0 },
    { 4, -18, 82, 73, -17, 4, 0, 0 }, { 4, -18, 80, 75, -17, 4, 0, 0 },
    { 4, -18, 78, 78, -18, 4, 0, 0 }, { 4, -17, 75, 80, -18, 4, 0, 0 },
    { 4, -17, 73, 82, -18, 4, 0, 0 }, { 4, -17, 70, 85, -18, 4, 0, 0 },
    { 4, -17, 68, 87, -18, 4, 0, 0 }, { 4, -16, 65, 89, -18, 4, 0, 0 },
    { 4, -16, 63, 91, -18, 4, 0, 0 }, { 4, -16, 60, 94, -18, 4, 0, 0 },
    { 3, -15, 58, 96, -18, 4, 0, 0 }, { 4, -15, 55, 98, -18, 4, 0, 0 },
    { 3, -14, 52, 100, -17, 4, 0, 0 }, { 3, -14, 50, 102, -17, 4, 0, 0 },
    { 3, -13, 47, 104, -17, 4, 0, 0 }, { 3, -13, 45, 106, -17, 4, 0, 0 },
    { 3, -12, 42, 108, -16, 3, 0, 0 }, { 3, -11, 40, 109, -16, 3, 0, 0 },
    { 3, -11, 37, 111, -15, 3, 0, 0 }, { 2, -10, 35, 113, -15, 3, 0, 0 },
    { 3, -10, 32, 114, -14, 3, 0, 0 }, { 2, -9, 29, 116, -13, 3, 0, 0 },
    { 2, -8, 27, 117, -13, 3, 0, 0 }, { 2, -8, 25, 119, -12, 2, 0, 0 },
    { 2, -7, 22, 120, -11, 2, 0, 0 }, { 1, -6, 20, 121, -10, 2, 0, 0 },
    { 1, -6, 18, 122, -9, 2, 0, 0 }, { 1, -5, 15, 123, -8, 2, 0, 0 },
    { 1, -4, 13, 124, -7, 1, 0, 0 }, { 1, -4, 11, 125, -6, 1, 0, 0 },
    { 1, -3, 8, 126, -5, 1, 0, 0 }, { 1, -2, 6, 126, -4, 1, 0, 0 },
    { 0, -1, 4, 127, -3, 1, 0, 0 }, { 0, 0, 2, 127, -1, 0, 0, 0 },

    { 0, 0, 0, 127, 1, 0, 0, 0 }, { 0, 0, -1, 127, 2, 0, 0, 0 },
    { 0, 1, -3, 127, 4, -2, 1, 0 }, { 0, 1, -5, 127, 6, -2, 1, 0 },
    { 0, 2, -6, 126, 8, -3, 1, 0 }, { -1, 2, -7, 126, 11, -4, 2, -1 },
    { -1, 3, -8, 125, 13, -5, 2, -1 }, { -1, 3, -10, 124, 16, -6, 3, -1 },
    { -1, 4, -11, 123, 18, -7, 3, -1 }, { -1, 4, -12, 122, 20, -7, 3, -1 },
    { -1, 4, -13, 121, 23, -8, 3, -1 }, { -2, 5, -14, 120, 25, -9, 4, -1 },
    { -1, 5, -15, 119, 27, -10, 4, -1 }, { -1, 5, -16, 118, 30, -11, 4, -1 },
    { -2, 6, -17, 116, 33, -12, 5, -1 }, { -2, 6, -17, 114, 35, -12, 5, -1 },
    { -2, 6, -18, 113, 38, -13, 5, -1 }, { -2, 7, -19, 111, 41, -14, 6, -2 },
    { -2, 7, -19, 110, 43, -15, 6, -2 }, { -2, 7, -20, 108, 46, -15, 6, -2 },
    { -2, 7, -20, 106, 49, -16, 6, -2 }, { -2, 7, -21, 104, 51, -16, 7, -2 },
    { -2, 7, -21, 102, 54, -17, 7, -2 }, { -2, 8, -21, 100, 56, -18, 7, -2 },
    { -2, 8, -22, 98, 59, -18, 7, -2 }, { -2, 8, -22, 96, 62, -19, 7, -2 },
    { -2, 8, -22, 94, 64, -19, 7, -2 }, { -2, 8, -22, 91, 67, -20, 8, -2 },
    { -2, 8, -22, 89, 69, -20, 8, -2 }, { -2, 8, -22, 87, 72, -21, 8, -2 },
    { -2, 8, -21, 84, 74, -21, 8, -2 }, { -2, 8, -22, 82, 77, -21, 8, -2 },

```

```

{-2, 8, -21, 79, 79, -21, 8, -2}, {-2, 8, -21, 77, 82, -22, 8, -2},
{-2, 8, -21, 74, 84, -21, 8, -2}, {-2, 8, -21, 72, 87, -22, 8, -2},
{-2, 8, -20, 69, 89, -22, 8, -2}, {-2, 8, -20, 67, 91, -22, 8, -2},
{-2, 7, -19, 64, 94, -22, 8, -2}, {-2, 7, -19, 62, 96, -22, 8, -2},
{-2, 7, -18, 59, 98, -22, 8, -2}, {-2, 7, -18, 56, 100, -21, 8, -2},
{-2, 7, -17, 54, 102, -21, 7, -2}, {-2, 7, -16, 51, 104, -21, 7, -2},
{-2, 6, -16, 49, 106, -20, 7, -2}, {-2, 6, -15, 46, 108, -20, 7, -2},
{-2, 6, -15, 43, 110, -19, 7, -2}, {-2, 6, -14, 41, 111, -19, 7, -2},
{-1, 5, -13, 38, 113, -18, 6, -2}, {-1, 5, -12, 35, 114, -17, 6, -2},
{-1, 5, -12, 33, 116, -17, 6, -2}, {-1, 4, -11, 30, 118, -16, 5, -1},
{-1, 4, -10, 27, 119, -15, 5, -1}, {-1, 4, -9, 25, 120, -14, 5, -2},
{-1, 3, -8, 23, 121, -13, 4, -1}, {-1, 3, -7, 20, 122, -12, 4, -1},
{-1, 3, -7, 18, 123, -11, 4, -1}, {-1, 3, -6, 16, 124, -10, 3, -1},
{-1, 2, -5, 13, 125, -8, 3, -1}, {-1, 2, -4, 11, 126, -7, 2, -1},
{ 0, 1, -3, 8, 126, -6, 2, 0}, { 0, 1, -2, 6, 127, -5, 1, 0},
{ 0, 1, -2, 4, 127, -3, 1, 0}, { 0, 0, 0, 2, 127, -1, 0, 0},

```

```

{ 0, 0, 0, 1, 127, 0, 0, 0 }, { 0, 0, 0, -1, 127, 2, 0, 0 },
{ 0, 0, 1, -3, 127, 4, -1, 0 }, { 0, 0, 1, -4, 126, 6, -2, 1 },
{ 0, 0, 1, -5, 126, 8, -3, 1 }, { 0, 0, 1, -6, 125, 11, -4, 1 },
{ 0, 0, 1, -7, 124, 13, -4, 1 }, { 0, 0, 2, -8, 123, 15, -5, 1 },
{ 0, 0, 2, -9, 122, 18, -6, 1 }, { 0, 0, 2, -10, 121, 20, -6, 1 },
{ 0, 0, 2, -11, 120, 22, -7, 2 }, { 0, 0, 2, -12, 119, 25, -8, 2 },
{ 0, 0, 3, -13, 117, 27, -8, 2 }, { 0, 0, 3, -13, 116, 29, -9, 2 },
{ 0, 0, 3, -14, 114, 32, -10, 3 }, { 0, 0, 3, -15, 113, 35, -10, 2 },
{ 0, 0, 3, -15, 111, 37, -11, 3 }, { 0, 0, 3, -16, 109, 40, -11, 3 },
{ 0, 0, 3, -16, 108, 42, -12, 3 }, { 0, 0, 4, -17, 106, 45, -13, 3 },
{ 0, 0, 4, -17, 104, 47, -13, 3 }, { 0, 0, 4, -17, 102, 50, -14, 3 },
{ 0, 0, 4, -17, 100, 52, -14, 3 }, { 0, 0, 4, -18, 98, 55, -15, 4 },
{ 0, 0, 4, -18, 96, 58, -15, 3 }, { 0, 0, 4, -18, 94, 60, -16, 4 },
{ 0, 0, 4, -18, 91, 63, -16, 4 }, { 0, 0, 4, -18, 89, 65, -16, 4 },
{ 0, 0, 4, -18, 87, 68, -17, 4 }, { 0, 0, 4, -18, 85, 70, -17, 4 },
{ 0, 0, 4, -18, 82, 73, -17, 4 }, { 0, 0, 4, -18, 80, 75, -17, 4 },
{ 0, 0, 4, -18, 78, 78, -18, 4 }, { 0, 0, 4, -17, 75, 80, -18, 4 },
{ 0, 0, 4, -17, 73, 82, -18, 4 }, { 0, 0, 4, -17, 70, 85, -18, 4 },
{ 0, 0, 4, -17, 68, 87, -18, 4 }, { 0, 0, 4, -16, 65, 89, -18, 4 },
{ 0, 0, 4, -16, 63, 91, -18, 4 }, { 0, 0, 4, -16, 60, 94, -18, 4 },
{ 0, 0, 3, -15, 58, 96, -18, 4 }, { 0, 0, 4, -15, 55, 98, -18, 4 },
{ 0, 0, 3, -14, 52, 100, -17, 4 }, { 0, 0, 3, -14, 50, 102, -17, 4 },
{ 0, 0, 3, -13, 47, 104, -17, 4 }, { 0, 0, 3, -13, 45, 106, -17, 4 },
{ 0, 0, 3, -12, 42, 108, -16, 3 }, { 0, 0, 3, -11, 40, 109, -16, 3 },
{ 0, 0, 3, -11, 37, 111, -15, 3 }, { 0, 0, 2, -10, 35, 113, -15, 3 },
{ 0, 0, 3, -10, 32, 114, -14, 3 }, { 0, 0, 2, -9, 29, 116, -13, 3 },
{ 0, 0, 2, -8, 27, 117, -13, 3 }, { 0, 0, 2, -8, 25, 119, -12, 2 },
{ 0, 0, 2, -7, 22, 120, -11, 2 }, { 0, 0, 1, -6, 20, 121, -10, 2 },
{ 0, 0, 1, -6, 18, 122, -9, 2 }, { 0, 0, 1, -5, 15, 123, -8, 2 },
{ 0, 0, 1, -4, 13, 124, -7, 1 }, { 0, 0, 1, -4, 11, 125, -6, 1 },
{ 0, 0, 1, -3, 8, 126, -5, 1 }, { 0, 0, 1, -2, 6, 126, -4, 1 },
{ 0, 0, 0, -1, 4, 127, -3, 1 }, { 0, 0, 0, 0, 2, 127, -1, 0 },

```

```
{ 0, 0, 0, 0, 2, 127, -1, 0 }
}
```

### 7.11.3.6. Setup shear process

The input for this process is an array warpParams representing an affine transformation.

The outputs from this process are the variable warpValid and variables alpha, beta, gamma, delta representing two shearing operations that combine to make the full affine transformation.

The variable alpha0 is set equal to Clip3( -32768, 32767, warpParams[ 2 ] - (1 << WARPEDMODEL\_PREC\_BITS) ).

The variable beta0 is set equal to Clip3( -32768, 32767, warpParams[ 3 ] ).

The resolve divisor process specified in [section 7.11.3.7](#) is invoked with warpParams[ 2 ] as input, and the outputs assigned to divShift and divFactor.

The variable v is set equal to ( warpParams[ 4 ] << WARPEDMODEL\_PREC\_BITS ).

The variable gamma0 is set equal to Clip3( -32768, 32767, Round2Signed( v \* divFactor, divShift ) ).

The variable w is set equal to ( warpParams[ 3 ] \* warpParams[ 4 ] ).

The variable delta0 is set equal to Clip3( -32768, 32767, warpParams[ 5 ] - Round2Signed( w \* divFactor, divShift ) - (1 << WARPEDMODEL\_PREC\_BITS) ).

The output variables alpha, beta, gamma, delta are set as follows:

```
alpha = Round2Signed( alpha0, WARP_PARAM_REDUCE_BITS ) << WARP_PARAM_REDUCE_BITS
beta  = Round2Signed( beta0,  WARP_PARAM_REDUCE_BITS ) << WARP_PARAM_REDUCE_BITS
gamma = Round2Signed( gamma0, WARP_PARAM_REDUCE_BITS ) << WARP_PARAM_REDUCE_BITS
delta = Round2Signed( delta0, WARP_PARAM_REDUCE_BITS ) << WARP_PARAM_REDUCE_BITS
```

The output warpValid is set as follows:

- If  $4 * \text{Abs}(\alpha) + 7 * \text{Abs}(\beta)$  is greater than or equal to  $(1 \ll \text{WARPEDMODEL\_PREC\_BITS})$ , warpValid is set equal to 0.
- If  $4 * \text{Abs}(\gamma) + 4 * \text{Abs}(\delta)$  is greater than or equal to  $(1 \ll \text{WARPEDMODEL\_PREC\_BITS})$ , warpValid is set equal to 0.
- Otherwise, warpValid is set equal to 1.

### 7.11.3.7. Resolve divisor process

The input for this process is a variable d.

The outputs for this process are variables `divShift` and `divFactor` that can be used to perform an approximate division by `d` via multiplying by `divFactor` and shifting right by `divShift`.

The variable `n` (representing the location of the most significant bit in `Abs(d)`) is set equal to `FloorLog2( Abs(d) )`.

The variable `e` is set equal to `Abs( d ) - ( 1 << n )`.

The variable `f` is set as follows:

- If `n > DIV_LUT_BITS`, `f` is set equal to `Round2( e, n - DIV_LUT_BITS )`.
- Otherwise, `f` is set equal to `e << ( DIV_LUT_BITS - n )`.

The output variable `divShift` is set equal to `( n + DIV_LUT_PREC_BITS )`.

The output variable `divFactor` is set as follows:

- If `d` is less than 0, `divFactor` is set equal to `-Div_Lut[ f ]`.
- Otherwise, `divFactor` is set equal to `Div_Lut[ f ]`.

The lookup table `Div_Lut` is specified as:

```
Div_Lut[DIV_LUT_NUM] = {
    16384, 16320, 16257, 16194, 16132, 16070, 16009, 15948, 15888, 15828, 15768,
    15709, 15650, 15592, 15534, 15477, 15420, 15364, 15308, 15252, 15197, 15142,
    15087, 15033, 14980, 14926, 14873, 14821, 14769, 14717, 14665, 14614, 14564,
    14513, 14463, 14413, 14364, 14315, 14266, 14218, 14170, 14122, 14075, 14028,
    13981, 13935, 13888, 13843, 13797, 13752, 13707, 13662, 13618, 13574, 13530,
    13487, 13443, 13400, 13358, 13315, 13273, 13231, 13190, 13148, 13107, 13066,
    13026, 12985, 12945, 12906, 12866, 12827, 12788, 12749, 12710, 12672, 12633,
    12596, 12558, 12520, 12483, 12446, 12409, 12373, 12336, 12300, 12264, 12228,
    12193, 12157, 12122, 12087, 12053, 12018, 11984, 11950, 11916, 11882, 11848,
    11815, 11782, 11749, 11716, 11683, 11651, 11619, 11586, 11555, 11523, 11491,
    11460, 11429, 11398, 11367, 11336, 11305, 11275, 11245, 11215, 11185, 11155,
    11125, 11096, 11067, 11038, 11009, 10980, 10951, 10923, 10894, 10866, 10838,
    10810, 10782, 10755, 10727, 10700, 10673, 10645, 10618, 10592, 10565, 10538,
    10512, 10486, 10460, 10434, 10408, 10382, 10356, 10331, 10305, 10280, 10255,
    10230, 10205, 10180, 10156, 10131, 10107, 10082, 10058, 10034, 10010, 9986,
    9963, 9939, 9916, 9892, 9869, 9846, 9823, 9800, 9777, 9754, 9732,
    9709, 9687, 9664, 9642, 9620, 9598, 9576, 9554, 9533, 9511, 9489,
    9468, 9447, 9425, 9404, 9383, 9362, 9341, 9321, 9300, 9279, 9259,
    9239, 9218, 9198, 9178, 9158, 9138, 9118, 9098, 9079, 9059, 9039,
    9020, 9001, 8981, 8962, 8943, 8924, 8905, 8886, 8867, 8849, 8830,
    8812, 8793, 8775, 8756, 8738, 8720, 8702, 8684, 8666, 8648, 8630,
    8613, 8595, 8577, 8560, 8542, 8525, 8508, 8490, 8473, 8456, 8439,
    8422, 8405, 8389, 8372, 8355, 8339, 8322, 8306, 8289, 8273, 8257,
    8240, 8224, 8208, 8192
}
```



### 7.11.3.8. Warp estimation process

This process produces the array LocalWarpParams based on NumSamples candidates in CandList by performing a least squares fit.

It also produces a variable LocalValid indicating whether the process was successful.

A 2x2 matrix A, and two length 2 arrays Bx and By are constructed as follows:

```

for ( i = 0; i < 2; i++ ) {
    for ( j = 0; j < 2; j++ ) {
        A[i][j] = 0
    }
    Bx[i] = 0
    By[i] = 0
}
w4 = Num_4x4_Blocks_Wide[MiSize]
h4 = Num_4x4_Blocks_High[MiSize]
midY = MiRow * 4 + h4 * 2 - 1
midX = MiCol * 4 + w4 * 2 - 1
suy = midY * 8
sux = midX * 8
duy = suy + Mv[0][0]
dux = sux + Mv[0][1]
for ( i = 0; i < NumSamples; i++ ) {
    sy = CandList[i][0] - suy
    sx = CandList[i][1] - sux
    dy = CandList[i][2] - duy
    dx = CandList[i][3] - dux
    if ( Abs(sx - dx) < LS_MV_MAX && Abs(sy - dy) < LS_MV_MAX ) {
        A[0][0] += ls_product(sx, sx) + 8
        A[0][1] += ls_product(sx, sy) + 4
        A[1][1] += ls_product(sy, sy) + 8
        Bx[0] += ls_product(sx, dx) + 8
        Bx[1] += ls_product(sy, dx) + 4
        By[0] += ls_product(sx, dy) + 4
        By[1] += ls_product(sy, dy) + 8
    }
}

```

where ls\_product is specified as:

```

ls_product(a, b) {
    return ( (a * b) >> 2) + (a + b)
}

```

**Note:** The matrix A is symmetric so entry A[1][0] is omitted.

The variable det (containing the determinant of the matrix A) is set equal to  $A[0][0] * A[1][1] - A[0][1] * A[0][1]$ .

The variable LocalValid is set as follows:

- If det is equal to 0, LocalValid is set equal to 0.
- Otherwise, LocalValid is set equal to 1.

If det is equal to 0, this process terminates at this point.

The resolve divisor process specified in [section 7.11.3.7](#) is invoked with det as input, and the outputs assigned to divShift and divFactor.

The local warp parameters in LocalWarpParams are derived as follows:

```
divShift -= WARPEDMODEL_PREC_BITS
if ( divShift < 0 ) {
    divFactor = divFactor << (-divShift)
    divShift = 0
}
LocalWarpParams[2] = diag(    A[1][1] * Bx[0] - A[0][1] * Bx[1])
LocalWarpParams[3] = nondiag( -A[0][1] * Bx[0] + A[0][0] * Bx[1])
LocalWarpParams[4] = nondiag(  A[1][1] * By[0] - A[0][1] * By[1])
LocalWarpParams[5] = diag(    -A[0][1] * By[0] + A[0][0] * By[1])

mvx = Mv[ 0 ][ 1 ]
mvy = Mv[ 0 ][ 0 ]
vx = mvx * (1 << (WARPEDMODEL_PREC_BITS - 3)) -
    (midX * (LocalWarpParams[2] - (1 << WARPEDMODEL_PREC_BITS)) + midY * LocalWarpParams[3])
vy = mvy * (1 << (WARPEDMODEL_PREC_BITS - 3)) -
    (midX * LocalWarpParams[4] + midY * (LocalWarpParams[5] - (1 << WARPEDMODEL_PREC_BITS)))
LocalWarpParams[0] = Clip3(-WARPEDMODEL_TRANS_CLAMP, WARPEDMODEL_TRANS_CLAMP - 1, vx)
LocalWarpParams[1] = Clip3(-WARPEDMODEL_TRANS_CLAMP, WARPEDMODEL_TRANS_CLAMP - 1, vy)
```

where diag and nondiag are specified to divide and clamp using divFactor and divShift as follows:

```

nondiag(v) {
    return Clip3(-WARPEDMODEL_NONDIAGAFFINE_CLAMP + 1,
                WARPEDMODEL_NONDIAGAFFINE_CLAMP - 1,
                Round2Signed(v * divFactor, divShift))
}

diag(v) {
    return Clip3((1 << WARPEDMODEL_PREC_BITS) - WARPEDMODEL_NONDIAGAFFINE_CLAMP + 1,
                (1 << WARPEDMODEL_PREC_BITS) + WARPEDMODEL_NONDIAGAFFINE_CLAMP - 1,
                Round2Signed(v * divFactor, divShift))
}

```

### 7.11.3.9. Overlapped motion compensation process

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- variables w and h specifying the width and height of the region to be predicted.

The outputs of this process are modified inter predicted samples in the current frame CurrFrame.

This process blends the inter predicted samples for the current block with inter predicted samples based on motion vectors from the above and left blocks.

The maximum number of overlapped predictions is limited based on the size of the block.

For small blocks, only the left neighbor will be used to form the prediction.

The variables subX and subY describing the subsampling of the current plane are derived as follows:

- If plane is equal to 0, subX and subY are set equal to 0.
- Otherwise (plane is not equal to 0), subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The process is specified as:

```

if ( AvailU ) {
    if ( get_plane_residual_size( MiSize, plane ) >= BLOCK_8X8 ) {
        pass = 0
        w4 = Num_4x4_Blocks_Wide[ MiSize ]
        x4 = MiCol
        y4 = MiRow
        nCount = 0
        nLimit = Min(4, Mi_Width_Log2[ MiSize ])
        while ( nCount < nLimit && x4 < Min( MiCols, MiCol + w4 ) ) {
            candRow = MiRow - 1
            candCol = x4 | 1
            candSz = MiSizes[ candRow ][ candCol ]
            step4 = Clip3( 2, 16, Num_4x4_Blocks_Wide[ candSz ] )
            if ( RefFrames[ candRow ][ candCol ][ 0 ] > INTRA_FRAME ) {
                nCount += 1
                predW = Min( w, ( step4 * MI_SIZE ) >> subX )
                predH = Min( h >> 1, 32 >> subY )
                mask = get_obmc_mask( predH )
                predict_overlap( )
            }
            x4 += step4
        }
    }
}

if ( AvailL ) {
    pass = 1
    h4 = Num_4x4_Blocks_High[ MiSize ]
    x4 = MiCol
    y4 = MiRow
    nCount = 0
    nLimit = Min(4, Mi_Height_Log2[ MiSize ])
    while ( nCount < nLimit && y4 < Min( MiRows, MiRow + h4 ) ) {
        candCol = MiCol - 1
        candRow = y4 | 1
        candSz = MiSizes[ candRow ][ candCol ]
        step4 = Clip3( 2, 16, Num_4x4_Blocks_High[ candSz ] )
        if ( RefFrames[ candRow ][ candCol ][ 0 ] > INTRA_FRAME ) {
            nCount += 1
            predW = Min( w >> 1, 32 >> subX )
            predH = Min( h, ( step4 * MI_SIZE ) >> subY )
            mask = get_obmc_mask( predW )
            predict_overlap( )
        }
        y4 += step4
    }
}

```

When the function `predict_overlap` is invoked, the following ordered steps apply to form the overlap prediction for a region of size `predW` by `predH` based on the candidate motion vector:

1. The motion vector mv is set equal to  $Mvs[ \text{candRow} ][ \text{candCol} ][ 0 ]$ .
2. The variable refIdx is set equal to  $\text{ref\_frame\_idx}[ \text{RefFrames}[ \text{candRow} ][ \text{candCol} ][ 0 ] - \text{LAST\_FRAME} ]$ .
3. The variable predX is set equal to  $(x4 * 4) \gg \text{subX}$ .
4. The variable predY is set equal to  $(y4 * 4) \gg \text{subY}$ .
5. The motion vector scaling process in [section 7.11.3.3](#) is invoked with plane, refIdx, predX, predY, mv as inputs and the output being the initial location startX, startY, and the step sizes stepX, stepY.
6. The block inter prediction process in [section 7.11.3.4](#) is invoked with plane, refIdx, startX, startY, stepX, stepY, predW, predH, candRow, candCol as inputs and the output is assigned to the 2D array obmcPred.
7.  $\text{obmcPred}[ i ][ j ]$  is set equal to  $\text{Clip1}( \text{obmcPred}[ i ][ j ] )$  for  $i = 0..\text{predH}-1$  and  $j = 0..\text{predW}-1$ .
8. The blending process in [section 7.11.3.10](#) is invoked with plane, predX, predY, predW, predH, pass, obmcPred, and mask as inputs.

The function `get_obmc_mask` returns a blending mask as follows:

```
get_obmc_mask( length ) {
    if ( length == 2 ) {
        return Obmc_Mask_2
    } else if ( length == 4 ) {
        return Obmc_Mask_4
    } else if ( length == 8 ) {
        return Obmc_Mask_8
    } else if ( length == 16 ) {
        return Obmc_Mask_16
    } else {
        return Obmc_Mask_32
    }
}
```

The blending masks are defined as follows:

```

Obmc_Mask_2[2] = { 45, 64 }

Obmc_Mask_4[4] = { 39, 50, 59, 64 }

Obmc_Mask_8[8] = { 36, 42, 48, 53, 57, 61, 64, 64 }

Obmc_Mask_16[16] = { 34, 37, 40, 43, 46, 49, 52, 54,
                    56, 58, 60, 61, 64, 64, 64, 64 }

Obmc_Mask_32[32] = { 33, 35, 36, 38, 40, 41, 43, 44,
                    45, 47, 48, 50, 51, 52, 53, 55,
                    56, 57, 58, 59, 60, 60, 61, 62,
                    64, 64, 64, 64, 64, 64, 64, 64 }

```

### 7.11.3.10. Overlap blending process

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- variables predX and predY specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- variables predW and predH specifying the width and height of the region to be predicted,
- a variable pass equal to 0 if blending above samples, or equal to 1 if blending left samples,
- a 2d array obmcPred containing the samples predicted from a neighboring motion vector,
- an array mask containing the blending weights.

The outputs of this process are modified inter predicted samples in the current frame CurrFrame.

For  $i = 0..(\text{predH} - 1)$  and  $j = 0..(\text{predW} - 1)$ , the following ordered steps apply:

1. The variable m specifying the blending factor is specified as follows:
  - If pass is equal to 0 (blend from above), m is set equal to mask[ i ].
  - Otherwise (pass is equal to 1 meaning blend from left), m is set equal to mask[ j ].
2. CurrFrame[ plane ][ predY + i ][ predX + j ] is set equal to  $\text{Round2}( m * \text{CurrFrame}[ \text{plane} ][ \text{predY} + i ][ \text{predX} + j ] + (64 - m) * \text{obmcPred}[ i ][ j ], 6)$

### 7.11.3.11. Wedge mask process

The input to this process is:

- variables w and h specifying the width and height of the region to be predicted.

This process sets up a mask array for the luma samples.

The mask is specified as:

```
for ( i = 0; i < h; i++ ) {  
    for ( j = 0; j < w; j++ ) {  
        Mask[ i ][ j ] = WedgeMasks[ MiSize ][ wedge_sign ][ wedge_index ][ i ][ j ]  
    }  
}
```

where WedgeMasks is a fixed lookup table that is generated by the following function:

```

initialise_wedge_mask_table( ) {
    w = MASK_MASTER_SIZE
    h = MASK_MASTER_SIZE
    for ( j = 0; j < w; j++ ) {
        shift = MASK_MASTER_SIZE / 4
        for ( i = 0; i < h; i += 2 ) {
            MasterMask[ WEDGE_OBLIQUE63 ][ i ][ j ] = Wedge_Master_Oblique_Even[ Clip3( 0,
MASK_MASTER_SIZE - 1, j - shift ) ]
            shift -= 1
            MasterMask[ WEDGE_OBLIQUE63 ][ i + 1 ][ j ] = Wedge_Master_Oblique_Odd[ Clip3( 0,
MASK_MASTER_SIZE - 1, j - shift ) ]
            MasterMask[ WEDGE_VERTICAL ][ i ][ j ] = Wedge_Master_Vertical[ j ]
            MasterMask[ WEDGE_VERTICAL ][ i + 1 ][ j ] = Wedge_Master_Vertical[ j ]
        }
    }
    for ( i = 0; i < h; i++ ) {
        for ( j = 0; j < w; j++ ) {
            msk = MasterMask[ WEDGE_OBLIQUE63 ][ i ][ j ]
            MasterMask[ WEDGE_OBLIQUE27 ][ j ][ i ] = msk
            MasterMask[ WEDGE_OBLIQUE117 ][ i ][ w - 1 - j ] = 64 - msk
            MasterMask[ WEDGE_OBLIQUE153 ][ w - 1 - j ][ i ] = 64 - msk
            MasterMask[ WEDGE_HORIZONTAL ][ j ][ i ] = MasterMask[ WEDGE_VERTICAL ][ i ][ j ]
        }
    }
    for ( bsize = BLOCK_8X8; bsize < BLOCK_SIZES; bsize++ ) {
        if ( Wedge_Bits[ bsize ] > 0 ) {
            w = Block_Width[ bsize ]
            h = Block_Height[ bsize ]
            for ( wedge = 0; wedge < WEDGE_TYPES; wedge++ ) {
                dir = get_wedge_direction(bsize, wedge)
                xoff = MASK_MASTER_SIZE / 2 - ((get_wedge_xoff(bsize, wedge) * w) >> 3)
                yoff = MASK_MASTER_SIZE / 2 - ((get_wedge_yoff(bsize, wedge) * h) >> 3)
                sum = 0
                for ( i = 0; i < w; i++ )
                    sum += MasterMask[ dir ][ yoff ][ xoff+i ]
                for ( i = 1; i < h; i++ )
                    sum += MasterMask[ dir ][ yoff+i ][ xoff ]
                avg = (sum + (w + h - 1) / 2) / (w + h - 1)
                flipSign = (avg < 32)
                for ( i = 0; i < h; i++ ) {
                    for ( j = 0; j < w; j++ ) {
                        WedgeMasks[ bsize ][ flipSign ][ wedge ][ i ][ j ] = MasterMask[ dir ][ yoff+i
][ xoff+j ]
                        WedgeMasks[ bsize ][ !flipSign ][ wedge ][ i ][ j ] = 64 - MasterMask[ dir ][
yoff+i ][ xoff+j ]
                    }
                }
            }
        }
    }
}

```



```
}

```

The 1d lookup tables are defined as:

```
Wedge_Master_Oblique_Odd[MASK_MASTER_SIZE] = {
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  2,  6,  18,
    37, 53, 60, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
}

Wedge_Master_Oblique_Even[MASK_MASTER_SIZE] = {
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  4,  11,  27,
    46, 58, 62, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
}

Wedge_Master_Vertical[MASK_MASTER_SIZE] = {
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  2,  7,  21,
    43, 57, 62, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
}
```

The get\_wedge functions are defined as:

```

block_shape(bsize) {
    w4 = Num_4x4_Blocks_Wide[bsize]
    h4 = Num_4x4_Blocks_High[bsize]
    if ( h4 > w4 )
        return 0
    else if ( h4 < w4 )
        return 1
    else
        return 2
}

get_wedge_direction(bsize, index) {
    return Wedge_Codebook[block_shape(bsize)][index][0]
}

get_wedge_xoff(bsize, index) {
    return Wedge_Codebook[block_shape(bsize)][index][1]
}

get_wedge_yoff(bsize, index) {
    return Wedge_Codebook[block_shape(bsize)][index][2]
}

Wedge_Codebook[3][16][3] = {
    {
        { WEDGE_OBLIQUE27, 4, 4 }, { WEDGE_OBLIQUE63, 4, 4 },
        { WEDGE_OBLIQUE117, 4, 4 }, { WEDGE_OBLIQUE153, 4, 4 },
        { WEDGE_HORIZONTAL, 4, 2 }, { WEDGE_HORIZONTAL, 4, 4 },
        { WEDGE_HORIZONTAL, 4, 6 }, { WEDGE_VERTICAL, 4, 4 },
        { WEDGE_OBLIQUE27, 4, 2 }, { WEDGE_OBLIQUE27, 4, 6 },
        { WEDGE_OBLIQUE153, 4, 2 }, { WEDGE_OBLIQUE153, 4, 6 },
        { WEDGE_OBLIQUE63, 2, 4 }, { WEDGE_OBLIQUE63, 6, 4 },
        { WEDGE_OBLIQUE117, 2, 4 }, { WEDGE_OBLIQUE117, 6, 4 },
    },
    {
        { WEDGE_OBLIQUE27, 4, 4 }, { WEDGE_OBLIQUE63, 4, 4 },
        { WEDGE_OBLIQUE117, 4, 4 }, { WEDGE_OBLIQUE153, 4, 4 },
        { WEDGE_VERTICAL, 2, 4 }, { WEDGE_VERTICAL, 4, 4 },
        { WEDGE_VERTICAL, 6, 4 }, { WEDGE_HORIZONTAL, 4, 4 },
        { WEDGE_OBLIQUE27, 4, 2 }, { WEDGE_OBLIQUE27, 4, 6 },
        { WEDGE_OBLIQUE153, 4, 2 }, { WEDGE_OBLIQUE153, 4, 6 },
        { WEDGE_OBLIQUE63, 2, 4 }, { WEDGE_OBLIQUE63, 6, 4 },
        { WEDGE_OBLIQUE117, 2, 4 }, { WEDGE_OBLIQUE117, 6, 4 },
    },
    {
        { WEDGE_OBLIQUE27, 4, 4 }, { WEDGE_OBLIQUE63, 4, 4 },
        { WEDGE_OBLIQUE117, 4, 4 }, { WEDGE_OBLIQUE153, 4, 4 },
        { WEDGE_HORIZONTAL, 4, 2 }, { WEDGE_HORIZONTAL, 4, 6 },
        { WEDGE_VERTICAL, 2, 4 }, { WEDGE_VERTICAL, 6, 4 },
        { WEDGE_OBLIQUE27, 4, 2 }, { WEDGE_OBLIQUE27, 4, 6 },
    }
}

```

```

    { WEDGE_OBLIQUE153, 4, 2 }, { WEDGE_OBLIQUE153, 4, 6 },
    { WEDGE_OBLIQUE63, 2, 4 }, { WEDGE_OBLIQUE63, 6, 4 },
    { WEDGE_OBLIQUE117, 2, 4 }, { WEDGE_OBLIQUE117, 6, 4 },
  }
}

```

The wedge direction constants used above are defined as follows:

Constant	Value
WEDGE_HORIZONTAL	0
WEDGE_VERTICAL	1
WEDGE_OBLIQUE27	2
WEDGE_OBLIQUE63	3
WEDGE_OBLIQUE117	4
WEDGE_OBLIQUE153	5

### 7.11.3.12. Difference weight mask process

The input to this process is:

- an array preds containing the predicted samples,
- variables w and h specifying the width and height of the region to be predicted.

This process prepares an array Mask containing the blending weights for the luma samples.

The process sets the array based on the difference between the two predictions as follows:

```

for ( i = 0; i < h; i++ ) {
  for ( j = 0; j < w; j++ ) {
    diff = Abs(preds[ 0 ][ i ][ j ] - preds[ 1 ][ i ][ j ])
    diff = Round2(diff, (BitDepth - 8) + InterPostRound)
    m = Clip3(0, 64, 38 + diff / 16)
    if ( mask_type )
      Mask[ i ][ j ] = 64 - m
    else
      Mask[ i ][ j ] = m
  }
}

```

### 7.11.3.13. Intra mode variant mask process

The input to this process is:

- a variable plane specifying which plane is being predicted,
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- variables w and h specifying the width and height of the region to be predicted.

This process prepares an array Mask containing the blending weights for the luma samples.

The process sets the array based on the mode used for intra prediction as follows:

```
sizeScale = MAX_SB_SIZE / Max( h, w )
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        if ( interintra_mode == II_V_PRED ) {
            Mask[ i ][ j ] = Ii_Weights_1d[ i * sizeScale ]
        } else if ( interintra_mode == II_H_PRED ) {
            Mask[ i ][ j ] = Ii_Weights_1d[ j * sizeScale ]
        } else if ( interintra_mode == II_SMOOTH_PRED ) {
            Mask[ i ][ j ] = Ii_Weights_1d[ Min(i, j) * sizeScale ]
        } else {
            Mask[ i ][ j ] = 32
        }
    }
}
```

where the table Ii\_Weights\_1d is defined as:

```
Ii_Weights_1d[MAX_SB_SIZE] = {
    60, 58, 56, 54, 52, 50, 48, 47, 45, 44, 42, 41, 39, 38, 37, 35, 34, 33, 32,
    31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 22, 21, 20, 19, 19, 18, 18, 17, 16,
    16, 15, 15, 14, 14, 13, 13, 12, 12, 12, 11, 11, 10, 10, 10, 9, 9, 9, 8,
    8, 8, 8, 7, 7, 7, 7, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 4, 4,
    4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
}
```

### 7.11.3.14. Mask blend process

The inputs to this process are

- an array preds containing the predicted samples,

- a variable plane specifying which plane is being predicted,
- variables dstX and dstY specifying the location of the top left sample in the CurrFrame[ plane ] array of the region to be predicted,
- variables w and h specifying the width and height of the region to be predicted.

The process combines two predictions according to the mask.

It makes use of an array Mask containing the blending weights to apply (the weights are defined for the current plane samples if compound\_type is equal to COMPOUND\_INTRA, or the luma plane otherwise).

The variables subX and subY describing the subsampling of the current plane are derived as follows:

- If plane is equal to 0, subX and subY are set equal to 0.
- Otherwise (plane is not equal to 0), subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The process is specified as follows:

```

for ( y = 0; y < h; y++ ) {
  for ( x = 0; x < w; x++ ) {
    if ( (!subX && !subY) && (interintra && !wedge_interintra) ) {
      m = Mask[ y ][ x ]
    } else if ( subX && !subY ) {
      m = Round2( Mask[ y ][ 2*x ] + Mask[ y ][ 2*x+1 ], 1 )
    } else if ( !subX && subY ) {
      m = Round2( Mask[ 2*y ][ x ] + Mask[ 2*y+1 ][ x ], 1 )
    } else {
      m = Round2( Mask[ 2*y ][ 2*x ] + Mask[ 2*y ][ 2*x+1 ] +
        Mask[ 2*y+1 ][ 2*x ] + Mask[ 2*y+1 ][ 2*x+1 ], 2 )
    }
    if ( interintra ) {
      pred0 = Clip1( Round2( preds[ 0 ][ y ][ x ], InterPostRound ) )
      pred1 = CurrFrame[plane][y+dstY][x+dstX]
      CurrFrame[plane][y+dstY][x+dstX] = Round2( m * pred1 + (64 - m) * pred0, 6 )
    } else {
      pred0 = preds[ 0 ][ y ][ x ]
      pred1 = preds[ 1 ][ y ][ x ]
      CurrFrame[plane][y+dstY][x+dstX] = Clip1( Round2( m * pred0 + (64 - m) * pred1, 6 +
InterPostRound ) )
    }
  }
}

```

### 7.11.3.15. Distance weights process

The inputs to this process are variables candRow and candCol specifying the location (in units of 4x4 blocks) of the motion vector information to be used.

This process computes weights to be used for blending predictions together based on the expected output times of the reference frames.

The weights are computed as follows:

```
for ( refList = 0; refList < 2; refList++ ) {
    h = OrderHints[ RefFrames[ candRow ][ candCol ][ refList ] ]
    dist[ refList ] = Clip3( 0, MAX_FRAME_DISTANCE, Abs( get_relative_dist( h, OrderHint ) ) )
}
d0 = dist[ 1 ]
d1 = dist[ 0 ]
order = d0 <= d1
if ( d0 == 0 || d1 == 0 ) {
    FwdWeight = Quant_Dist_Lookup[ 3 ][ order ]
    BckWeight = Quant_Dist_Lookup[ 3 ][ 1 - order ]
} else {
    for ( i = 0; i < 3; i++ ) {
        c0 = Quant_Dist_Weight[ i ][ order ]
        c1 = Quant_Dist_Weight[ i ][ 1 - order ]
        if ( order ) {
            if ( d0 * c0 > d1 * c1 )
                break
        } else {
            if ( d0 * c0 < d1 * c1 )
                break
        }
    }
    FwdWeight = Quant_Dist_Lookup[ i ][ order ]
    BckWeight = Quant_Dist_Lookup[ i ][ 1 - order ]
}
```

where the tables Quant\_Dist\_Lookup and Quant\_Dist\_Weight are specified as:

```
Quant_Dist_Weight[ 4 ][ 2 ] = {
    { 2, 3 }, { 2, 5 }, { 2, 7 }, { 1, MAX_FRAME_DISTANCE }
}

Quant_Dist_Lookup[ 4 ][ 2 ] = {
    { 9, 7 }, { 11, 5 }, { 12, 4 }, { 13, 3 }
}
```

## 7.11.4. Palette prediction process

The palette prediction process is invoked for palette coded intra blocks to predict a part of the block using the limited palette.

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- variables startX and startY specifying the location of the top left sample in the CurrFrame[ plane ] array of the current transform block,
- variables x and y specifying the location in 4x4 units relative to the top left sample of the current transform block,
- a variable txSz, specifying the size of the current transform block.

The outputs of this process are palette predicted samples in the current frame CurrFrame.

The variable w specifying the width of the transform block is set equal to Tx\_Width[ txSz ].

The variable h specifying the height of the transform block is set equal to Tx\_Height[ txSz ].

The variable palette is specified as follows:

- If plane is 0, palette is set to palette\_colors\_y.
- Otherwise, if plane is 1, palette is set to palette\_colors\_u.
- Otherwise (plane is 2), palette is set to palette\_colors\_v.

The variable map is specified as follows:

- If plane is 0, map is set to ColorMapY.
- Otherwise (plane is not 0), map is set to ColorMapUV.

The current frame is updated as follows:

- CurrFrame[ plane ][ startY + i ][ startX + j ] is set equal to palette[ map[ y \* 4 + i ][ x \* 4 + j ] ] for i = 0..h-1 and j = 0..w-1.

## 7.11.5. Predict chroma from luma process

The chroma from luma process uses reconstructed luma samples to form a prediction for the chroma samples. The high frequencies are taken from the reconstructed luma samples and combined with DC predicted chroma samples.

The inputs to this process are:

- a variable plane (greater than zero) specifying which plane is being predicted,
- variables startX and startY specifying the location of the top left sample in the CurrFrame[ plane ] array of the current transform block,
- a variable txSz, specifying the size of the current transform block.

The outputs of this process are modified chroma predicted samples in the current frame CurrFrame.

The variable w specifying the width of the transform block is set equal to Tx\_Width[ txSz ].

The variable h specifying the height of the transform block is set equal to Tx\_Height[ txSz ].

The variable subX is set equal to subsampling\_x.

The variable subY is set equal to subsampling\_y.

The variable alpha depends on the plane as follows:

- If plane is equal to 1, alpha is set equal to CflAlphaU.
- Otherwise (plane is equal to 2), alpha is set equal to CflAlphaV.

An array L (containing subsampled reconstructed luma samples with 3 fractional bits of precision) and lumaAvg (representing the average reconstructed luma intensity with 3 fractional bits of precision) is specified as:

```
lumaAvg = 0
for ( i = 0; i < h; i++ ) {
    lumaY = (startY + i) << subY
    lumaY = Min( lumaY, MaxLumaH - (1 << subY) )
    for ( j = 0; j < w; j++ ) {
        lumaX = (startX + j) << subX
        lumaX = Min( lumaX, MaxLumaW - (1 << subX) )
        t = 0
        for ( dy = 0; dy <= subY; dy += 1 )
            for ( dx = 0; dx <= subX; dx += 1 )
                t += CurrFrame[ 0 ][ lumaY + dy ]
                    [ lumaX + dx ]
        v = t << ( 3 - subX - subY )
        L[ i ][ j ] = v
        lumaAvg += v
    }
}
lumaAvg = Round2( lumaAvg, Tx_Width_Log2[ txSz ] + Tx_Height_Log2[ txSz ] )
```

The predicted chroma samples are specified as:

```
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        dc = CurrFrame[ plane ][ startY + i ][ startX + j ]
        scaledLuma = Round2Signed( alpha * ( L[ i ][ j ] - lumaAvg ), 6 )
        CurrFrame[ plane ][ startY + i ][ startX + j ] = Clip1( dc + scaledLuma )
    }
}
```

## 7.12. Reconstruction and dequantization

### 7.12.1. General

This section details the process of reconstructing a block of coefficients using dequantization and inverse transforms.



## 7.12.2. Dequantization functions

This section defines the functions `get_dc_quant` and `get_ac_quant` that are needed by the dequantization process.

The quantization parameters are derived from lookup tables.

The function `dc_q( b )` is specified as `Dc_Qlookup[ (BitDepth-8) >> 1 ][ Clip3( 0, 255, b ) ]` where `Dc_Qlookup` is defined as follows:

```

Dc_Qlookup[ 3 ][ 256 ] = {
{
    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
},
{
    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
},
{

```

```

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
}
}

```

The function `ac_q( b )` is specified as `Ac_Qlookup[ (BitDepth-8) >> 1 ][ Clip3( 0, 255, b ) ]` where `Ac_Qlookup` is defined as follows:

```

Ac_Qlookup[ 3 ][ 256 ] = {
{
  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
},
{
  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
},
{

```

```

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
}
}

```

The function `get_qindex( ignoreDeltaQ, segmentId )` returns the quantizer index for the current block and is specified by the following:

- If `seg_feature_active_idx( segmentId, SEG_LVL_ALT_Q )` is equal to 1 the following ordered steps apply:
  1. Set the variable data equal to `FeatureData[ segmentId ][ SEG_LVL_ALT_Q ]`.
  2. Set `qindex` equal to `base_q_idx + data`.
  3. If `ignoreDeltaQ` is equal to 0 and `delta_q_present` is equal to 1, set `qindex` equal to `CurrentQIndex + data`.
  4. Return `Clip3( 0, 255, qindex )`.

- Otherwise, if ignoreDeltaQ is equal to 0 and delta\_q\_present is equal to 1, return CurrentQIndex.
- Otherwise, return base\_q\_idx.

**Note:** When using both delta quantization and lossless segments, care should be taken that get\_qindex returns 0 for the lossless segments. One approach is to set FeatureData[ segmentId ][ SEG\_LVL\_ALT\_Q ] to -255 for the lossless segments.

The function get\_dc\_quant( plane ) returns the quantizer value for the dc coefficient for a particular plane and is derived as follows:

- If plane is equal to 0, return dc\_q( get\_qindex( 0, segment\_id ) + DeltaQYDc ).
- Otherwise if plane is equal to 1, return dc\_q( get\_qindex( 0, segment\_id ) + DeltaQUDc ).
- Otherwise (plane is equal to 2), return dc\_q( get\_qindex( 0, segment\_id ) + DeltaQVDc ).

The function get\_ac\_quant( plane ) returns the quantizer value for the ac coefficient for a particular plane and is derived as follows:

- If plane is equal to 0, return ac\_q( get\_qindex( 0, segment\_id ) ).
- Otherwise if plane is equal to 1, return ac\_q( get\_qindex( 0, segment\_id ) + DeltaQUAc ).
- Otherwise (plane is equal to 2), return ac\_q( get\_qindex( 0, segment\_id ) + DeltaQVAc ).

### 7.12.3. Reconstruct process

The reconstruct process is invoked to perform dequantization, inverse transform and reconstruction. This process is triggered at a point defined by a function call to reconstruct in the transform block syntax table described in [section 5.11.35](#).

The inputs to this process are:

- a variable plane specifying which plane is being reconstructed,
- variables x and y specifying the location of the top left sample in the CurrFrame[ plane ] array of the current transform block,
- a variable txSz, specifying the size of the transform block.

The outputs of this process are reconstructed samples in the current frame CurrFrame.

The reconstruction and dequantization process is defined as follows:

The variable dqDenom is derived as follows:

- If txSz is equal to TX\_32X32, TX\_16X32, TX\_32X16, TX\_16X64, or TX\_64X16, dqDenom is set equal to 2.

- Otherwise, if txSz is equal to TX\_64X64, TX\_32X64, or TX\_64X32, dqDenom is set equal to 4.
- Otherwise, dqDenom is set equal to 1.

The variable log2W (specifying the base 2 logarithm of the width of the transform block) is set equal to Tx\_Width\_Log2[ txSz ].

The variable log2H (specifying the base 2 logarithm of the height of the transform block) is set equal to Tx\_Height\_Log2[ txSz ].

The variable w (specifying the width of the transform block) is set equal to  $1 \ll \log2W$ .

The variable h (specifying the height of the transform block) is set equal to  $1 \ll \log2H$ .

The variable tw is set equal to  $\text{Min}(32, w)$ .

The variable th is set equal to  $\text{Min}(32, h)$ .

The variable flipUD is derived as follows. If PlaneTxType is equal to one of FLIPADST\_DCT, FLIPADST\_ADST, V\_FLIPADST, or FLIPADST\_FLIPADST, flipUD is set equal to 1. Otherwise, flipUD is set equal to 0.

The variable flipLR is derived as follows. If PlaneTxType is equal to one of DCT\_FLIPADST, ADST\_FLIPADST, H\_FLIPADST, or FLIPADST\_FLIPADST, flipLR is set equal to 1. Otherwise, flipLR is set equal to 0.

The following ordered steps apply:

1. For  $i = 0..(th-1)$ , for  $j = 0..(tw-1)$ , the following ordered steps apply:
  - a. The variable q is derived as follows:
    - If i is equal to 0 and j is equal to 0, the variable q is set equal to `get_dc_quant( plane )`.
    - Otherwise (i, j or both are not equal to 0), the variable q is set equal to `get_ac_quant( plane )`.
  - b. The variable q2 is derived as follows:
    - If `using_qmatrix` is equal to 1, `PlaneTxType` is less than IDTX, and `SegQMLLevel[ plane ][ segment_id ]` is less than 15, q2 is set equal to `Round2( q * Quantizer_Matrix[ SegQMLLevel[ plane ][ segment_id ] ][ plane > 0 ][ Qm_Offset[ txSz ] + i * tw + j ], 5 )`.
    - Otherwise, q2 is set equal to q.
  - c. The variable dq is set equal to `Quant[ i * tw + j ] * q2`.
  - d. The variable sign is set equal to `( dq < 0 ) ? -1 : 1`.
  - e. The variable dq2 is set equal to `sign * ( Abs( dq ) & 0xFFFFFFFF ) / dqDenom`.
  - f. `Dequant[ i ][ j ]` is set equal to `Clip3( - ( 1 << ( 7 + BitDepth ) ), ( 1 << ( 7 + BitDepth ) ) - 1, dq2 )`.

2. Invoke the 2D inverse transform block process defined in [section 7.13.3](#) with the variable txSz as input. The inverse transform outputs are stored in the Residual buffer.
3. For  $i = 0..(h-1)$ , for  $j = 0..(w-1)$ , the following applies:
  - The variable xx is set equal to  $\text{flipLR} ? (w - j - 1) : j$ .
  - The variable yy is set equal to  $\text{flipUD} ? (h - i - 1) : i$ .
  - $\text{CurrFrame}[\text{plane}][y + yy][x + xx]$  is set equal to  $\text{Clip1}(\text{CurrFrame}[\text{plane}][y + yy][x + xx] + \text{Residual}[i][j])$ .

If Lossless is equal to 1, it is a requirement of bitstream conformance that the values written into the Residual array in step 2 are representable by a signed integer with  $1 + \text{BitDepth}$  bits.

**Note:** When Lossless is equal to 0, values written into Residual may not be representable by  $1 + \text{BitDepth}$  bits (for example, due to quantization noise). The constraints in other parts of the specification ensure that the values will always be representable by a signed integer with  $\text{Max}(\text{BitDepth} + 5, 15)$  bits.

## 7.13. Inverse transform process

### 7.13.1. General

This section details the inverse transforms used during the reconstruction processes detailed in [section 7.12](#).

### 7.13.2. 1D transforms

#### 7.13.2.1. Butterfly functions

This section defines the butterfly functions B and H used by the 1D transform processes.

The inverse transform process works by writing values into an array T.

The function `brev(numBits, x)` returns the bit-reversal of numBits of x and is specified as follows:

```
brev( numBits, x ) {
  t = 0
  for ( i = 0; i < numBits; i++ ) {
    bit = (x >> i) & 1
    t += bit << (numBits - 1 - i)
  }
  return t
}
```

The function `B( a, b, angle, 0, r )` performs a butterfly rotation specified by the following ordered steps:



1. The variable  $x$  is set equal to  $T[a] * \cos_{128}(\text{angle}) - T[b] * \sin_{128}(\text{angle})$ .
2. The variable  $y$  is set equal to  $T[a] * \sin_{128}(\text{angle}) + T[b] * \cos_{128}(\text{angle})$ .
3.  $T[a]$  is set equal to  $\text{Round2}(x, 12)$ .
4.  $T[b]$  is set equal to  $\text{Round2}(y, 12)$ .

It is a requirement of bitstream conformance that the values saved into the array  $T$  by this function are representable by a signed integer using  $r$  bits of precision.

The function  $\cos_{128}(\text{angle})$  is specified for integer values of the input  $\text{angle}$  by the following ordered steps:

1. Set a variable  $\text{angle2}$  equal to  $\text{angle} \& 255$ .
2. If  $\text{angle2}$  is greater than or equal to 0 and less than or equal to 64, return  $\text{Cos128\_Lookup}[\text{angle2}]$ .
3. If  $\text{angle2}$  is greater than 64 and less than or equal to 128, return  $\text{Cos128\_Lookup}[128 - \text{angle2}] * -1$ .
4. If  $\text{angle2}$  is greater than 128 and less than or equal to 192, return  $\text{Cos128\_Lookup}[\text{angle2} - 128] * -1$ .
5. Otherwise (if  $\text{angle2}$  is greater than 192 and less than 256), return  $\text{Cos128\_Lookup}[256 - \text{angle2}]$ .

Where  $\text{Cos128\_Lookup}$  is a constant lookup table defined as:

```
Cos128_Lookup[ 65 ] = {
    4096, 4095, 4091, 4085, 4076, 4065, 4052, 4036,
    4017, 3996, 3973, 3948, 3920, 3889, 3857, 3822,
    3784, 3745, 3703, 3659, 3612, 3564, 3513, 3461,
    3406, 3349, 3290, 3229, 3166, 3102, 3035, 2967,
    2896, 2824, 2751, 2675, 2598, 2520, 2440, 2359,
    2276, 2191, 2106, 2019, 1931, 1842, 1751, 1660,
    1567, 1474, 1380, 1285, 1189, 1092, 995, 897,
    799, 700, 601, 501, 401, 301, 201, 101, 0
}
```

The function  $\sin_{128}(\text{angle})$  is defined to be  $\cos_{128}(\text{angle} - 64)$ .

**Note:** The  $\cos_{128}$  function implements the expression  $4096 * \cos(\text{angle} * \pi / 128)$  rounded to the nearest integer. The  $\sin_{128}$  function implements the expression  $4096 * \sin(\text{angle} * \pi / 128)$  rounded to the nearest integer.

When the  $\text{angle}$  is equal to  $32 + 64 * k$  for integer  $k$  the butterfly rotation can be equivalently performed with two fewer multiplications (because the magnitude of  $\cos_{128}(32 + 64 * k)$  is always equal to that of  $\sin_{128}(32 + 64 * k)$ ) by the following process:

1. The variable  $v$  is set equal to  $(\text{angle} \& 64) ? T[a] + T[b] : T[a] - T[b]$ .

2. The variable  $w$  is set equal to  $(\text{angle} \ll 64) \gg -T[a] + T[b] : T[a] + T[b]$ .
3. The variable  $x$  is set equal to  $v * \cos128(\text{angle})$ .
4. The variable  $y$  is set equal to  $w * \cos128(\text{angle})$ .
5.  $T[a]$  is set equal to  $\text{Round2}(x, 12)$ .
6.  $T[b]$  is set equal to  $\text{Round2}(y, 12)$ .

It is a requirement of bitstream conformance that the values saved into the array  $T$  by this function are representable by a signed integer using  $r$  bits of precision.

The function  $B(a, b, \text{angle}, 1, r)$  performs a butterfly rotation and flip specified by the following ordered steps:

1. The function  $B(a, b, \text{angle}, 0, r)$  is invoked.
2. The contents of  $T[a]$  and  $T[b]$  are exchanged.

The function  $H(a, b, 0, r)$  performs a Hadamard rotation specified by the following ordered steps:

1. The variable  $x$  is set equal to  $T[a]$ .
2. The variable  $y$  is set equal to  $T[b]$ .
3.  $T[a]$  is set equal to  $\text{Clip3}(-(1 \ll (r - 1)), (1 \ll (r - 1)) - 1, x + y)$ .
4.  $T[b]$  is set equal to  $\text{Clip3}(-(1 \ll (r - 1)), (1 \ll (r - 1)) - 1, x - y)$ .

The function  $H(a, b, 1, r)$  performs a Hadamard rotation with flipped indices and is specified as follows:

1. The function  $H(b, a, 0, r)$  is invoked.

### 7.13.2.2. Inverse DCT array permutation process

This process performs an in-place permutation of the array  $T$  of length  $2^n$  for  $2 \leq n \leq 6$  which is required before execution of the inverse DCT process.

The input to this process is a variable  $n$  that specifies the base 2 logarithm of the length of the input array.

A temporary array named  $\text{copyT}$  is set equal to  $T$ .

$T[i]$  is set equal to  $\text{copyT}[\text{brev}(n, i)]$  for  $i = 0..((1 \ll n) - 1)$ .

### 7.13.2.3. Inverse DCT process

This process performs an in-place inverse discrete cosine transform of the permuted array  $T$  which is of length  $2^n$  for  $2 \leq n \leq 6$ .

The inputs to this process are:

- a variable  $n$  that specifies the base 2 logarithm of the length of the input array,
- a variable  $r$  that specifies the intermediate clamping range.

The following ordered steps apply:

1. Invoke the inverse DCT permutation process as specified in [section 7.13.2.2](#) with the input variable  $n$ .
2. If  $n$  is equal to 6, invoke  $B(32 + i, 63 - i, 63 - 4 * \text{brev}(4, i), 0, r)$  for  $i = 0..15$ .
3. If  $n$  is greater than or equal to 5, invoke  $B(16 + i, 31 - i, 6 + (\text{brev}(3, 7 - i) << 3), 0, r)$  for  $i = 0..7$ .
4. If  $n$  is equal to 6, invoke  $H(32 + i * 2, 33 + i * 2, i \& 1, r)$  for  $i = 0..15$ .
5. If  $n$  is greater than or equal to 4, invoke  $B(8 + i, 15 - i, 12 + (\text{brev}(2, 3 - i) << 4), 0, r)$  for  $i = 0..3$ .
6. If  $n$  is greater than or equal to 5, invoke  $H(16 + 2 * i, 17 + 2 * i, i \& 1, r)$  for  $i = 0..7$ .
7. If  $n$  is equal to 6, invoke  $B(62 - i * 4 - j, 33 + i * 4 + j, 60 - 16 * \text{brev}(2, i) + 64 * j, 1, r)$  for  $i = 0..3$ , for  $j = 0..1$ .
8. If  $n$  is greater than or equal to 3, invoke  $B(4 + i, 7 - i, 56 - 32 * i, 0, r)$  for  $i = 0..1$ .
9. If  $n$  is greater than or equal to 4, invoke  $H(8 + 2 * i, 9 + 2 * i, i \& 1, r)$  for  $i = 0..3$ .
10. If  $n$  is greater than or equal to 5, invoke  $B(30 - 4 * i - j, 17 + 4 * i + j, 24 + (j << 6) + ((1 - i) << 5), 1, r)$  for  $i = 0..1$ , for  $j = 0..1$ .
11. If  $n$  is equal to 6, invoke  $H(32 + i * 4 + j, 35 + i * 4 - j, i \& 1, r)$  for  $i = 0..7$ , for  $j = 0..1$ .
12. Invoke  $B(2 * i, 2 * i + 1, 32 + 16 * i, 1 - i, r)$  for  $i = 0..1$ .
13. If  $n$  is greater than or equal to 3, invoke  $H(4 + 2 * i, 5 + 2 * i, i, r)$  for  $i = 0..1$ .
14. If  $n$  is greater than or equal to 4, invoke  $B(14 - i, 9 + i, 48 + 64 * i, 1, r)$  for  $i = 0..1$ .
15. If  $n$  is greater than or equal to 5, invoke  $H(16 + 4 * i + j, 19 + 4 * i - j, i \& 1, r)$  for  $i = 0..3$ , for  $j = 0..1$ .
16. If  $n$  is equal to 6, invoke  $B(61 - i * 8 - j, 34 + i * 8 + j, 56 - i * 32 + (j >> 1) * 64, 1, r)$  for  $i = 0..1$ , for  $j = 0..3$ .
17. Invoke  $H(i, 3 - i, 0, r)$  for  $i = 0..1$ .
18. If  $n$  is greater than or equal to 3, invoke  $B(6, 5, 32, 1, r)$ .
19. If  $n$  is greater than or equal to 4, invoke  $H(8 + 4 * i + j, 11 + 4 * i - j, i, r)$  for  $i = 0..1$ , for  $j = 0..1$ .
20. If  $n$  is greater than or equal to 5, invoke  $B(29 - i, 18 + i, 48 + (i >> 1) * 64, 1, r)$  for  $i = 0..3$ .
21. If  $n$  is equal to 6, invoke  $H(32 + 8 * i + j, 39 + 8 * i - j, i \& 1, r)$  for  $i = 0..3$ , for  $j = 0..3$ .
22. If  $n$  is greater than or equal to 3, invoke  $H(i, 7 - i, 0, r)$  for  $i = 0..3$ .

23. If  $n$  is greater than or equal to 4, invoke  $B(13 - i, 10 + i, 32, 1, r)$  for  $i = 0..1$ .
24. If  $n$  is greater than or equal to 5, invoke  $H(16 + i * 8 + j, 23 + i * 8 - j, i, r)$  for  $i = 0..1$ , for  $j = 0..3$ .
25. If  $n$  is equal to 6, invoke  $B(59 - i, 36 + i, i < 4 ? 48 : 112, 1, r)$  for  $i = 0..7$ .
26. If  $n$  is greater than or equal to 4, invoke  $H(i, 15 - i, 0, r)$  for  $i = 0..7$ .
27. If  $n$  is greater than or equal to 5, invoke  $B(27 - i, 20 + i, 32, 1, r)$  for  $i = 0..3$ .
28. If  $n$  is equal to 6, the following steps apply for  $i = 0..7$ :
  - Invoke  $H(32 + i, 47 - i, 0, r)$ .
  - Invoke  $H(48 + i, 63 - i, 1, r)$ .
29. If  $n$  is greater than or equal to 5, invoke  $H(i, 31 - i, 0, r)$  for  $i = 0..15$ .
30. If  $n$  is equal to 6, invoke  $B(55 - i, 40 + i, 32, 1, r)$  for  $i = 0..7$ .
31. If  $n$  is equal to 6, invoke  $H(i, 63 - i, 0, r)$  for  $i = 0..31$ .

#### 7.13.2.4. Inverse ADST input array permutation process

This process performs the in-place permutation of the array  $T$  of length  $2^n$  which is required as the first step of the inverse ADST, where  $3 \leq n \leq 4$ .

The input to this process is a variable  $n$  that specifies the base 2 logarithm of the length of the input array.

The variable  $n0$  is set equal to  $1 \ll n$ .

A temporary array named  $copyT$  is set equal to  $T$ .

The following steps apply for  $i = 0..(n0-1)$ :

- The variable  $idx$  is set equal to  $(i \& 1) ? (i - 1) : (n0 - i - 1)$ .
- $T[i]$  is set equal to  $copyT[idx]$ .

#### 7.13.2.5. Inverse ADST output array permutation process

This process performs the in-place permutation of the array  $T$  of length  $2^n$  which is required before the final step of the inverse ADST, where  $3 \leq n \leq 4$ .

The input to this process is a variable  $n$  that specifies the base 2 logarithm of the length of the input array.

The variable  $n0$  is set equal to  $1 \ll n$ .

A temporary array named  $copyT$  is set equal to  $T$ .

The following steps apply for  $i = 0..(n0-1)$ :

- The variable  $a$  is set equal to  $((i \gg 3) \& 1)$ .
- The variable  $b$  is set equal to  $((i \gg 2) \& 1) \wedge ((i \gg 3) \& 1)$ .
- The variable  $c$  is set equal to  $((i \gg 1) \& 1) \wedge ((i \gg 2) \& 1)$ .
- The variable  $d$  is set equal to  $(i \& 1) \wedge ((i \gg 1) \& 1)$ .
- The variable  $idx$  is set equal to  $((d \ll 3) | (c \ll 2) | (b \ll 1) | a) \gg (4 - n)$ .
- $T[i]$  is set equal to  $(i \& 1) ? (-copyT[idx]) : copyT[idx]$ .

### 7.13.2.6. Inverse ADST4 process

This process performs an in-place inverse ADST process on the array  $T$  of size 4.

The input to this process is a variable  $r$ , specifying the intermediate clamping range.

The following applies:

```

s[ 0 ] = SINPI_1_9 * T[ 0 ]
s[ 1 ] = SINPI_2_9 * T[ 0 ]
s[ 2 ] = SINPI_3_9 * T[ 1 ]
s[ 3 ] = SINPI_4_9 * T[ 2 ]
s[ 4 ] = SINPI_1_9 * T[ 2 ]
s[ 5 ] = SINPI_2_9 * T[ 3 ]
s[ 6 ] = SINPI_4_9 * T[ 3 ]
a7 = T[ 0 ] - T[ 2 ]
b7 = a7 + T[ 3 ]

s[ 0 ] = s[ 0 ] + s[ 3 ]
s[ 1 ] = s[ 1 ] - s[ 4 ]
s[ 3 ] = s[ 2 ]
s[ 2 ] = SINPI_3_9 * b7

s[ 0 ] = s[ 0 ] + s[ 5 ]
s[ 1 ] = s[ 1 ] - s[ 6 ]

x[ 0 ] = s[ 0 ] + s[ 3 ]
x[ 1 ] = s[ 1 ] + s[ 3 ]
x[ 2 ] = s[ 2 ]
x[ 3 ] = s[ 0 ] + s[ 1 ]

x[ 3 ] = x[ 3 ] - s[ 3 ]

T[ 0 ] = Round2( x[ 0 ], 12 )
T[ 1 ] = Round2( x[ 1 ], 12 )
T[ 2 ] = Round2( x[ 2 ], 12 )
T[ 3 ] = Round2( x[ 3 ], 12 )

```

where the constants used are defined as follows:

Symbol	Value
SINPI_1_9	1321
SINPI_2_9	2482
SINPI_3_9	3344
SINPI_4_9	3803

It is a requirement of bitstream conformance that all values stored in the s and x arrays by this process are representable by a signed integer using  $r + 12$  bits of precision.

It is a requirement of bitstream conformance that values stored in the variable a7 by this process are representable by a signed integer using  $r + 1$  bits of precision.

It is a requirement of bitstream conformance that values stored in the variable b7 by this process are representable by a signed integer using  $r$  bits of precision.

### 7.13.2.7. Inverse ADST8 process

This process performs an in-place inverse ADST process on the array T of size 8.

The input to this process is a variable r, specifying the intermediate clamping range.

The following ordered steps apply:

1. Invoke the ADST input array permutation process specified in [section 7.13.2.4](#) with the input variable n set to 3.
2. Invoke  $B(2 * i, 2 * i + 1, 60 - 16 * i, 1, r)$  for  $i = 0..3$ .
3. Invoke  $H(i, 4 + i, 0, r)$  for  $i = 0..3$ .
4. Invoke  $B(4 + 3 * i, 5 + i, 48 - 32 * i, 1, r)$  for  $i = 0..1$ .
5. Invoke  $H(4 * j + i, 2 + 4 * j + i, 0, r)$  for  $i = 0..1$ , for  $j = 0..1$ .
6. Invoke  $B(2 + 4 * i, 3 + 4 * i, 32, 1, r)$  for  $i = 0..1$ .
7. Invoke the ADST output array permutation process specified in [section 7.13.2.5](#) with the input variable n set to 3.

### 7.13.2.8. Inverse ADST16 process

This process performs an in-place inverse ADST process on the array T of size 16.

The input to this process is a variable r, specifying the intermediate clamping range.

The following ordered steps apply:

1. Invoke the ADST input array permutation process specified in [section 7.13.2.4](#) with the input variable n set to 4.
2. Invoke  $B(2 * i, 2 * i + 1, 62 - 8 * i, 1, r)$  for  $i = 0..7$ .
3. Invoke  $H(i, 8 + i, 0, r)$  for  $i = 0..7$ .
4. Invoke  $B(8 + 2 * i, 9 + 2 * i, 56 - 32 * i, 1, r)$  and  $B(13 + 2 * i, 12 + 2 * i, 8 + 32 * i, 1, r)$  for  $i = 0..1$ .
5. Invoke  $H(8 * j + i, 4 + 8 * j + i, 0, r)$  for  $i = 0..3$ , for  $j = 0..1$ .
6. Invoke  $B(4 + 8 * j + 3 * i, 5 + 8 * j + i, 48 - 32 * i, 1, r)$  for  $i = 0..1$ , for  $j = 0..1$ .
7. Invoke  $H(4 * j + i, 2 + 4 * j + i, 0, r)$  for  $i = 0..1$ , for  $j = 0..3$ .
8. Invoke  $B(2 + 4 * i, 3 + 4 * i, 32, 1, r)$  for  $i = 0..3$ .
9. Invoke the ADST output array permutation process specified in [section 7.13.2.5](#) with the input variable n set to 4.

### 7.13.2.9. Inverse ADST process

This process performs an in-place inverse ADST process on the array T of size  $2^n$  for  $2 \leq n \leq 4$ .

The inputs to this process are:

- a variable  $n$  that specifies the base 2 logarithm of the length of the input array.
- a variable  $r$  that specifies the intermediate clamping range.

The following steps apply:

- If  $n$  is equal to 2, invoke the inverse ADST4 process in [section 7.13.2.6](#), with the input variable  $r$ .
- Otherwise, if  $n$  is equal to 3, invoke the inverse ADST8 process in [section 7.13.2.7](#), with the input variable  $r$ .
- Otherwise ( $n$  is equal to 4), invoke the inverse ADST16 process in [section 7.13.2.8](#), with the input variable  $r$ .

### 7.13.2.10. Inverse Walsh-Hadamard transform process

The input to this process is a variable  $shift$  that specifies the amount of pre-scaling.

This process does an in-place transform of the array  $T$  (of length 4) by the following ordered steps:

```

a = T[ 0 ] >> shift
c = T[ 1 ] >> shift
d = T[ 2 ] >> shift
b = T[ 3 ] >> shift
a += c
d -= b
e = (a - d) >> 1
b = e - b
c = e - c
a -= b
d += c
T[ 0 ] = a
T[ 1 ] = b
T[ 2 ] = c
T[ 3 ] = d

```

### 7.13.2.11. Inverse identity transform 4 process

The process does an in-place transform of the array  $T$  (of length 4) by the following calculation for  $i = 0..3$ :

```

T[ i ] = Round2( T[ i ] * 5793, 12 )

```

### 7.13.2.12. Inverse identity transform 8 process

The process does an in-place transform of the array  $T$  (of length 8) by the following calculation for  $i = 0..7$ :



$$T[i] = T[i] * 2$$

### 7.13.2.13. Inverse identity transform 16 process

The process does an in-place transform of the array T (of length 16) by the following calculation for  $i = 0..15$ :

$$T[i] = \text{Round2}(T[i] * 11586, 12)$$

### 7.13.2.14. Inverse identity transform 32 process

The process does an in-place transform of the array T (of length 32) by the following calculation for  $i = 0..31$ :

$$T[i] = T[i] * 4$$

### 7.13.2.15. Inverse identity transform process

This process performs an in-place identity transform process (with a size-dependent scaling factor) on the array T of size  $2^n$  for  $2 \leq n \leq 5$ .

The input to this process is a variable n that specifies the base 2 logarithm of the length of the input array.

The process to invoke depends on n as follows:

- If n is equal to 2, invoke the inverse identity transform 4 process in [section 7.13.2.11](#).
- Otherwise, if n is equal to 3, invoke the inverse identity transform 8 process in [section 7.13.2.12](#).
- Otherwise, if n is equal to 4, invoke the inverse identity transform 16 process in [section 7.13.2.13](#).
- Otherwise (n is equal to 5), invoke the inverse identity transform 32 process in [section 7.13.2.14](#).

## 7.13.3. 2D inverse transform process

This process performs a 2D inverse transform for an array of coefficients stored in the 2D array Dequant. The output is placed in the 2D array Residual.

The input to this process is a variable txSz that specifies the transform size.

Set the variable log2W equal to  $\text{Tx\_Width\_Log2}[txSz]$ .

Set the variable log2H equal to  $\text{Tx\_Height\_Log2}[txSz]$ .

Set the variable w equal to  $1 \ll \log2W$ .

Set the variable h equal to  $1 \ll \log2H$ .

Set the variable rowShift equal to Lossless ? 0 : Transform\_Row\_Shift[ txSz ].

Set the variable colShift equal to Lossless ? 0 : 4.

Set the variable rowClampRange equal to BitDepth + 8.

Set the variable colClampRange equal to Max( BitDepth + 6, 16 ).

The row transforms with  $i = 0..(h-1)$  are applied as follows:

- $T[j]$  is derived as follows for  $j = 0..(w-1)$ :
  - If  $i$  and  $j$  are both less than 32,  $T[j]$  is set equal to Dequant[  $i$  ][  $j$  ].
  - Otherwise,  $T[j]$  is set equal to 0.
- If Abs(  $\log_2 W - \log_2 H$  ) is equal to 1,  $T[j]$  is set equal to Round2(  $T[j] * 2896, 12$  ) for  $j = 0..(w-1)$ .
- If Lossless is equal to 1, invoke the Inverse WHT process as specified in [section 7.13.2.10](#) with shift equal to 2.
- Otherwise, if PlaneTxType is equal to one of DCT\_DCT, ADST\_DCT, FLIPADST\_DCT or H\_DCT, invoke the inverse DCT process as specified in [section 7.13.2.3](#) with the input variable  $n$  equal to  $\log_2 W$  and the input variable  $r$  equal to rowClampRange.
- Otherwise, if PlaneTxType is equal to one of DCT\_ADST, ADST\_ADST, DCT\_FLIPADST, FLIPADST\_FLIPADST, ADST\_FLIPADST, FLIPADST\_ADST, H\_ADST, or H\_FLIPADST, invoke the inverse ADST process as specified in [section 7.13.2.9](#) with input variable  $n$  equal to  $\log_2 W$  and the input variable  $r$  equal to rowClampRange.
- Otherwise, invoke the inverse identity transform process specified in [section 7.13.2.15](#) with the input variable  $n$  equal to  $\log_2 W$ .
- Set Residual[  $i$  ][  $j$  ] equal to Round2(  $T[j]$ , rowShift ) for  $j = 0..(w-1)$ .

Between the row and column transforms, Residual[  $i$  ][  $j$  ] is set equal to Clip3( - (  $1 << ( \text{colClampRange} - 1 )$  ), (  $1 << ( \text{colClampRange} - 1 )$  ) - 1, Residual[  $i$  ][  $j$  ] ) for  $i = 0..(h-1)$ , for  $j = 0..(w-1)$ .

The column transforms with  $j = 0..(w-1)$  are applied as follows:

- Set  $T[i]$  equal to Residual[  $i$  ][  $j$  ] for  $i = 0..(h-1)$ .
- If Lossless is equal to 1, invoke the Inverse WHT process as specified in [section 7.13.2.10](#) with shift equal to 0.
- Otherwise, if PlaneTxType is equal to one of DCT\_DCT, DCT\_ADST, DCT\_FLIPADST or V\_DCT, invoke the inverse DCT process as specified in [section 7.13.2.3](#) with the input variable  $n$  equal to  $\log_2 H$  and the input variable  $r$  equal to colClampRange.

- Otherwise, if PlaneTxType is equal to one of ADST\_DCT, ADST\_ADST, FLIPADST\_DCT, FLIPADST\_FLIPADST, ADST\_FLIPADST, FLIPADST\_ADST, V\_ADST, or V\_FLIPADST, invoke the inverse ADST process as specified in [section 7.13.2.9](#) with input variable n equal to log2H and the input variable r equal to colClampRange.
- Otherwise, invoke the inverse identity transform process specified in [section 7.13.2.15](#) with the input variable n equal to log2H.
- Residual[ i ][ j ] is set equal to Round2( T[ i ], colShift ) for i = 0..(h-1).

where Transform\_Row\_Shift is defined as:

```
Transform_Row_Shift[ TX_SIZES_ALL ] = {
    0, 1, 2, 2, 2, 0, 0, 1, 1,
    1, 1, 1, 1, 1, 1, 2, 2, 2, 2
}
```

## 7.14. Loop filter process

### 7.14.1. General

Input to this process is the array CurrFrame of reconstructed samples.

Output from this process is a modified array CurrFrame containing deblocked samples.

The purpose of the loop filter is to eliminate (or at least reduce) visually objectionable artifacts associated with the semi-independence of the coding of super blocks and their constituent sub-blocks.

The loop filter is applied on all vertical boundaries followed by all horizontal boundaries as follows:

```
for ( plane = 0; plane < NumPlanes; plane++ ) {
    if ( plane == 0 ||
        loop_filter_level[ 1 + plane ] ) {
        for ( pass = 0; pass < 2; pass++ ) {
            rowStep = ( plane == 0 ) ? 1 : ( 1 << subsampling_y )
            colStep = ( plane == 0 ) ? 1 : ( 1 << subsampling_x )
            for ( row = 0; row < MiRows; row += rowStep )
                for ( col = 0; col < MiCols; col += colStep )
                    loop_filter_edge( plane, pass, row, col )
        }
    }
}
```

When the function loop\_filter\_edge is called, the edge loop filter process specified in [section 7.14.2](#) is invoked with the variables plane, pass, row, and col as inputs.

**Note:** The loop filter is an integral part of the decoding process, in that the results of loop filtering are used in the prediction of subsequent frames.

**Note:** The loop filtering is designed so that any order of filtering for the edges will give identical results, provided that the vertical boundaries are filtered before the horizontal boundaries.

## 7.14.2. Edge loop filter process

The inputs to this process are:

- a variable plane specifying whether the process is filtering Y, U, or V samples,
- a variable pass specifying the direction of the edges. pass equal to 0 means the process is filtering vertical block boundaries, and pass equal to 1 means the process is filtering horizontal block boundaries,
- variables row and col specifying the location of the edge in units of 4x4 blocks in the luma plane.

The outputs of this process are modified values in the array CurrFrame.

The variables subX and subY describing the subsampling of the current plane are derived as follows:

- If plane is equal to 0, subX and subY are set equal to 0.
- Otherwise (plane is not equal to 0), subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The variables dx and dy are derived as follows:

- If pass is equal to 0, then dx is set equal to 1, dy is set equal to 0.
- Otherwise (pass is equal to 1), dy is set equal to 1, dx is set equal to 0.

dx and dy specify the offset between the samples to be filtered.

The variable x is set equal to col \* MI\_SIZE.

The variable y is set equal to row \* MI\_SIZE.

x and y contain the location in luma coordinates.

The variables row and col are adjusted as follows:

- row is set equal to ( row | subY )
- col is set equal to ( col | subX )

The variable onScreen (equal to 1 if the samples on both sides of the boundary lie in the visible area) is derived as follows:

- If  $x$  is greater than or equal to  $\text{FrameWidth}$ ,  $\text{onScreen}$  is set equal to 0.
- Otherwise, if  $y$  is greater than or equal to  $\text{FrameHeight}$ ,  $\text{onScreen}$  is set equal to 0.
- Otherwise, if  $\text{pass}$  is equal to 0 and  $x$  is equal to 0,  $\text{onScreen}$  is set equal to 0.
- Otherwise, if  $\text{pass}$  is equal to 1 and  $y$  is equal to 0,  $\text{onScreen}$  is set equal to 0.
- Otherwise,  $\text{onScreen}$  is set equal to 1.

If  $\text{onScreen}$  is equal to 0, then this process immediately returns and no filtering is applied to this edge.

The variables  $xP$  and  $yP$  (containing the location in the current plane) are derived as follows:

- Set  $xP$  equal to  $x \gg \text{subX}$
- Set  $yP$  equal to  $y \gg \text{subY}$

The variables  $\text{prevRow}$  and  $\text{prevCol}$  (containing the location of the mode info block on the other side of the boundary) are derived as follows:

- Set  $\text{prevRow}$  equal to  $\text{row} - (\text{dy} \ll \text{subY})$
- Set  $\text{prevCol}$  equal to  $\text{col} - (\text{dx} \ll \text{subX})$

Set the variable  $\text{MiSize}$  equal to  $\text{MiSizes}[\text{row}][\text{col}]$ .

Set the variable  $\text{txSz}$  equal to  $\text{LoopfilterTxSizes}[\text{plane}][\text{row} \gg \text{subY}][\text{col} \gg \text{subX}]$ .

Set the variable  $\text{planeSize}$  equal to  $\text{get\_plane\_residual\_size}(\text{MiSize}, \text{plane})$

Set the variable  $\text{skip}$  equal to  $\text{Skips}[\text{row}][\text{col}]$ .

Set the variable  $\text{isIntra}$  equal to  $\text{RefFrames}[\text{row}][\text{col}][0] \leq \text{INTRA\_FRAME}$ .

Set the variable  $\text{prevTxSz}$  equal to  $\text{LoopfilterTxSizes}[\text{plane}][\text{prevRow} \gg \text{subY}][\text{prevCol} \gg \text{subX}]$ .

The variable  $\text{isBlockEdge}$  (equal to 1 if the samples cross a prediction block edge) is derived as follows:

- If  $\text{pass}$  is equal to 0 and  $xP$  is an exact multiple of  $\text{Block\_Width}[\text{planeSize}]$ ,  $\text{isBlockEdge}$  is set equal to 1.
- Otherwise, if  $\text{pass}$  is equal to 1 and  $yP$  is an exact multiple of  $\text{Block\_Height}[\text{planeSize}]$ ,  $\text{isBlockEdge}$  is set equal to 1.
- Otherwise,  $\text{isBlockEdge}$  is set equal to 0.

The variable  $\text{isTxEdge}$  (equal to 1 if the samples cross a transform block edge) is derived as follows:

- If  $\text{pass}$  is equal to 0 and  $xP$  is an exact multiple of  $\text{Tx\_Width}[\text{txSz}]$ ,  $\text{isTxEdge}$  is set equal to 1.

- Otherwise, if pass is equal to 1 and yP is an exact multiple of Tx\_Height[ txSz ], isTxEdge is set equal to 1.
- Otherwise, isTxEdge is set equal to 0.

The variable applyFilter (equal to 1 if the samples are filtered) is derived as follows:

- If isTxEdge is equal to 0, applyFilter is set equal to 0.
- Otherwise, if isBlockEdge is equal to 1 or skip is equal to 0 or isIntra is equal to 1, applyFilter is set equal to 1.
- Otherwise applyFilter is set equal to 0.

The filter size process specified in [section 7.14.3](#) is invoked with the inputs txSz, prevTxSz, pass, subX, subY, and plane, and the output assigned to the variable filterSize (containing the maximum filter size that can be used).

The adaptive filter strength process specified in [section 7.14.4](#) is invoked with the inputs row, col, plane, and pass, and the output assigned to the variables lvl, limit, blimit, and thresh.

If lvl is equal to 0, the adaptive filter strength process specified in [section 7.14.4](#) is invoked with the inputs prevRow, prevCol, plane, and pass, and the output assigned to the variables lvl, limit, blimit, and thresh.

For the variable i taking values from 0 to MI\_SIZE - 1, the following applies:

- If applyFilter is equal to 1 and lvl is greater than zero, the sample filtering process specified in [section 7.14.6](#) is invoked with the input variable x set equal to xP + dy \* i, the input variable y set equal to yP + dx \* i, and the variables plane, limit, blimit, thresh, dx, dy, and filterSize supplied as inputs.

**Note:** the vector (dx,dy) represents the direction of the filter, while (dy,dx) represents the direction of the boundary.

## 7.14.3. Filter size process

The inputs to this process are:

- a variable txSz specifying the size of the transform block,
- a variable prevTxSz specifying the size of the transform block on the other side of the boundary,
- a variable pass specifying the direction of the edges,
- variables subX and subY describing the subsampling of the current plane,
- a variable plane specifying whether the process is filtering Y, U, or V samples.

The output of this process is the variable filterSize containing the maximum filter size that can be used in samples.

The purpose of this process is to reduce the width of the chroma filters and to ensure that different boundaries can be filtered in parallel.

The variable `baseSize` is derived as follows:

- If `pass` is equal to 0, `baseSize` is set equal to  $\text{Min}( \text{Tx\_Width}[ \text{prevTxSz} ], \text{Tx\_Width}[ \text{txSz} ] )$ ,
- Otherwise (`pass` is equal to 1), `baseSize` is set equal to  $\text{Min}( \text{Tx\_Height}[ \text{prevTxSz} ], \text{Tx\_Height}[ \text{txSz} ] )$ .

The output variable `filterSize` is derived as follows:

- If `plane` is equal to 0, `filterSize` is set equal to  $\text{Min}( 16, \text{baseSize} )$ ,
- Otherwise, (`plane` is greater than 0), `filterSize` is set equal to  $\text{Min}( 8, \text{baseSize} )$ .

## 7.14.4. Adaptive filter strength process

The inputs to this process are:

- the variables `row` and `col` specifying the luma location in units of 4x4 blocks,
- the variable `plane` specifying whether the process is filtering Y, U or V samples,
- the variable `pass` specifying the direction of the edge being filtered. `pass` equal to 0 means the process is filtering vertical block boundaries, and `pass` equal to 1 means the process is filtering horizontal block boundaries.

The outputs of this process are the variables `lvl`, `limit`, `blimit`, and `thresh`.

The output variable `lvl` is derived as follows:

- The variable `segment` is set equal to `SegmentIds[ row ][ col ]`.
- The variable `ref` is set equal to `RefFrames[ row ][ col ][ 0 ]`.
- The variable `mode` is set equal to `YModes[ row ][ col ]`.
- The variable `modeType` is derived as follows:
  1. If `mode` is greater than or equal to `NEARESTMV`, and not equal to `GLOBALMV`, and not equal to `GLOBAL_GLOBALMV`, `modeType` is set equal to 1.
  2. Otherwise (if `mode` is an intra type or `GLOBALMV` or `GLOBAL_GLOBALMV`), `modeType` is set equal to 0.
- The variable `deltaLF` is derived as follows:
  1. If `delta_lf_multi` is equal to 0, `deltaLF` is set equal to `DeltaLFs[ row ][ col ][ 0 ]`.
  2. Otherwise (`delta_lf_multi` is equal to 1), `deltaLF` is set equal to `DeltaLFs[ row ][ col ][ ( plane == 0 ) ? pass : ( plane + 1 ) ]`.

- The adaptive filter strength selection process specified in [section 7.14.5](#) is invoked, with segment, ref, modeType, deltaLF, plane, and pass as inputs, and the output being the output variable lvl.

The variable shift is derived as follows:

- If loop\_filter\_sharpness is greater than 4, shift is set equal to 2.
- Otherwise, if loop\_filter\_sharpness is greater than 0, shift is set equal to 1.
- Otherwise, shift is set equal to 0.

The output variable limit is derived as follows:

- If loop\_filter\_sharpness is greater than 0, limit is set equal to  $\text{Clip3}(1, 9 - \text{loop\_filter\_sharpness}, \text{lvl} \gg \text{shift})$ .
- Otherwise, limit is set equal to  $\text{Max}(1, \text{lvl} \gg \text{shift})$ .

The output variable blimit is set equal to  $2 * (\text{lvl} + 2) + \text{limit}$ .

The output variable thresh is set equal to  $\text{lvl} \gg 4$ .

## 7.14.5. Adaptive filter strength selection process

The inputs to this process are:

- The variable segment, specifying the current segment id,
- The variable ref, specifying the reference frame type (INTRA\_FRAME, LAST\_FRAME, etc.),
- The variable modeType, specifying the loop filter mode type,
- The variable deltaLF, specifying the loop filter delta value,
- The variable plane, specifying whether the process is filtering Y, U or V samples,
- The variable pass, specifying the direction of the edge being filtered. pass equal to 0 means the process is filtering vertical block boundaries, and pass equal to 1 means the process is filtering horizontal block boundaries.

The output of this process is a filter strength level.

This process is invoked to select a loop filter strength level.

The variable i is set equal to  $(\text{plane} == 0) ? \text{pass} : (\text{plane} + 1)$ .

The variable baseFilterLevel is set equal to  $\text{Clip3}(0, \text{MAX\_LOOP\_FILTER}, \text{deltaLF} + \text{loop\_filter\_level}[\text{i}])$ .

The following ordered steps apply:

1. The variable lvlSeg is set equal to baseFilterLevel.



2. The variable `feature` is set equal to `SEG_LVL_ALT_LF_Y_V + i`.
3. If `seg_feature_active_idx( segment, feature )` is equal to 1 the following ordered steps apply:
  - a. `lvsSeg` is set equal to `FeatureData[ segment ][ feature ] + lvsSeg`.
  - b. `lvsSeg` is set equal to `Clip3( 0, MAX_LOOP_FILTER, lvsSeg )`.
4. If `loop_filter_delta_enabled` is equal to 1, then the following ordered steps apply:
  - a. The variable `nShift` is set equal to `lvsSeg >> 5`.
  - b. If `ref` is equal to `INTRA_FRAME`, then `lvsSeg` is set equal to `lvsSeg + ( loop_filter_ref_deltas[ INTRA_FRAME ] << nShift )`.
  - c. Otherwise, if `ref` is not equal to `INTRA_FRAME`, then `lvsSeg` is set equal to `lvsSeg + ( loop_filter_ref_deltas[ ref ] << nShift ) + ( loop_filter_mode_deltas[ modeType ] << nShift )`.
  - d. `lvsSeg` is set equal to `Clip3(0, MAX_LOOP_FILTER, lvsSeg)`.
5. Return `lvsSeg`.

## 7.14.6. Sample filtering process

### 7.14.6.1. General

The inputs to this process are:

- variables `x` and `y` specifying the location within `CurrFrame[ plane ]`,
- a variable `plane` specifying whether the block is the Y, U or V plane,
- variables `limit`, `blimit`, `thresh` that specify the strength of the filtering operation,
- variables `dx` and `dy` specifying the direction perpendicular to the edge being filtered,
- a variable `filterSize` of specifying the maximum size of filter allowed.

The outputs of this process are modified values in the array `CurrFrame`.

First the filter mask process specified in [section 7.14.6.2](#) is invoked with the inputs `x`, `y`, `plane`, `limit`, `blimit`, `thresh`, `dx`, `dy`, and `filterSize`, and the output is assigned to the variables `hevMask`, `filterMask`, `flatMask`, and `flatMask2`.

Then the appropriate filter process is invoked with the inputs `x`, `y`, `plane`, `dx`, `dy` as follows:

- If `filterMask` is equal to 0, no filter is invoked.
- Otherwise, if `filterSize` is equal to 4 or `flatMask` is equal to 0, the narrow filter process specified in [section 7.14.6.3](#) is invoked with the additional input variable `hevMask`.

- Otherwise, if filterSize is equal to 8 or flatMask2 is equal to 0, the wide filter process specified in [section 7.14.6.4](#) is invoked with the additional input variable log2Size set to 3.
- Otherwise, the wide filter process specified in [section 7.14.6.4](#) is invoked with the additional input variable log2Size set to 4.

### 7.14.6.2. Filter mask process

The inputs to this process are:

- variables x and y specifying the location within CurrFrame[ plane ],
- a variable plane specifying whether the block is the Y, U or V plane,
- variables limit, blimit, thresh that specify the strength of the filtering operation,
- variables dx and dy specifying the direction perpendicular to the edge being filtered,
- a variable filterSize of specifying the maximum size of filter allowed.

The outputs from this process are the variables:

- hevMask,
- filterMask,
- flatMask, (only used if filterSize >= 8),
- flatMask2 (only used if filterSize >= 16).

The values output for these masks depend on the differences between samples on either side of the specified boundary. These samples are specified as follows:

```

q0 = CurrFrame[ plane ][ y ][ x ]
q1 = CurrFrame[ plane ][ y + dy ][ x + dx ]
q2 = CurrFrame[ plane ][ y + dy * 2 ][ x + dx * 2 ]
q3 = CurrFrame[ plane ][ y + dy * 3 ][ x + dx * 3 ]
q4 = CurrFrame[ plane ][ y + dy * 4 ][ x + dx * 4 ]
q5 = CurrFrame[ plane ][ y + dy * 5 ][ x + dx * 5 ]
q6 = CurrFrame[ plane ][ y + dy * 6 ][ x + dx * 6 ]
p0 = CurrFrame[ plane ][ y - dy ][ x - dx ]
p1 = CurrFrame[ plane ][ y - dy * 2 ][ x - dx * 2 ]
p2 = CurrFrame[ plane ][ y - dy * 3 ][ x - dx * 3 ]
p3 = CurrFrame[ plane ][ y - dy * 4 ][ x - dx * 4 ]
p4 = CurrFrame[ plane ][ y - dy * 5 ][ x - dx * 5 ]
p5 = CurrFrame[ plane ][ y - dy * 6 ][ x - dx * 6 ]
p6 = CurrFrame[ plane ][ y - dy * 7 ][ x - dx * 7 ]

```

**Note:** Samples q4, q5, q6, p4, p5, and p6 are only used if filterSize is equal to 16.

The value of hevMask indicates whether the sample has high edge variance. It is calculated as follows:

```
hevMask = 0
threshBd = thresh << (BitDepth - 8)
hevMask |= (Abs( p1 - p0 ) > threshBd)
hevMask |= (Abs( q1 - q0 ) > threshBd)
```

The variable filterLen, representing the number of taps each side of the central sample in the filter, is derived as follows:

- If filterSize is equal to 4, filterLen is set equal to 4.
- Otherwise, if plane is not equal to 0, filterLen is set equal to 6.
- Otherwise, if filterSize is equal to 8, filterLen is set equal to 8.
- Otherwise, filterLen is set equal to 16.

The value of filterMask indicates whether adjacent samples close to the edge (within four samples either side of the specified boundary) vary by less than the limits given by limit and blimit. It is used to determine if any filtering should occur and is calculated as follows:

```
limitBd = limit << (BitDepth - 8)
blimitBd = blimit << (BitDepth - 8)
mask = 0
mask |= (Abs( p1 - p0 ) > limitBd)
mask |= (Abs( q1 - q0 ) > limitBd)
mask |= (Abs( p0 - q0 ) * 2 + Abs( p1 - q1 ) / 2 > blimitBd)
if ( filterLen >= 6 ) {
    mask |= (Abs( p2 - p1 ) > limitBd)
    mask |= (Abs( q2 - q1 ) > limitBd)
}
if ( filterLen >= 8 ) {
    mask |= (Abs( p3 - p2 ) > limitBd)
    mask |= (Abs( q3 - q2 ) > limitBd)
}
filterMask = (mask == 0)
```

The value of flatMask is only required when filterSize >= 8. It measures whether samples from each side of the specified boundary are in a flat region. That is whether those samples are at most  $(1 \ll (\text{BitDepth} - 8))$  different from the sample on the boundary. It is calculated as follows:

```

thresholdBd = 1 << (BitDepth - 8)
if ( filterSize >= 8 ) {
    mask = 0
    mask |= (Abs( p1 - p0 ) > thresholdBd)
    mask |= (Abs( q1 - q0 ) > thresholdBd)
    mask |= (Abs( p2 - p0 ) > thresholdBd)
    mask |= (Abs( q2 - q0 ) > thresholdBd)
    if ( filterLen >= 8 ) {
        mask |= (Abs( p3 - p0 ) > thresholdBd)
        mask |= (Abs( q3 - q0 ) > thresholdBd)
    }
    flatMask = (mask == 0)
}

```

The value of flatMask2 is only required when filterSize >= 16. It measures whether at least seven samples from each side of the specified boundary are in a flat region assuming the first four on each side are (so the full region is flat if flatMask & flatMask2 == 0). The value of flatMask2 is calculated as follows:

```

thresholdBd = 1 << (BitDepth - 8)
if ( filterSize >= 16 ) {
    mask = 0
    mask |= (Abs( p6 - p0 ) > thresholdBd)
    mask |= (Abs( q6 - q0 ) > thresholdBd)
    mask |= (Abs( p5 - p0 ) > thresholdBd)
    mask |= (Abs( q5 - q0 ) > thresholdBd)
    mask |= (Abs( p4 - p0 ) > thresholdBd)
    mask |= (Abs( q4 - q0 ) > thresholdBd)
    flatMask2 = (mask == 0)
}

```

### 7.14.6.3. Narrow filter process

The inputs to this filter are:

- a variable hevMask specifying whether this is a high edge variance case,
- variables x, y specifying the location within CurrFrame[ plane ],
- a variable plane specifying whether the block is the Y, U or V plane,
- variables limit, blimit, thresh that specify the strength of the filtering operation,
- variables dx and dy specifying the direction perpendicular to the edge being filtered.

This process modifies up to two samples on each side of the specified boundary depending on the value of hevMask as follows:

- If hevMask is equal to 0 (i.e. the samples do not have high edge variance), this process modifies two samples on each side of the specified boundary, using a filter constructed from just the inner two (one from each side of the specified boundary).
- Otherwise (the samples do have high edge variance), this process only modifies the one value on each side of the specified boundary, using a filter constructed from four input samples (two from each side of the specified boundary).

The process subtracts  $0x80 \ll (\text{BitDepth} - 8)$  from the input sample values so that they are in the range  $-(1 \ll (\text{BitDepth} - 1))$  to  $(1 \ll (\text{BitDepth} - 1)) - 1$  inclusive. Intermediate values are made to be in this range by the following function:

```
filter4_clamp( value ) {
    return Clip3( -(1 << (BitDepth - 1)), (1 << (BitDepth - 1)) - 1, value )
}
```

The process is specified as follows:

```
q0 = CurrFrame[ plane ][ y ][ x ]
q1 = CurrFrame[ plane ][ y + dy ][ x + dx ]
p0 = CurrFrame[ plane ][ y - dy ][ x - dx ]
p1 = CurrFrame[ plane ][ y - dy * 2 ][ x - dx * 2 ]
ps1 = p1 - (0x80 << (BitDepth - 8))
ps0 = p0 - (0x80 << (BitDepth - 8))
qs0 = q0 - (0x80 << (BitDepth - 8))
qs1 = q1 - (0x80 << (BitDepth - 8))
filter = hevMask ? filter4_clamp( ps1 - qs1 ) : 0
filter = filter4_clamp( filter + 3 * (qs0 - ps0) )
filter1 = filter4_clamp( filter + 4 ) >> 3
filter2 = filter4_clamp( filter + 3 ) >> 3
oq0 = filter4_clamp( qs0 - filter1 ) + (0x80 << (BitDepth - 8))
op0 = filter4_clamp( ps0 + filter2 ) + (0x80 << (BitDepth - 8))
CurrFrame[ plane ][ y ][ x ] = oq0
CurrFrame[ plane ][ y - dy ][ x - dx ] = op0
if ( !hevMask ) {
    filter = Round2( filter1, 1 )
    oq1 = filter4_clamp( qs1 - filter ) + (0x80 << (BitDepth - 8))
    op1 = filter4_clamp( ps1 + filter ) + (0x80 << (BitDepth - 8))
    CurrFrame[ plane ][ y + dy ][ x + dx ] = oq1
    CurrFrame[ plane ][ y - dy * 2 ][ x - dx * 2 ] = op1
}
```

#### 7.14.6.4. Wide filter process

The inputs to this filter are:

- variables x, y specifying the the location within CurrFrame[ plane ],

- a variable plane specifying whether the block is the Y, U or V plane,
- variables dx and dy specifying the direction perpendicular to the edge being filtered,
- a variable log2Size specifying the base 2 logarithm of the number of taps.

This filter is only applied when samples from each side of the boundary are detected to be in a flat region.

The variable n (specifying the number of filter taps on each side of the central sample) is set as follows:

- If log2Size is equal to 4, n is set equal to 6.
- Otherwise if plane is equal to 0, n is set equal to 3.
- Otherwise (log2Size is equal to 3 and plane is greater than 0), n is set equal to 2.

The variable n2 (specifying the number of filter taps equal to 2 on each side of the central sample needed to give a unity DC gain) is set as follows:

- If log2Size is equal to 3 and plane is equal to 0, n2 is set equal to 0.
- Otherwise (log2Size is equal to 4 or plane is greater than 0), n2 is set equal to 1.

This process modifies the samples on each side of the specified boundary by applying a low pass filter as follows:

```
for ( i = -n; i < n; i++ ) {
    t = 0
    for ( j = -n; j <= n; j++ ) {
        p = Clip3( -( n + 1 ), n, i + j )
        tap = ( Abs( j ) <= n2 ) ? 2 : 1
        t += CurrFrame[ plane ][ y + p * dy ][ x + p * dx ] * tap
    }
    F[ i ] = Round2( t, log2Size )
}
for ( i = -n; i < n; i++ )
    CurrFrame[ plane ][ y+i * dy ][ x+i * dx ] = F[ i ]
```

where F is an array with indices from -n to n - 1 used to store the filtered results.

## 7.15. CDEF process

Input to this process is the array CurrFrame of reconstructed samples.

Output from this process is the array CdefFrame containing deringed samples.

The purpose of CDEF is to perform deringing based on the detected direction of blocks.

CDEF parameters are stored for each 64 by 64 block of luma samples.

The CDEF filter is applied on each 8 by 8 block as follows:

```

step4 = Num_4x4_Blocks_Wide[ BLOCK_8X8 ]
cdefSize4 = Num_4x4_Blocks_Wide[ BLOCK_64X64 ]
cdefMask4 = ~(cdefSize4 - 1)
for ( r = 0; r < MiRows; r += step4 ) {
    for ( c = 0; c < MiCols; c += step4 ) {
        baseR = r & cdefMask4
        baseC = c & cdefMask4
        idx = cdef_idx[ baseR ][ baseC ]
        cdef_block(r, c, idx)
    }
}

```

When the `cdef_block` function is called, the CDEF block process specified in [section 7.15.1](#) is invoked with `r`, `c`, and `idx` as inputs.

## 7.15.1. CDEF block process

The inputs to this process are:

- variables `r` and `c` specifying the location of an 8x8 block in units of 4x4 blocks in the luma plane,
- a variable `idx` specifying which set of CDEF parameters to use, or -1 to signal that no filtering should be applied.

The block is first copied to the `CdefFrame` as follows:

```

startY = r * MI_SIZE
endY = startY + MI_SIZE * 2
startX = c * MI_SIZE
endX = startX + MI_SIZE * 2
for ( y = startY; y < endY; y++ ) {
    for ( x = startX; x < endX; x++ ) {
        CdefFrame[ 0 ][ y ][ x ] = CurrFrame[ 0 ][ y ][ x ]
    }
}
if ( NumPlanes > 1 ) {
    startY >>= subsampling_y
    endY >>= subsampling_y
    startX >>= subsampling_x
    endX >>= subsampling_x
    for ( y = startY; y < endY; y++ ) {
        for ( x = startX; x < endX; x++ ) {
            CdefFrame[ 1 ][ y ][ x ] = CurrFrame[ 1 ][ y ][ x ]
            CdefFrame[ 2 ][ y ][ x ] = CurrFrame[ 2 ][ y ][ x ]
        }
    }
}
}

```

**Note** If CDEF filtering turns out to be needed, then the contents of CdefFrame will be overwritten later in this process.

If idx is equal to -1, then the process returns immediately after performing this copy.

The variable coeffShift is set equal to BitDepth - 8.

The variable skip is set equal to ( Skips[ r ][ c ] && Skips[ r + 1 ][ c ] && Skips[ r ][ c + 1 ] && Skips[ r + 1 ][ c + 1 ] ).

If skip is equal to 0, the CDEF direction process specified in [section 7.15.2](#) is invoked with r and c as inputs, and the outputs assigned to variables yDir and var.

If skip is equal to 0, the following ordered steps apply:

1. The variable priStr is set equal to cdef\_y\_pri\_strength[ idx ] << coeffShift.
2. The variable secStr is set equal to cdef\_y\_sec\_strength[ idx ] << coeffShift.
3. The variable dir is set equal to ( priStr == 0 ) ? 0 : yDir.
4. The variable varStr is set equal to ( var >> 6 ) ? Min( FloorLog2( var >> 6 ), 12 ) : 0.
5. The variable priStr is set equal to ( var ? ( priStr \* ( 4 + varStr ) + 8 ) >> 4 : 0 ).
6. The variable damping is set equal to CdefDamping + coeffShift.
7. The CDEF filter process specified in [section 7.15.3](#) is invoked with plane equal to 0, r, c, priStr, secStr, damping, and dir as input.
8. If NumPlanes is equal to 1, the process terminates at this point (i.e. filtering is not done for the U and V planes).
9. The variable priStr is set equal to cdef\_uv\_pri\_strength[ idx ] << coeffShift.
10. The variable secStr is set equal to cdef\_uv\_sec\_strength[ idx ] << coeffShift.
11. The variable dir is set equal to ( priStr == 0 ) ? 0 : Cdef\_Uv\_Dir[ subsampling\_x ][ subsampling\_y ][ yDir ].
12. The variable damping is set equal to CdefDamping + coeffShift - 1.
13. The CDEF filter process specified in [section 7.15.3](#) is invoked with plane equal to 1, r, c, priStr, secStr, damping, and dir as input.
14. The CDEF filter process specified in [section 7.15.3](#) is invoked with plane equal to 2, r, c, priStr, secStr, damping, and dir as input.

Cdef\_Uv\_Dir is a constant lookup table defined as:



```

Cdef_Uv_Dir[ 2 ][ 2 ][ 8 ] = {
    { {0, 1, 2, 3, 4, 5, 6, 7},
      {1, 2, 2, 2, 3, 4, 6, 0} },
    { {7, 0, 2, 4, 5, 6, 6, 6},
      {0, 1, 2, 3, 4, 5, 6, 7} }
}

```

## 7.15.2. CDEF direction process

The inputs to this process are variables *r* and *c* specifying the location of an 8x8 block in units of 4x4 blocks in the luma plane.

The outputs of this process are:

- a variable *yDir* containing the direction of this block,
- a variable *var* containing the variance for this block.

This block uses luma samples to measure the direction and variance of a block.

The process is specified as:

```

for ( i = 0; i < 8; i++ ) {
    cost[i] = 0
    for ( j = 0; j < 15; j++ )
        partial[i][j] = 0
}
bestCost = 0
yDir = 0
x0 = c << MI_SIZE_LOG2
y0 = r << MI_SIZE_LOG2
for ( i = 0; i < 8; i++ ) {
    for ( j = 0; j < 8; j++ ) {
        x = (CurrFrame[ 0 ][y0 + i][x0 + j] >> (BitDepth - 8)) - 128
        partial[0][i + j] += x
        partial[1][i + j / 2] += x
        partial[2][i] += x
        partial[3][3 + i - j / 2] += x
        partial[4][7 + i - j] += x
        partial[5][3 - i / 2 + j] += x
        partial[6][j] += x
        partial[7][i / 2 + j] += x
    }
}
for ( i = 0; i < 8; i++ ) {
    cost[2] += partial[2][i] * partial[2][i]
    cost[6] += partial[6][i] * partial[6][i]
}
cost[2] *= Div_Table[8]
cost[6] *= Div_Table[8]
for ( i = 0; i < 7; i++ ) {
    cost[0] += (partial[0][i] * partial[0][i] +
                partial[0][14 - i] * partial[0][14 - i]) *
                Div_Table[i + 1]
    cost[4] += (partial[4][i] * partial[4][i] +
                partial[4][14 - i] * partial[4][14 - i]) *
                Div_Table[i + 1]
}
cost[0] += partial[0][7] * partial[0][7] * Div_Table[8]
cost[4] += partial[4][7] * partial[4][7] * Div_Table[8]
for ( i = 1; i < 8; i += 2 ) {
    for ( j = 0; j < 4 + 1; j++ ) {
        cost[i] += partial[i][3 + j] * partial[i][3 + j]
    }
    cost[i] *= Div_Table[8]
    for ( j = 0; j < 4 - 1; j++ ) {
        cost[i] += (partial[i][j] * partial[i][j] +
                    partial[i][10 - j] * partial[i][10 - j]) *
                    Div_Table[2 * j + 2]
    }
}
}
for ( i = 0; i < 8; i++ ) {

```

```

    if ( cost[i] > bestCost ) {
        bestCost = cost[i]
        yDir = i
    }
}
var = (bestCost - cost[(yDir + 4) & 7]) >> 10

```

where the Div\_Table is a constant lookup table specified as:

```

Div_Table[9] = {
    0, 840, 420, 280, 210, 168, 140, 120, 105
}

```

### 7.15.3. CDEF filter process

The inputs to this process are:

- a variable plane specifying which plane is being predicted,
- variables r and c specifying the location of an 8x8 block in units of 4x4 blocks in the luma plane,
- a variable priStr specifying the primary filter strength,
- a variable secStr specifying the secondary filter strength,
- a variable damping specifying a shift used for damping,
- a variable dir specifying the detected direction of the block.

The process modifies samples in CdefFrame based on filtering samples from CurrFrame.

MiColStart, MiRowStart, MiColEnd, MiRowEnd are set equal to the values they had when the syntax element MiSizes[ r ][ c ] was written.

**Note:** These variables are used by the `is_inside_filter_region` function to determine which samples are available for use in filtering.

The variable `coeffShift` is set equal to `BitDepth - 8`.

The filtering is applied as follows:

```

subX = (plane > 0) ? subsampling_x : 0
subY = (plane > 0) ? subsampling_y : 0
x0 = (c * MI_SIZE) >> subX
y0 = (r * MI_SIZE) >> subY
w = 8 >> subX
h = 8 >> subY
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        sum = 0
        x = CurrFrame[plane][y0 + i][x0 + j]
        max = x
        min = x
        for ( k = 0; k < 2; k++ ) {
            for ( sign = -1; sign <= 1; sign += 2 ) {
                p = cdef_get_at(plane, x0, y0, i, j, dir, k, sign, subX, subY)
                if ( CdefAvailable ) {
                    sum += Cdef_Pri_Taps[(priStr >> coeffShift) & 1][k] * constrain(p - x, priStr,
damping)

                    max = Max(p, max)
                    min = Min(p, min)
                }
                for ( dirOff = -2; dirOff <= 2; dirOff += 4 ) {
                    s = cdef_get_at(plane, x0, y0, i, j, (dir + dirOff) & 7, k, sign, subX, subY)
                    if ( CdefAvailable ) {
                        sum += Cdef_Sec_Taps[(priStr >> coeffShift) & 1][k] * constrain(s - x,
secStr, damping)

                        max = Max(s, max)
                        min = Min(s, min)
                    }
                }
            }
        }
        CdefFrame[plane][y0 + i][x0 + j] = Clip3(min, max, x + ((8 + sum - (sum < 0)) >> 4) )
    }
}

```

where Cdef\_Pri\_Taps and Cdef\_Sec\_Taps are constant lookup tables specified as:

```

Cdef_Pri_Taps[2][2] = {
    { 4, 2 }, { 3, 3 }
}

Cdef_Sec_Taps[2][2] = {
    { 2, 1 }, { 2, 1 }
}

```

constrain is specified as:

```

constrain(diff, threshold, damping) {
    if ( !threshold )
        return 0
    dampingAdj = Max(0, damping - FloorLog2( threshold ) )
    sign = (diff < 0) ? -1 : 1
    return sign * Clip3(0, Abs(diff), threshold - (Abs(diff) >> dampingAdj) )
}

```

cdef\_get\_at fetches a sample from CurrFrame and sets CdefAvailable according to whether the sample is available. cdef\_get\_at is specified as:

```

cdef_get_at(plane, x0, y0, i, j, dir, k, sign, subX, subY) {
    y = y0 + i + sign * Cdef_Directions[dir][k][0]
    x = x0 + j + sign * Cdef_Directions[dir][k][1]
    candidateR = (y << subY) >> MI_SIZE_LOG2
    candidateC = (x << subX) >> MI_SIZE_LOG2
    if ( is_inside_filter_region( candidateR, candidateC ) ) {
        CdefAvailable = 1
        return CurrFrame[ plane ][ y ][ x ]
    } else {
        CdefAvailable = 0
        return 0
    }
}

```

where Cdef\_Directions is a constant lookup table defined as:

```

Cdef_Directions[8][2][2] = {
    { { -1, 1 }, { -2, 2 } },
    { { 0, 1 }, { -1, 2 } },
    { { 0, 1 }, { 0, 2 } },
    { { 0, 1 }, { 1, 2 } },
    { { 1, 1 }, { 2, 2 } },
    { { 1, 0 }, { 2, 1 } },
    { { 1, 0 }, { 2, 0 } },
    { { 1, 0 }, { 2, -1 } }
}

```

## 7.16. Upscaling process

Input to this process is an array inputFrame of width FrameWidth and height FrameHeight.

The output of this process is a horizontally upsampled frame of width UpscaledWidth and height FrameHeight.

If use\_superres is equal to 0, no upscaling is required and this process returns inputFrame.

This process is specified as:

```

for ( plane = 0; plane < NumPlanes; plane++ ) {
    if ( plane > 0 ) {
        subX = subsampling_x
        subY = subsampling_y
    } else {
        subX = 0
        subY = 0
    }
    downscaledPlaneW = Round2(FrameWidth, subX)
    upscaledPlaneW = Round2(UpscaledWidth, subX)
    planeH = Round2(FrameHeight, subY)
    stepX = ((downscaledPlaneW << SUPERRES_SCALE_BITS) + (upscaledPlaneW / 2)) / upscaledPlaneW
    err = (upscaledPlaneW * stepX) - (downscaledPlaneW << SUPERRES_SCALE_BITS)
    initialSubpelX =
        (-(upscaledPlaneW - downscaledPlaneW) << (SUPERRES_SCALE_BITS - 1)) + upscaledPlaneW / 2) /
    upscaledPlaneW +
        (1 << (SUPERRES_EXTRA_BITS - 1)) - err / 2
    initialSubpelX &= SUPERRES_SCALE_MASK
    miW = MiCols >> subX
    minX = 0
    maxX = miW * MI_SIZE - 1
    for ( y = 0; y < planeH; y++ ) {
        for ( x = 0; x < upscaledPlaneW; x++ ) {
            srcX = -(1 << SUPERRES_SCALE_BITS) + initialSubpelX + x*stepX
            srcXPx = (srcX >> SUPERRES_SCALE_BITS)
            srcXSubpel = (srcX & SUPERRES_SCALE_MASK) >> SUPERRES_EXTRA_BITS
            sum = 0
            for ( k = 0; k < SUPERRES_FILTER_TAPS; k++ ) {
                sampleX = Clip3(minX, maxX, srcXPx + (k - SUPERRES_FILTER_OFFSET))
                px = frame[plane][y][sampleX]
                sum += px * Upscale_Filter[srcXSubpel][k]
            }
            outputFrame[plane][y][x] = Clip1(Round2(sum, FILTER_BITS))
        }
    }
}

```

where Upscale\_Filter is specified as:

```

Upscale_Filter[SUPERRES_FILTER_SHIFTS][SUPERRES_FILTER_TAPS] = {
  { 0, 0, 0, 128, 0, 0, 0, 0 },      { 0, 0, -1, 128, 2, -1, 0, 0 },
  { 0, 1, -3, 127, 4, -2, 1, 0 },    { 0, 1, -4, 127, 6, -3, 1, 0 },
  { 0, 2, -6, 126, 8, -3, 1, 0 },    { 0, 2, -7, 125, 11, -4, 1, 0 },
  { -1, 2, -8, 125, 13, -5, 2, 0 },   { -1, 3, -9, 124, 15, -6, 2, 0 },
  { -1, 3, -10, 123, 18, -6, 2, -1 }, { -1, 3, -11, 122, 20, -7, 3, -1 },
  { -1, 4, -12, 121, 22, -8, 3, -1 }, { -1, 4, -13, 120, 25, -9, 3, -1 },
  { -1, 4, -14, 118, 28, -9, 3, -1 }, { -1, 4, -15, 117, 30, -10, 4, -1 },
  { -1, 5, -16, 116, 32, -11, 4, -1 }, { -1, 5, -16, 114, 35, -12, 4, -1 },
  { -1, 5, -17, 112, 38, -12, 4, -1 }, { -1, 5, -18, 111, 40, -13, 5, -1 },
  { -1, 5, -18, 109, 43, -14, 5, -1 }, { -1, 6, -19, 107, 45, -14, 5, -1 },
  { -1, 6, -19, 105, 48, -15, 5, -1 }, { -1, 6, -19, 103, 51, -16, 5, -1 },
  { -1, 6, -20, 101, 53, -16, 6, -1 }, { -1, 6, -20, 99, 56, -17, 6, -1 },
  { -1, 6, -20, 97, 58, -17, 6, -1 }, { -1, 6, -20, 95, 61, -18, 6, -1 },
  { -2, 7, -20, 93, 64, -18, 6, -2 }, { -2, 7, -20, 91, 66, -19, 6, -1 },
  { -2, 7, -20, 88, 69, -19, 6, -1 }, { -2, 7, -20, 86, 71, -19, 6, -1 },
  { -2, 7, -20, 84, 74, -20, 7, -2 }, { -2, 7, -20, 81, 76, -20, 7, -1 },
  { -2, 7, -20, 79, 79, -20, 7, -2 }, { -1, 7, -20, 76, 81, -20, 7, -2 },
  { -2, 7, -20, 74, 84, -20, 7, -2 }, { -1, 6, -19, 71, 86, -20, 7, -2 },
  { -1, 6, -19, 69, 88, -20, 7, -2 }, { -1, 6, -19, 66, 91, -20, 7, -2 },
  { -2, 6, -18, 64, 93, -20, 7, -2 }, { -1, 6, -18, 61, 95, -20, 6, -1 },
  { -1, 6, -17, 58, 97, -20, 6, -1 }, { -1, 6, -17, 56, 99, -20, 6, -1 },
  { -1, 6, -16, 53, 101, -20, 6, -1 }, { -1, 5, -16, 51, 103, -19, 6, -1 },
  { -1, 5, -15, 48, 105, -19, 6, -1 }, { -1, 5, -14, 45, 107, -19, 6, -1 },
  { -1, 5, -14, 43, 109, -18, 5, -1 }, { -1, 5, -13, 40, 111, -18, 5, -1 },
  { -1, 4, -12, 38, 112, -17, 5, -1 }, { -1, 4, -12, 35, 114, -16, 5, -1 },
  { -1, 4, -11, 32, 116, -16, 5, -1 }, { -1, 4, -10, 30, 117, -15, 4, -1 },
  { -1, 3, -9, 28, 118, -14, 4, -1 }, { -1, 3, -9, 25, 120, -13, 4, -1 },
  { -1, 3, -8, 22, 121, -12, 4, -1 }, { -1, 3, -7, 20, 122, -11, 3, -1 },
  { -1, 2, -6, 18, 123, -10, 3, -1 }, { 0, 2, -6, 15, 124, -9, 3, -1 },
  { 0, 2, -5, 13, 125, -8, 2, -1 },   { 0, 1, -4, 11, 125, -7, 2, 0 },
  { 0, 1, -3, 8, 126, -6, 2, 0 },     { 0, 1, -3, 6, 127, -4, 1, 0 },
  { 0, 1, -2, 4, 127, -3, 1, 0 },     { 0, 0, -1, 2, 128, -1, 0, 0 },
}

```

It is a requirement of bitstream conformance that upscaledPlaneW is strictly greater than downscaledPlaneW.

The output of this process is equal to outputFrame.

## 7.17. Loop restoration process

Input to this process are the arrays UpscaledCurrFrame (of reconstructed samples) and UpscaledCdefFrame (of deringed samples).

Output from this process is the array LrFrame of loop restored samples.

**Note:** Although this process loops over 4x4 blocks, loop restoration is designed to work in stripes 64 luma samples high without needing additional line buffers. Samples within the current stripe are fetched from UpscaledCdefFrame. Samples outside the current stripe are fetched from UpscaledCurrFrame (these samples will be deblocked, but will not have CDEF filtering applied).

The array LrFrame is set equal to a copy of UpscaledCdefFrame. (The contents of LrFrame will later be overwritten for blocks that require restoration filtering.)

If UsesLr is equal to 0, then the process returns immediately after performing this copy.

Otherwise, loop restoration is applied as follows:

```
for ( y = 0; y < FrameHeight; y += MI_SIZE ) {
  for ( x = 0; x < UpscaledWidth; x += MI_SIZE ) {
    for ( plane = 0; plane < NumPlanes; plane++ ) {
      if ( FrameRestorationType[ plane ] != RESTORE_NONE ) {
        row = y >> MI_SIZE_LOG2
        col = x >> MI_SIZE_LOG2
        loop_restore_block( plane, row, col )
      }
    }
  }
}
```

When loop\_restore\_block is called, the loop restore block process in [section 7.17.1](#) is invoked with plane, row, and col as inputs.

## 7.17.1. Loop restore block process

The inputs to this process are:

- a variable plane specifying whether the process is filtering Y, U, or V samples,
- variables row and col specifying the location of the block in units of 4x4 blocks in the upscaled luma plane.

The output of this process are samples in LrFrame[ plane ].

The variable lumaY is set equal to row \* MI\_SIZE.

The variable stripeNum (specifying the zero-based index of the current stripe) is set equal to (lumaY + 8) / 64.

**Note:** The stripes are offset upwards by 8 luma samples to make pipelined implementations more efficient. When a row of superblocks has been received, enough rows of deblocked output can be produced to allow loop restoration of the corresponding stripes.



The variables subX and subY are set equal to the subsampling for the current plane as follows:

- If plane is equal to 0, subX is set equal to 0 and subY is set equal to 0.
- Otherwise, subX is set equal to subsampling\_x and subY is set equal to subsampling\_y.

The variable StripeStartY (specifying the start of the stripe in units of samples in the current plane) is set equal to  $(-8 + \text{stripeNum} * 64) \gg \text{subY}$ .

The variable StripeEndY (specifying the end of the stripe in units of samples in the current plane) is set equal to  $\text{StripeStartY} + (64 \gg \text{subY}) - 1$ .

**Note:** StripeStartY and StripeEndY are used by the get source sample process to decide whether to fetch from UpscaledCurrFrame or UpscaledCdefFrame.

The variable unitSize (specifying the size of restoration units in units of samples in the current plane) is set equal to  $\text{LoopRestorationSize}[\text{plane}]$ .

The variable unitRows (specifying the number of restoration units down the frame) is set equal to  $\text{count\_units\_in\_frame}(\text{unitSize}, \text{Round2}(\text{FrameHeight}, \text{subY}))$ .

The variable unitCols (specifying the number of restoration units across the frame) is set equal to  $\text{count\_units\_in\_frame}(\text{unitSize}, \text{Round2}(\text{UpscaledWidth}, \text{subX}))$ .

**Note:** The number of restoration units in a frame can be different for chroma and luma.

The variable unitRow (specifying the vertical index of the current loop restoration unit) is set equal to  $\text{Min}(\text{unitRows} - 1, ((\text{row} * \text{MI\_SIZE} + 8) \gg \text{subY}) / \text{unitSize})$ .

The variable unitCol (specifying the horizontal index of the current loop restoration unit) is set equal to  $\text{Min}(\text{unitCols} - 1, (\text{col} * \text{MI\_SIZE} \gg \text{subX}) / \text{unitSize})$ .

The horizontal extent of the space allowed for filtering is specified as follows:

The variable PlaneEndX (specifying the horizontal extent of the space allowed for filtering) is set equal to  $\text{Round2}(\text{UpscaledWidth}, \text{subX}) - 1$ .

The variable PlaneEndY (specifying the vertical extent of the space allowed for filtering) is set equal to  $\text{Round2}(\text{FrameHeight}, \text{subY}) - 1$ .

The variable x is set equal to  $(\text{col} * \text{MI\_SIZE} \gg \text{subX})$ .

The variable y is set equal to  $(\text{row} * \text{MI\_SIZE} \gg \text{subY})$ .

The variable w is set equal to  $\text{Min}(\text{MI\_SIZE} \gg \text{subX}, \text{PlaneEndX} - x + 1)$ .

The variable  $h$  is set equal to  $\text{Min}(\text{MI\_SIZE} \gg \text{subY}, \text{PlaneEndY} - y + 1)$ .

(Variables  $x$  and  $y$  represent the position of the block in samples relative to the top-left corner of the current plane. Variables  $w$  and  $h$  represent the size of the block in samples.)

**Note:** Although the filter is described as operating on small blocks, the output will be the same if larger blocks are used - provided all contained samples belong to the same loop restoration unit.

The variable  $rType$  (specifying the loop restoration type) is set equal to  $\text{LrType}[\text{plane}][\text{unitRow}][\text{unitCol}]$ .

The filter to used depends on  $rType$  as follows:

- If  $rType$  is equal to `RESTORE_WIENER`, the Wiener filter process specified in [section 7.17.4](#) is invoked with  $\text{plane}$ ,  $\text{unitRow}$ ,  $\text{unitCol}$ ,  $x$ ,  $y$ ,  $w$ , and  $h$  as inputs.
- Otherwise, if  $rType$  is equal to `RESTORE_SGRPROJ`, the self guided filter process specified in [section 7.17.2](#) is invoked with  $\text{plane}$ ,  $\text{unitRow}$ ,  $\text{unitCol}$ ,  $x$ ,  $y$ ,  $w$ , and  $h$  as inputs.
- Otherwise ( $rType$  is equal to `RESTORE_NONE`), no filtering is applied.

## 7.17.2. Self guided filter process

The inputs to this block are:

- a variable  $\text{plane}$  specifying whether the process is filtering Y, U, or V samples,
- variables  $\text{unitRow}$  and  $\text{unitCol}$  specifying the position of the loop restoration unit,
- variables  $x$  and  $y$  specifying the position of the block in samples relative to the top-left corner of the current plane,
- variables  $w$  and  $h$  specifying the size of the block in samples.

The arrays  $\text{flt0}$  and  $\text{flt1}$  are prepared by the following ordered steps:

1. The variable  $\text{set}$  is set equal to  $\text{LrSgrSet}[\text{plane}][\text{unitRow}][\text{unitCol}]$ .
2. The variable  $\text{pass}$  is set equal to 0.
3. The box filter process specified in [section 7.17.3](#) is invoked with  $\text{plane}$ ,  $x$ ,  $y$ ,  $w$ ,  $h$ ,  $\text{set}$ , and  $\text{pass}$  as inputs, and the output assigned to  $\text{flt0}$ .
4. The variable  $\text{pass}$  is set equal to 1.
5. The box filter process specified in [section 7.17.3](#) is invoked with  $\text{plane}$ ,  $x$ ,  $y$ ,  $w$ ,  $h$ ,  $\text{set}$ , and  $\text{pass}$  as inputs, and the output assigned to  $\text{flt1}$ .

The restoration process is then applied for each sample as follows:

```

w0 = LrSgrXqd[ plane ][ unitRow ][ unitCol ][ 0 ]
w1 = LrSgrXqd[ plane ][ unitRow ][ unitCol ][ 1 ]
w2 = (1 << SGRPROJ_PRJ_BITS) - w0 - w1
r0 = Sgr_Params[ set ][ 0 ]
r1 = Sgr_Params[ set ][ 2 ]
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        u = UpscaledCdefFrame[ plane ][ y + i ][ x + j ] << SGRPROJ_RST_BITS
        v = w1 * u
        if ( r0 )
            v += w0 * flt0[ i ][ j ]
        else
            v += w0 * u
        if ( r1 )
            v += w2 * flt1[ i ][ j ]
        else
            v += w2 * u
        s = Round2( v, SGRPROJ_RST_BITS + SGRPROJ_PRJ_BITS )
        LrFrame[ plane ][ y + i ][ x + j ] = Clip1( s )
    }
}

```

### 7.17.3. Box filter process

The inputs to this process are:

- a variable plane specifying whether the process is filtering Y, U, or V samples,
- variables x and y specifying the position of the block in samples relative to the top-left corner of the current plane,
- variables w and h specifying the size of the block in samples,
- a variable set specifying the strength of the filtering,
- a variable pass (equal to 0 or 1), specifying if the process is generating the first or second filtered output.

The output of this process is a 2d array F.

The variable r (specifying that the box filters have side length  $2*r+1$ ) is set equal to `Sgr_Params[ set ][ pass * 2 + 0 ]`.

If r is equal to 0, then this process immediately terminates.

The variable eps (specifying a scaling for the output) is set equal to `Sgr_Params[ set ][ pass * 2 + 1 ]`.

The 2d arrays A and B (note these arrays are valid for coordinates including an extra sample around the boundary) are prepared as follows:

```

n = ( 2 * r + 1 ) * ( 2 * r + 1 )
n2e = n * n * eps
s = (((1 << SGRPROJ_MTABLE_BITS) + n2e / 2) / n2e)
for ( i = -1; i < h + 1; i++ ) {
    for ( j = -1; j < w + 1; j++ ) {
        a = 0
        b = 0
        for ( dy = -r; dy <= r; dy++ ) {
            for ( dx = -r; dx <= r; dx++ ) {
                c = get_source_sample( plane, x + j + dx, y + i + dy )
                a += c * c
                b += c
            }
        }
        a = Round2( a, 2 * (BitDepth - 8) )
        d = Round2( b, BitDepth - 8 )
        p = Max( 0, a * n - d * d )
        z = Round2( p * s, SGRPROJ_MTABLE_BITS )
        if ( z >= 255 )
            a2 = 256
        else if ( z == 0 )
            a2 = 1
        else
            a2 = ((z << SGRPROJ_SGR_BITS) + (z/2)) / (z + 1)
        oneOverN = ((1 << SGRPROJ_RECIP_BITS) + (n/2)) / n
        b2 = ( (1 << SGRPROJ_SGR_BITS) - a2 ) * b * oneOverN
        A[ i ][ j ] = a2
        B[ i ][ j ] = Round2( b2, SGRPROJ_RECIP_BITS )
    }
}

```

**Note:** When pass is equal to 0, only odd rows (i.e. entries  $A[i][j]$  and  $B[i][j]$  with  $i$  odd) will be used to generate the output.

where the call to `get_source_sample` specifies that the get source sample process specified in [section 7.17.6](#) should be invoked and the output assigned to variable `c`.

After `A` and `B` are prepared, the output array `F` is generated as follows:

```

for ( i = 0; i < h; i++ ) {
    shift = 5
    if ( pass == 0 && ( i & 1 ) ) {
        shift = 4
    }
    for ( j = 0; j < w; j++ ) {
        a = 0
        b = 0
        for ( dy = -1; dy <= 1; dy++ ) {
            for ( dx = -1; dx <= 1; dx++ ) {
                if ( pass == 0 ) {
                    if ( ( i + dy ) & 1 ) {
                        weight = ( dx == 0 ) ? 6 : 5
                    } else {
                        weight = 0
                    }
                } else {
                    weight = ( dx == 0 || dy == 0 ) ? 4 : 3
                }
                a += weight * A[ i + dy ][ j + dx ]
                b += weight * B[ i + dy ][ j + dx ]
            }
        }
        v = a * UpscaledCdefFrame[ plane ][ y + i ][ x + j ] + b
        F[ i ][ j ] = Round2( v, SGRPROJ_SGR_BITS + shift - SGRPROJ_RST_BITS )
    }
}

```

**Note:** When pass is equal to 0, the weights for even rows of A and B are always equal to 0.

The constant lookup table Sgr\_Params is specified as:

```

Sgr_Params[ ( 1 << SGRPROJ_PARAMS_BITS ) ][ 4 ] = {
    { 2, 12, 1, 4 }, { 2, 15, 1, 6 }, { 2, 18, 1, 8 }, { 2, 21, 1, 9 },
    { 2, 24, 1, 10 }, { 2, 29, 1, 11 }, { 2, 36, 1, 12 }, { 2, 45, 1, 13 },
    { 2, 56, 1, 14 }, { 2, 68, 1, 15 }, { 0, 0, 1, 5 }, { 0, 0, 1, 8 },
    { 0, 0, 1, 11 }, { 0, 0, 1, 14 }, { 2, 30, 0, 0 }, { 2, 75, 0, 0 }
}

```

## 7.17.4. Wiener filter process

The inputs to this block are:

- a variable plane specifying whether the process is filtering Y, U, or V samples,
- variables unitRow and unitCol specifying the position of the loop restoration unit,

- variables x and y specifying the position of the block in samples relative to the top-left corner of the current plane,
- variables w and h specifying the size of the block in samples.

The output from this process are modified samples in LrFrame.

The sub-sample interpolation is effected via two one-dimensional convolutions. First a horizontal filter is used to build up a temporary array, and then this array is vertically filtered to obtain the final prediction.

The rounding variables derivation process specified in [section 7.11.3.2](#) is invoked with the input variable isCompound set equal to 0.

The Wiener coefficient process specified in [section 7.17.5](#) is invoked with an input of LrWiener[ plane ][ unitRow ][ unitCol ][ 0 ] and the output assigned to vfilter.

The Wiener coefficient process specified in [section 7.17.5](#) is invoked with an input of LrWiener[ plane ][ unitRow ][ unitCol ][ 1 ] and the output assigned to hfilter.

**Note:** The horizontal filter needs to be applied before the vertical filter, but the horizontal coefficients are sent after the vertical coefficients.

The filtering is applied as follows:

- The array intermediate is specified as follows:

```
offset = (1 << (BitDepth + FILTER_BITS - InterRound0 - 1))
limit = (1 << (BitDepth + 1 + FILTER_BITS - InterRound0)) - 1
for ( r = 0; r < h + 7; r++ ) {
    for ( c = 0; c < w; c++ ) {
        s = 0
        for ( t = 0; t < 7; t++ )
            s += hfilter[ t ] * get_source_sample( plane, x + c + t - 3, y + r - 3 )
        v = Round2(s, InterRound0)
        intermediate[ r ][ c ] = Clip3( -offset, limit - offset, v )
    }
}
```

Where the call to get\_source\_sample specifies that the get source sample process specified in [section 7.17.6](#) should be invoked.

**Note:** The intermediate result is clipped so that ( intermediate[ r ][ c ] + offset ) fits in an unsigned variable with (BitDepth + 3) bits.

- The output samples are written as follows:

```

for ( r = 0; r < h; r++ ) {
    for ( c = 0; c < w; c++ ) {
        s = 0
        for ( t = 0; t < 7; t++ )
            s += vfilter[ t ] * intermediate[ r + t ][ c ]
        v = Round2( s, InterRound1 )
        LrFrame[ plane ][ y + r ][ x + c ] = Clip1( v )
    }
}

```

## 7.17.5. Wiener coefficient process

The input to this process is an array `coeff` containing 3 coefficients.

The output from this process is an array containing 7 coefficients.

The Wiener filter is always symmetrical and has a unit DC gain, so there are only three coefficients that need to be explicitly coded.

This process computes the full set of coefficients as follows:

```

filter[ 3 ] = 128
for ( i = 0; i < 3; i++ ) {
    c = coeff[ i ]
    filter[ i ] = c
    filter[ 6 - i ] = c
    filter[ 3 ] -= 2 * c
}

```

The output of the process is the array `filter`.

**Note:** When chroma is being filtered, `coeff[ 0 ]` will always be equal to 0, therefore `filter[ 0 ]` and `filter[ 6 ]` will always be equal to 0. In other words, luma uses a 7-tap filter, while chroma uses a 5-tap filter.

## 7.17.6. Get source sample process

The inputs to this process are:

- a variable `plane` specifying whether the process is filtering Y, U, or V samples,
- variables `x` and `y` specifying the location in the current plane in units of samples.

This process makes sure samples are taken from within the allowed extent for loop restoration filtering.

Samples within the current stripe are taken after Cdef filtering has been applied, samples outside the current stripe are taken before Cdef filtering.

The sample to return is specified as follows:

```
x = Min(PlaneEndX, x)
x = Max(0, x)
y = Min(PlaneEndY, y)
y = Max(0, y)
if ( y < StripeStartY ) {
    y = Max(StripeStartY - 2,y)
    return UpscaledCurrFrame[ plane ][ y ][ x ]
} else if ( y > StripeEndY ) {
    y = Min(StripeEndY + 2,y)
    return UpscaledCurrFrame[ plane ][ y ][ x ]
} else {
    return UpscaledCdefFrame[ plane ][ y ][ x ]
}
```

**Note:** This process can be called for samples on the three lines above and three lines below the current stripe. However, the coordinates are cropped such that only two lines above and below the stripe need to be fetched. In other words, requests for the third line (above or below) are given a copy of the second line.

## 7.18. Output process

### 7.18.1. General

This process is invoked to prepare output frames.

If scalability is being used (OperatingPointIdx not equal to 0), an application-specific function is called to decide whether this frame will be output. If this function returns a value equal to 0, then this process terminates immediately.

**Note:** Applications that are displaying the decoded video are expected to only display one frame from each temporal unit within the selected operating point. This frame should be the highest spatial layer that is both within the operating point and present within the temporal unit. Other applications may set their own policy about which frames are output.

The arrays OutY, OutU, and OutV are prepared and then film grain noise is added if required.

If show\_existing\_frame is equal to 1, then the decoder should copy OutY, OutU, and OutV from a previously decoded frame as follows:

- The variable w is set equal to RefUpscaledWidth[ frame\_to\_show\_map\_idx ].
- The variable h is set equal to RefFrameHeight[ frame\_to\_show\_map\_idx ].



- The variable subX is set equal to RefSubsamplingX[ frame\_to\_show\_map\_idx ].
- The variable subY is set equal to RefSubsamplingY[ frame\_to\_show\_map\_idx ].
- The array OutY is w samples across by h samples down and the sample at location x samples across and y samples down is given by  $\text{OutY}[y][x] = \text{FrameStore}[\text{frame\_to\_show\_map\_idx}][0][y][x]$  with  $x = 0..w - 1$  and  $y = 0..h - 1$ .
- The array OutU is  $(w + \text{subX}) \gg \text{subX}$  samples across by  $(h + \text{subY}) \gg \text{subY}$  samples down and the sample at location x samples across and y samples down is given by  $\text{OutU}[y][x] = \text{FrameStore}[\text{frame\_to\_show\_map\_idx}][1][y][x]$  with  $x = 0..((w + \text{subX}) \gg \text{subX}) - 1$  and  $y = 0..((h + \text{subY}) \gg \text{subY}) - 1$ .
- The array OutV is  $(w + \text{subX}) \gg \text{subX}$  samples across by  $(h + \text{subY}) \gg \text{subY}$  samples down and the sample at location x samples across and y samples down is given by  $\text{OutV}[y][x] = \text{FrameStore}[\text{frame\_to\_show\_map\_idx}][2][y][x]$  with  $x = 0..((w + \text{subX}) \gg \text{subX}) - 1$  and  $y = 0..((h + \text{subY}) \gg \text{subY}) - 1$ .
- The variable BitDepth is set equal to RefBitDepth[ frame\_to\_show\_map\_idx ].
- The bit depth for each sample is BitDepth.

Otherwise (show\_existing\_frame is equal to 0), then the decoder should copy the current frame as follows:

- The variable w is set equal to UpscaledWidth.
- The variable h is set equal to FrameHeight.
- The variable subX is set equal to subsampling\_x.
- The variable subY is set equal to subsampling\_y.
- The array OutY is w samples across by h samples down and the sample at location x samples across and y samples down is given by  $\text{OutY}[y][x] = \text{LrFrame}[0][y][x]$  with  $x = 0..w - 1$  and  $y = 0..h - 1$ .
- The array OutU is  $(w + \text{subX}) \gg \text{subX}$  samples across by  $(h + \text{subY}) \gg \text{subY}$  samples down and the sample at location x samples across and y samples down is given by  $\text{OutU}[y][x] = \text{LrFrame}[1][y][x]$  with  $x = 0..((w + \text{subX}) \gg \text{subX}) - 1$  and  $y = 0..((h + \text{subY}) \gg \text{subY}) - 1$ .
- The array OutV is  $(w + \text{subX}) \gg \text{subX}$  samples across by  $(h + \text{subY}) \gg \text{subY}$  samples down and the sample at location x samples across and y samples down is given by  $\text{OutV}[y][x] = \text{LrFrame}[2][y][x]$  with  $x = 0..((w + \text{subX}) \gg \text{subX}) - 1$  and  $y = 0..((h + \text{subY}) \gg \text{subY}) - 1$ .
- The bit depth for each sample is BitDepth.

If film\_grain\_params\_present is equal to 1 and apply\_grain is equal to 1, then the film grain synthesis process specified in [section 7.18.2](#) is invoked with inputs of w, h, subX, and subY. (This process modifies the output arrays OutY, OutU, OutV).

Finally, the frame to be output is defined to be the arrays OutY, OutU, OutV where the bit depth for each sample is BitDepth.

This frame to be output is the overall output of the decoding process and further processing (such as color conversion) is outside the scope of this specification.

For example, a real implementation might use these arrays to display the frame to the user, or a test system might save the arrays so the output can be verified.

**Note:** If NumPlanes is equal to 1, then the U and V planes should be ignored.

## 7.18.2. Film grain synthesis process

### 7.18.2.1. General

The inputs to this process are:

- variables w and h specifying the width and height of the frame,
- variables subX and subY specifying the subsampling parameters of the frame.

The process modifies the arrays OutY, OutU, OutV to add film grain noise as follows:

1. The variable RandomRegister (used for generating pseudo-random numbers) is set equal to grain\_seed.
2. The variable GrainCenter is set equal to  $128 \ll (\text{BitDepth} - 8)$ .
3. The variable GrainMin is set equal to  $- \text{GrainCenter}$ .
4. The variable GrainMax is set equal to  $(256 \ll (\text{BitDepth} - 8)) - 1 - \text{GrainCenter}$ .
5. The generate grain process specified in [section 7.18.2.3](#) is invoked.
6. The scaling lookup initialization process specified in [section 7.18.2.4](#) is invoked.
7. The add noise process specified in [section 7.18.2.5](#) is invoked with w, h, subX, and subY as inputs.

### 7.18.2.2. Random number process

The input to this process is a variable bits specifying the number of random bits to return.

The output of this process is a pseudo-random number based on the state in RandomRegister.

The process is specified as follows:

```

get_random_number( bits ) {
    r = RandomRegister
    bit = ((r >> 0) ^ (r >> 1) ^ (r >> 3) ^ (r >> 12)) & 1
    r = (r >> 1) | (bit << 15)
    result = (r >> (16 - bits)) & ((1 << bits) - 1)
    RandomRegister = r
    return result
}

```

The output of this process is the variable result.

### 7.18.2.3. Generate grain process

This process generates noise via an auto-regressive filter.

First an array LumaGrain 82 samples wide and 73 samples high of white noise is generated for luma as follows:

```

shift = 12 - BitDepth + grain_scale_shift
for ( y = 0; y < 73; y++ ) {
    for ( x = 0; x < 82; x++ ) {
        if ( num_y_points > 0 ) {
            g = Gaussian_Sequence[ get_random_number( 11 ) ]
        } else {
            g = 0
        }
        LumaGrain[ y ][ x ] = Round2( g, shift )
    }
}

```

where the function call `get_random_number` invokes the random number process specified in [section 7.18.2.2](#).

Then an auto-regressive filter is applied to the white noise as follows:

```

shift = ar_coeff_shift_minus_6 + 6
for ( y = 3; y < 73; y++ ) {
  for ( x = 3; x < 82 - 3; x++ ) {
    s = 0
    pos = 0
    for ( deltaRow = -ar_coeff_lag; deltaRow <= 0; deltaRow++ ) {
      for ( deltaCol = -ar_coeff_lag; deltaCol <= ar_coeff_lag; deltaCol++ ) {
        if ( deltaRow == 0 && deltaCol == 0 )
          break
        c = ar_coeffs_y_plus_128[ pos ] - 128
        s += LumaGrain[ y + deltaRow ][ x + deltaCol ] * c
        pos++
      }
    }
    LumaGrain[ y ][ x ] = Clip3( GrainMin, GrainMax, LumaGrain[ y ][ x ] + Round2( s, shift ) )
  }
}

```

If `mono_chrome` is equal to 0, the chroma grain is generated in a similar way, except the filtering includes a coefficient that introduces a correlation with the luma grain.

The variable `chromaW` (representing the width of the chroma noise array) is set equal to `(subsampling_x ? 44 : 82)`.

The variable `chromaH` (representing the height of the chroma noise array) is set equal to `(subsampling_y ? 38 : 73)`.

White noise arrays `CbGrain` and `CrGrain` `chromaW` samples wide and `chromaH` samples high are generated as follows:

```

shift = 12 - BitDepth + grain_scale_shift
RandomRegister = grain_seed ^ 0xb524
for ( y = 0; y < chromaH; y++ ) {
    for ( x = 0; x < chromaW; x++ ) {
        if ( num_cb_points > 0 || chroma_scaling_from_luma ) {
            g = Gaussian_Sequence[ get_random_number( 11 ) ]
        } else {
            g = 0
        }
        CbGrain[ y ][ x ] = Round2( g, shift )
    }
}
RandomRegister = grain_seed ^ 0x49d8
for ( y = 0; y < chromaH; y++ ) {
    for ( x = 0; x < chromaW; x++ ) {
        if ( num_cr_points > 0 || chroma_scaling_from_luma ) {
            g = Gaussian_Sequence[ get_random_number( 11 ) ]
        } else {
            g = 0
        }
        CrGrain[ y ][ x ] = Round2( g, shift )
    }
}

```

Then the auto-regressive filter is applied as follows:

```

shift = ar_coeff_shift_minus_6 + 6
for ( y = 3; y < chromaH; y++ ) {
  for ( x = 3; x < chromaW - 3; x++ ) {
    s0 = 0
    s1 = 0
    pos = 0
    for ( deltaRow = -ar_coeff_lag; deltaRow <= 0; deltaRow++ ) {
      for ( deltaCol = -ar_coeff_lag; deltaCol <= ar_coeff_lag; deltaCol++ ) {
        c0 = ar_coeffs_cb_plus_128[ pos ] - 128
        c1 = ar_coeffs_cr_plus_128[ pos ] - 128
        if ( deltaRow == 0 && deltaCol == 0 ) {
          if ( num_y_points > 0 ) {
            luma = 0
            lumaX = ( (x - 3) << subsampling_x ) + 3
            lumaY = ( (y - 3) << subsampling_y ) + 3
            for ( i = 0; i <= subsampling_y; i++ )
              for ( j = 0; j <= subsampling_x; j++ )
                luma += LumaGrain[ lumaY + i ][ lumaX + j ]
            luma = Round2( luma, subsampling_x + subsampling_y )
            s0 += luma * c0
            s1 += luma * c1
          }
          break
        }
        s0 += CbGrain[ y + deltaRow ][ x + deltaCol ] * c0
        s1 += CrGrain[ y + deltaRow ][ x + deltaCol ] * c1
        pos++
      }
    }
    CbGrain[ y ][ x ] = Clip3( GrainMin, GrainMax, CbGrain[ y ][ x ] + Round2( s0, shift ) )
    CrGrain[ y ][ x ] = Clip3( GrainMin, GrainMax, CrGrain[ y ][ x ] + Round2( s1, shift ) )
  }
}

```

**Note:** When `num_y_points` is equal to 0, this process may use uninitialized values within `ar_coeffs_y_plus_128` to compute `LumaGrain`. However, `LumaGrain` will never be read in this case so it does not matter what values are constructed. Similarly, when `num_cr_points/num_cb_points` are equal to 0 and `chroma_scaling_from_luma` is equal to 0, the `CbGrain/CrGrain` arrays will never be read.

#### 7.18.2.4. Scaling lookup initialization process

This process computes 3 lookup tables for the different color components.

Each lookup table `ScalingLut[ plane ]` contains 256 entries constructed by a piecewise linear interpolation of the given points as follows:

```

for ( plane = 0; plane < NumPlanes; plane++ ) {
    if ( plane == 0 || chroma_scaling_from_luma )
        numPoints = num_y_points
    else if ( plane == 1 )
        numPoints = num_cb_points
    else
        numPoints = num_cr_points
    if ( numPoints == 0 ) {
        for ( x = 0; x < 256; x++ ) {
            ScalingLut[ plane ][ x ] = 0
        }
    } else {
        for ( x = 0; x < get_x( plane, 0 ); x++ ) {
            ScalingLut[ plane ][ x ] = get_y( plane, 0 )
        }
        for ( i = 0; i < numPoints - 1; i++ ) {
            deltaY = get_y( plane, i + 1 ) - get_y( plane, i )
            deltaX = get_x( plane, i + 1 ) - get_x( plane, i )
            delta = deltaY * ( ( 65536 + (deltaX >> 1) ) / deltaX )
            for ( x = 0; x < deltaX; x++ ) {
                v = get_y( plane, i ) + ( ( x * delta + 32768 ) >> 16 )
                ScalingLut[ plane ][ get_x( plane, i ) + x ] = v
            }
        }
        for ( x = get_x( plane, numPoints - 1 ); x < 256; x++ ) {
            ScalingLut[ plane ][ x ] = get_y( plane, numPoints - 1 )
        }
    }
}

```

where the functions `get_x` and `get_y` return the coordinates for a specific point and are specified as:

```

get_x( plane, i ) {
    if ( plane == 0 || chroma_scaling_from_luma )
        return point_y_value[ i ]
    else if ( plane == 1 )
        return point_cb_value[ i ]
    else
        return point_cr_value[ i ]
}

get_y( plane, i ) {
    if ( plane == 0 || chroma_scaling_from_luma )
        return point_y_scaling[ i ]
    else if ( plane == 1 )
        return point_cb_scaling[ i ]
    else
        return point_cr_scaling[ i ]
}

```

### 7.18.2.5. Add noise synthesis process

The inputs to this process are:

- variables *w* and *h* specifying the width and height of the frame,
- variables *subX* and *subY* specifying the subsampling parameters of the frame.

This process combines the film grain with the image data.

First an array of noise data *noiseStripe* is generated for each 32 luma sample high stripe of the image.

*noiseStripe*[ *lumaNum* ][ 0 ] is 34 samples high and *w* samples wide (a few additional samples across are actually written to the array, but these are never read) and contains noise for the luma component.

*noiseStripe*[ *lumaNum* ][ 1 ] and *noiseStripe*[ *lumaNum* ][ 2 ] are (34 >> *subY*) samples high and Round2(*w*, *subX*) samples wide and contain noise for the chroma components.

*noiseStripe* represents the result of constructing square grain blocks and blending horizontally adjacent blocks together (although blending is only applied if *overlap\_flag* is equal to 1) and is constructed as follows:



```

lumaNum = 0
for ( y = 0; y < (h + 1)/2 ; y += 16 ) {
    RandomRegister = grain_seed
    RandomRegister ^= ((lumaNum * 37 + 178) & 255) << 8
    RandomRegister ^= ((lumaNum * 173 + 105) & 255)
    for ( x = 0; x < (w + 1)/2 ; x += 16 ) {
        rand = get_random_number( 8 )
        offsetX = rand >> 4
        offsetY = rand & 15
        for ( plane = 0 ; plane < NumPlanes; plane++ ) {
            planeSubX = ( plane > 0 ) ? subX : 0
            planeSubY = ( plane > 0 ) ? subY : 0
            planeOffsetX = planeSubX ? 6 + offsetX : 9 + offsetX * 2
            planeOffsetY = planeSubY ? 6 + offsetY : 9 + offsetY * 2
            for ( i = 0; i < 34 >> planeSubY ; i++ ) {
                for ( j = 0; j < 34 >> planeSubX ; j++ ) {
                    if ( plane == 0 )
                        g = LumaGrain[ planeOffsetY + i ][ planeOffsetX + j ]
                    else if ( plane == 1 )
                        g = CbGrain[ planeOffsetY + i ][ planeOffsetX + j ]
                    else
                        g = CrGrain[ planeOffsetY + i ][ planeOffsetX + j ]
                    if ( planeSubX == 0 ) {
                        if ( j < 2 && overlap_flag && x > 0 ) {
                            old = noiseStripe[ lumaNum ][ plane ][ i ][ x * 2 + j ]
                            if ( j == 0 ) {
                                g = old * 27 + g * 17
                            } else {
                                g = old * 17 + g * 27
                            }
                        }
                        g = Clip3( GrainMin, GrainMax, Round2(g, 5) )
                    }
                    noiseStripe[ lumaNum ][ plane ][ i ][ x * 2 + j ] = g
                } else {
                    if ( j == 0 && overlap_flag && x > 0 ) {
                        old = noiseStripe[ lumaNum ][ plane ][ i ][ x + j ]
                        g = old * 23 + g * 22
                        g = Clip3( GrainMin, GrainMax, Round2(g, 5) )
                    }
                    noiseStripe[ lumaNum ][ plane ][ i ][ x + j ] = g
                }
            }
        }
    }
}
lumaNum++
}

```

Then the noise stripes are blended together to form a noise image noiseImage as follows:

```

for ( plane = 0; plane < NumPlanes; plane++ ) {
  planeSubX = ( plane > 0 ) ? subX : 0
  planeSubY = ( plane > 0 ) ? subY : 0
  for ( y = 0; y < ( h + planeSubY ) >> planeSubY ; y++ ) {
    lumaNum = y >> ( 5 - planeSubY )
    i = y - ( lumaNum << ( 5 - planeSubY ) )
    for ( x = 0; x < ( w + planeSubX ) >> planeSubX ; x++ ) {
      g = noiseStripe[ lumaNum ][ plane ][ i ][ x ]
      if ( planeSubY == 0 ) {
        if ( i < 2 && lumaNum > 0 && overlap_flag ) {
          old = noiseStripe[ lumaNum - 1 ][ plane ][ i + 32 ][ x ]
          if ( i == 0 ) {
            g = old * 27 + g * 17
          } else {
            g = old * 17 + g * 27
          }
        }
        g = Clip3( GrainMin, GrainMax, Round2(g, 5) )
      }
    } else {
      if ( i < 1 && lumaNum > 0 && overlap_flag ) {
        old = noiseStripe[ lumaNum - 1 ][ plane ][ i + 16 ][ x ]
        g = old * 23 + g * 22
        g = Clip3( GrainMin, GrainMax, Round2(g, 5) )
      }
    }
    noiseImage[ plane ][ y ][ x ] = g
  }
}
}

```

**Note:** Although this process is specified in terms of full size noiseStripe and noiseImage arrays, the reference code shows how it is possible to implement the grain synthesis with just 2 line buffers for luma, and 1 line buffer for each chroma component.

Finally, the noise is blended with the original image data as follows:

```

if ( clip_to_restricted_range ) {
    minValue = 16 << (BitDepth - 8)
    maxLuma = 235 << (BitDepth - 8)
    if ( matrix_coefficients == MC_IDENTITY )
        maxChroma = maxLuma
    else
        maxChroma = 240 << (BitDepth - 8)
} else {
    minValue = 0
    maxLuma = (256 << (BitDepth - 8)) - 1
    maxChroma = maxLuma
}
ScalingShift = grain_scaling_minus_8 + 8
for ( y = 0; y < ( (h + subY) >> subY) ; y++ ) {
    for ( x = 0; x < ( (w + subX) >> subX) ; x++ ) {
        lumaX = x << subX
        lumaY = y << subY
        lumaNextX = Min( lumaX + 1, w - 1 )
        if ( subX )
            averageLuma = Round2( OutY[ lumaY ][ lumaX ] + OutY[ lumaY ][ lumaNextX ], 1 )
        else
            averageLuma = OutY[ lumaY ][ lumaX ]
        if ( num_cb_points > 0 || chroma_scaling_from_luma ) {
            orig = OutU[ y ][ x ]
            if ( chroma_scaling_from_luma ) {
                merged = averageLuma
            } else {
                combined = averageLuma * ( cb_luma_mult - 128 ) + orig * ( cb_mult - 128 )
                merged = Clip1( ( combined >> 6 ) + ( (cb_offset - 256) << (BitDepth - 8) ) )
            }
            noise = noiseImage[ 1 ][ y ][ x ]
            noise = Round2( scale_lut( 1, merged ) * noise, ScalingShift )
            OutU[ y ][ x ] = Clip3( minValue, maxChroma, orig + noise )
        }

        if ( num_cr_points > 0 || chroma_scaling_from_luma ) {
            orig = OutV[ y ][ x ]
            if ( chroma_scaling_from_luma ) {
                merged = averageLuma
            } else {
                combined = averageLuma * ( cr_luma_mult - 128 ) + orig * ( cr_mult - 128 )
                merged = Clip1( ( combined >> 6 ) + ( (cr_offset - 256) << (BitDepth - 8) ) )
            }
            noise = noiseImage[ 2 ][ y ][ x ]
            noise = Round2( scale_lut( 2, merged ) * noise, ScalingShift )
            OutV[ y ][ x ] = Clip3( minValue, maxChroma, orig + noise )
        }
    }
}
for ( y = 0; y < h ; y++ ) {

```

```

for ( x = 0; x < w ; x++ ) {
    orig = OutY[ y ][ x ]
    noise = noiseImage[ 0 ][ y ][ x ]
    noise = Round2( scale_lut( 0, orig ) * noise, ScalingShift )
    if ( num_y_points > 0 ) {
        OutY[ y ][ x ] = Clip3( minValue, maxLuma, orig + noise )
    }
}
}

```

where `scale_lut` is a function that performs a piecewise linear interpolation into the appropriate scaling table. The `scale_lut` function is specified as follows:

```

scale_lut( plane, index ) {
    shift = BitDepth - 8
    x = index >> shift
    rem = index - ( x << shift )
    if ( BitDepth == 8 || x == 255 ) {
        return ScalingLut[ plane ][ x ]
    } else {
        start = ScalingLut[ plane ][ x ]
        end = ScalingLut[ plane ][ x + 1 ]
        return start + Round2( (end - start) * rem, shift )
    }
}

```

## 7.19. Motion field motion vector storage process

This process applies some filtering and reordering to the motion vectors to prepare them for storage as part of the reference frame update process.

The following applies for `row = 0..MiRows-1`, for `col = 0..MiCols-1`:

```

MfRefFrames[ row ][ col ] = NONE
MfMvs[ row ][ col ][ 0 ] = 0
MfMvs[ row ][ col ][ 1 ] = 0
for ( list = 0; list < 2; list++ ) {
    r = RefFrames[ row ][ col ][ list ]
    if ( r > INTRA_FRAME ) {
        refIdx = ref_frame_idx[ r - LAST_FRAME ]
        dist = get_relative_dist( RefOrderHint[ refIdx ], OrderHint )
        if ( dist < 0 ) {
            mvRow = Mvs[ row ][ col ][ list ][ 0 ]
            mvCol = Mvs[ row ][ col ][ list ][ 1 ]
            if ( Abs( mvRow ) <= REFMVS_LIMIT && Abs( mvCol ) <= REFMVS_LIMIT ) {
                MfRefFrames[ row ][ col ] = r
                MfMvs[ row ][ col ][ 0 ] = mvRow
                MfMvs[ row ][ col ][ 1 ] = mvCol
            }
        }
    }
}

```

**Note:** Although this process stores all the motion vectors into MfMvs, only the values where row and col are both odd will affect the decoding process.

## 7.20. Reference frame update process

This process is invoked as the final step in decoding a frame.

The inputs to this process are the decoded samples for the current frame LrFrame[ plane ][ x ][ y ].

The output from this process is an updated set of reference frames and previous motion vectors.

For each value of *i* from 0 to NUM\_REF\_FRAMES - 1, the following applies if bit *i* of refresh\_frame\_flags is equal to 1 (i.e. if (refresh\_frame\_flags >> i) & 1 is equal to 1):

- RefValid[ *i* ] is set equal to 1.
- RefFrameId[ *i* ] is set equal to current\_frame\_id.
- RefUpscaledWidth[ *i* ] is set equal to UpscaledWidth.
- RefFrameWidth[ *i* ] is set equal to FrameWidth.
- RefFrameHeight[ *i* ] is set equal to FrameHeight.
- RefRenderWidth[ *i* ] is set equal to RenderWidth.
- RefRenderHeight[ *i* ] is set equal to RenderHeight.

- `RefMiCols[ i ]` is set equal to `MiCols`.
- `RefMiRows[ i ]` is set equal to `MiRows`.
- `RefFrameType[ i ]` is set equal to `frame_type`.
- `RefSubsamplingX[ i ]` is set equal to `subsampling_x`.
- `RefSubsamplingY[ i ]` is set equal to `subsampling_y`.
- `RefBitDepth[ i ]` is set equal to `BitDepth`.
- `SavedOrderHints[ i ][ j + LAST_FRAME ]` is set equal to `OrderHints[ j + LAST_FRAME ]` for `j = 0..REFS_PER_FRAME-1`.
- `FrameStore[ i ][ 0 ][ y ][ x ]` is set equal to `LrFrame[ 0 ][ y ][ x ]` for `x = 0..UpscaledWidth-1`, for `y = 0..FrameHeight-1`.
- `FrameStore[ i ][ plane ][ y ][ x ]` is set equal to `LrFrame[ plane ][ y ][ x ]` for `plane = 1..2`, for `x = 0..((UpscaledWidth + subsampling_x) >> subsampling_x) - 1`, for `y = 0..((FrameHeight + subsampling_y) >> subsampling_y) - 1`.
- `SavedRefFrames[ i ][ row ][ col ]` is set equal to `MfRefFrames[ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- `SavedMvs[ i ][ row ][ col ][ comp ]` is set equal to `MfMvs[ row ][ col ][ comp ]` for `comp = 0..1`, for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- `SavedGmParams[ i ][ ref ][ j ]` is set equal to `gm_params[ ref ][ j ]` for `ref = LAST_FRAME..ALTREF_FRAME`, for `j = 0..5`.
- `SavedSegmentIds[ i ][ row ][ col ]` is set equal to `SegmentIds[ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- The function `save_cdfs( i )` is invoked (see below).
- If `film_grain_params_present` is equal to 1, the function `save_grain_params( i )` is invoked (see below).
- The function `save_loop_filter_params( i )` is invoked (see below).
- The function `save_segmentation_params( i )` is invoked (see below).

For each value of `i` from 0 to `NUM_REF_FRAMES - 1`, the following applies if bit `i` of `refresh_frame_flags` is equal to 1 (i.e. if `(refresh_frame_flags >> i) & 1` is equal to 1):

- `RefOrderHint[ i ]` is set equal to `OrderHint`.

`save_cdfs( ctx )` is a function call that indicates that all the CDF arrays are saved into frame context number `ctx` in the range 0 to `(NUM_REF_FRAMES - 1)`. When this function is invoked the following takes place:

- A copy of each CDF array mentioned in the semantics for `init_coeff_cdfs` and `init_non_coeff_cdfs` is saved in an area of memory indexed by `ctx`.

`save_grain_params( i )` is a function call that indicates that all the syntax elements that can be read in `film_grain_params` should be saved into an area of memory indexed by `i`.

`save_loop_filter_params( i )` is a function call that indicates that the values of `loop_filter_ref_deltas[ j ]` for `j = 0 .. TOTAL_REFS_PER_FRAME-1`, and the values of `loop_filter_mode_deltas[ j ]` for `j = 0 .. 1` should be saved into an area of memory indexed by `i`.

`save_segmentation_params( i )` is a function call that indicates that the values of `FeatureEnabled[ j ][ k ]` and `FeatureData[ j ][ k ]` for `j = 0 .. MAX_SEGMENTS-1`, for `k = 0 .. SEG_LVL_MAX-1` should be saved into an area of memory indexed by `i`.

**Note:** Although this process stores all the motion vectors into `SavedMvs`, only the values where `row` and `col` are both odd will affect the decoding process.

## 7.21. Reference frame loading process

This process is the reverse of the reference frame update process specified in [section 7.20](#). It loads saved values for a previous reference frame back into the current frame variables. The index of the saved reference frame to load is given by the syntax element `frame_to_show_map_idx`.

- `current_frame_id` is set equal to `RefFrameId[ i ]`.
- `UpscaledWidth` is set equal to `RefUpscaledWidth[ frame_to_show_map_idx ]`.
- `FrameWidth` is set equal to `RefFrameWidth[ frame_to_show_map_idx ]`.
- `FrameHeight` is set equal to `RefFrameHeight[ frame_to_show_map_idx ]`.
- `RenderWidth` is set equal to `RefRenderWidth[ frame_to_show_map_idx ]`.
- `RenderHeight` is set equal to `RefRenderHeight[ frame_to_show_map_idx ]`.
- `MiCols` is set equal to `RefMiCols[ frame_to_show_map_idx ]`.
- `MiRows` is set equal to `RefMiRows[ frame_to_show_map_idx ]`.
- `subsampling_x` is set equal to `RefSubsamplingX[ frame_to_show_map_idx ]`.
- `subsampling_y` is set equal to `RefSubsamplingY[ frame_to_show_map_idx ]`.
- `BitDepth` is set equal to `RefBitDepth[ frame_to_show_map_idx ]`.
- `OrderHint` is set equal to `RefOrderHint[ frame_to_show_map_idx ]`.

- `OrderHints[ j + LAST_FRAME ]` is set equal to `SavedOrderHints[ frame_to_show_map_idx ][ j + LAST_FRAME ]` for `j = 0..REFS_PER_FRAME-1`.
- `LrFrame[ 0 ][ y ][ x ]` is set equal to `FrameStore[ frame_to_show_map_idx ][ 0 ][ y ][ x ]` for `x = 0..UpscaledWidth-1`, for `y = 0..FrameHeight-1`.
- `LrFrame[ plane ][ y ][ x ]` is set equal to `FrameStore[ frame_to_show_map_idx ][ plane ][ y ][ x ]` for `plane = 1..2`, for `x = 0..((UpscaledWidth + subsampling_x) >> subsampling_x) - 1`, for `y = 0..((FrameHeight + subsampling_y) >> subsampling_y) - 1`.
- `MfRefFrames[ row ][ col ]` is set equal to `SavedRefFrames[ frame_to_show_map_idx ][ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- `MfMvs[ row ][ col ][ comp ]` is set equal to `SavedMvs[ frame_to_show_map_idx ][ row ][ col ][ comp ]` for `comp = 0..1`, for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- `gm_params[ ref ][ j ]` is set equal to `SavedGmParams[ frame_to_show_map_idx ][ ref ][ j ]` for `ref = LAST_FRAME..ALTREF_FRAME`, for `j = 0..5`.
- `SegmentIds[ row ][ col ]` is set equal to `SavedSegmentIds[ frame_to_show_map_idx ][ row ][ col ]` for `row = 0..MiRows-1`, for `col = 0..MiCols-1`.
- The function `load_cdfs( frame_to_show_map_idx )` is invoked.
- If `film_grain_params_present` is equal to 1, the function `load_grain_params( frame_to_show_map_idx )` is invoked (see [section 6.8.20](#)).
- The function `load_loop_filter_params( frame_to_show_map_idx )` is invoked (see below).
- The function `load_segmentation_params( frame_to_show_map_idx )` is invoked (see below).

`load_loop_filter_params( i )` is a function call that indicates that the values of `loop_filter_ref_deltas[ j ]` for `j = 0 .. TOTAL_REFS_PER_FRAME-1`, and the values of `loop_filter_mode_deltas[ j ]` for `j = 0 .. 1` should be loaded from an area of memory indexed by `i`.

`load_segmentation_params( i )` is a function call that indicates that the values of `FeatureEnabled[ j ][ k ]` and `FeatureData[ j ][ k ]` for `j = 0 .. MAX_SEGMENTS-1`, for `k = 0 .. SEG_LVL_MAX-1` should be loaded from an area of memory indexed by `i`.



## 8. Parsing process

### 8.1. Parsing process for f(n)

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to f(n).

The next n bits are read from the bit stream.

This process is specified as follows:

```
x = 0
for ( i = 0; i < n; i++ ) {
    x = 2 * x + read_bit( )
}
```

read\_bit( ) reads the next bit from the bitstream and advances the bitstream position indicator by 1. If the bitstream is provided as a series of bytes, then the first bit is given by the most significant bit of the first byte.

The value for the syntax element is given by x.

## 8.2. Parsing process for symbol decoder

### 8.2.1. General

Aside from the uncompressed header and the partition sizes, the entire bitstream is entropy coded. The entropy decoder is referred to as the “Symbol decoder” and the functions init\_symbol( sz ), exit\_symbol( ), read\_symbol( cdf ), and read\_bool( ) are used in this Specification to indicate the entropy decoding operation.

### 8.2.2. Initialization process for symbol decoder

The input to this process is a variable sz specifying the number of bytes to be read by the Symbol decoder.

This process is invoked when the function init\_symbol( sz ) is called from the syntax structure.

**Note:** The bit position will always be byte aligned when init\_symbol is invoked because the uncompressed header and the data partitions are always a whole number of bytes long.

The variable numBits is set equal to Min( sz \* 8, 15).

The variable buf is read using the f(numBits) parsing process.

The variable paddedBuf is set equal to ( buf << (15 - numBits) ).

The variable SymbolValue is set to ((1 << 15) - 1) ^ paddedBuf.

The variable `SymbolRange` is set to  $1 \ll 15$ .

The variable `SymbolMaxBits` is set to  $8 * sz - 15$ .

`SymbolMaxBits` (when non-negative) represents the number of bits still available to be read. It is allowed for this number to go negative (either here or during `read_symbol`). `SymbolMaxBits` (when negative) signifies that all available bits have been read, and that `-SymbolMaxBits` of padding zero bits have been used in the symbol decoding process. These padding zero bits are not present in the bitstream.

The array `TileIntraFrameYModeCdf` is set equal to a copy of `Default_Intra_Frame_Y_Mode_Cdf`.

A copy is made of each of the CDF arrays mentioned in the semantics for `init_coeff_cdfs` and `init_non_coeff_cdfs`. The name of the copy is the name of the CDF array prefixed with “Tile”. This copying produces the following arrays:

- `TileYModeCdf`
- `TileUVModeCflNotAllowedCdf`
- `TileUVModeCflAllowedCdf`
- `TileAngleDeltaCdf`
- `TileIntrabcCdf`
- `TilePartitionW8Cdf`
- `TilePartitionW16Cdf`
- `TilePartitionW32Cdf`
- `TilePartitionW64Cdf`
- `TilePartitionW128Cdf`
- `TileSegmentIdCdf`
- `TileSegmentIdPredictedCdf`
- `TileTx8x8Cdf`
- `TileTx16x16Cdf`
- `TileTx32x32Cdf`
- `TileTx64x64Cdf`
- `TileTxfmSplitCdf`
- `TileFilterIntraModeCdf`

- TileFilterIntraCdf
- TileInterpFilterCdf
- TileMotionModeCdf
- TileNewMvCdf
- TileZeroMvCdf
- TileRefMvCdf
- TileCompoundModeCdf
- TileDrlModeCdf
- TileIsInterCdf
- TileCompModeCdf
- TileSkipModeCdf
- TileSkipCdf
- TileCompRefCdf
- TileCompBwdRefCdf
- TileSingleRefCdf
- TileMvJointCdf
- TileMvSignCdf
- TileMvClassCdf
- TileMvClass0BitCdf
- TileMvFrCdf
- TileMvClass0FrCdf
- TileMvClass0HpCdf
- TileMvBitCdf
- TileMvHpCdf
- TilePaletteYModeCdf

- TilePaletteUVModeCdf
- TilePaletteYSizeCdf
- TilePaletteUVSizeCdf
- TilePaletteSize2YColorCdf
- TilePaletteSize2UVColorCdf
- TilePaletteSize3YColorCdf
- TilePaletteSize3UVColorCdf
- TilePaletteSize4YColorCdf
- TilePaletteSize4UVColorCdf
- TilePaletteSize5YColorCdf
- TilePaletteSize5UVColorCdf
- TilePaletteSize6YColorCdf
- TilePaletteSize6UVColorCdf
- TilePaletteSize7YColorCdf
- TilePaletteSize7UVColorCdf
- TilePaletteSize8YColorCdf
- TilePaletteSize8UVColorCdf
- TileDeltaQCdf
- TileDeltaLFCdf
- TileDeltaLFMultiCdf[ i ] for i = 0..FRAME\_LF\_COUNT-1
- TileIntraTxTypeSet1Cdf
- TileIntraTxTypeSet2Cdf
- TileInterTxTypeSet1Cdf
- TileInterTxTypeSet2Cdf
- TileInterTxTypeSet3Cdf

- TileUseObmcCdf
- TileInterIntraCdf
- TileCompRefTypeCdf
- TileCflSignCdf
- TileUniCompRefCdf
- TileWedgeInterIntraCdf
- TileCompGroupIdxCdf
- TileCompoundIdxCdf
- TileCompoundTypeCdf
- TileInterIntraModeCdf
- TileWedgeIndexCdf
- TileCflAlphaCdf
- TileUseWienerCdf
- TileUseSgrprojCdf
- TileRestorationTypeCdf
- TileTxbSkipCdf
- TileEobPt16Cdf
- TileEobPt32Cdf
- TileEobPt64Cdf
- TileEobPt128Cdf
- TileEobPt256Cdf
- TileEobPt512Cdf
- TileEobPt1024Cdf
- TileEobExtraCdf
- TileDcSignCdf

- TileCoeffBaseEobCdf
- TileCoeffBaseCdf
- TileCoeffBrCdf

### 8.2.3. Boolean decoding process

This process decodes a pseudo-raw bit assuming equal probability for decoding a 0 or a 1.

This process is invoked when the function `read_bool( )` is called from the `read_literal` function in [section 8.2.5](#).

An array `cdf` of length 3 is constructed as follows:

```
cdf[ 0 ] = 1 << 14
cdf[ 1 ] = 1 << 15
cdf[ 2 ] = 0
```

The output of this process is given by `read_symbol( cdf )`.

**Note:** This `cdf` array is constructed each time `read_bool` is invoked. This means that implementations can omit the `cdf` update performed by `read_symbol` when invoked from `read_bool` because the modified values are never used.

### 8.2.4. Exit process for symbol decoder

This process is invoked when the function `exit_symbol( )` is called from the syntax structure.

It is a requirement of bitstream conformance that `SymbolMaxBits` is greater than or equal to -14 whenever this process is invoked.

The variable `trailingBitPosition` is set equal to `get_position() - Min(15, SymbolMaxBits+15)`.

The bitstream position indicator is advanced by `Max(0, SymbolMaxBits)`. (This skips over any trailing bits that have not already been read during symbol decode.)

The variable `paddingEndPosition` is set equal to `get_position()`.

**Note:** `paddingEndPosition` will always be a multiple of 8 indicating that the bit position is byte aligned.

It is a requirement of bitstream conformance that the bit at position `trailingBitPosition` is equal to 1.

It is a requirement of bitstream conformance that the bit at position `x` is equal to 0 for values of `x` strictly between `trailingBitPosition` and `paddingEndPosition`.

**Note:** This exit process consumes the OBU trailing bits for a tile group.

If `disable_frame_end_update_cdf` is equal to 0 and `TileNum` is equal to `context_update_tile_id`, a copy is made of the final CDF values for each of the CDF arrays mentioned in the semantics for `init_coeff_cdfs` and `init_non_coeff_cdfs`. The name of the destination for the copy is the name of the CDF array prefixed with “Saved”. The name of the source for the copy is the name of the CDF array prefixed with “Tile”. For example, an array `SavedYModeCdf` will be created with values equal to `TileYModeCdf`.

## 8.2.5. Parsing process for `read_literal`

This process is invoked when the function `read_literal( n )` is invoked.

This process is specified as follows:

```
x = 0
for ( i = 0 ; i < n; i++ ) {
    x = 2 * x + read_bool( )
}
```

The return value for the function is given by `x`.

## 8.2.6. Symbol decoding process

The input to this process is an array `cdf` of length `N + 1` which specifies the cumulative distribution for a symbol with `N` possible values.

The output of this process is the variable `symbol`, containing a decoded syntax element. The process also modifies the input array `cdf` to adapt the probabilities to the content of the stream.

This process is invoked when the function `read_symbol( cdf )` is called.

**Note:** When this process is invoked, `N` will be greater than 1 and `cdf[ N-1 ]` will be equal to `1 << 15`.

The variables `cur`, `prev`, and `symbol` are calculated as follows:

```

cur = SymbolRange
symbol = -1
do {
    symbol++
    prev = cur
    f = ( 1 << 15 ) - cdf[ symbol ]
    cur = ((SymbolRange >> 8) * (f >> EC_PROB_SHIFT)) >> (7 - EC_PROB_SHIFT)
    cur += EC_MIN_PROB * (N - symbol - 1)
} while ( SymbolValue < cur )

```

**Note:** Implementations may prefer to store the inverse cdf to move the subtraction out of this loop.

The variable SymbolRange is set to prev - cur.

The variable SymbolValue is set equal to SymbolValue - cur.

The range and value are renormalized by the following ordered steps:

1. The variable bits is set to  $15 - \text{FloorLog2}(\text{SymbolRange})$ . This represents the number of new bits to be added to SymbolValue.
2. The variable SymbolRange is set to  $\text{SymbolRange} \ll \text{bits}$ .
3. The variable numBits is set equal to  $\text{Min}(\text{bits}, \text{Max}(0, \text{SymbolMaxBits}))$ . This represents the number of new bits to read from the bitstream.
4. The variable newData is read using the  $f(\text{numBits})$  parsing process.
5. The variable paddedData is set equal to  $\text{newData} \ll (\text{bits} - \text{numBits})$ .
6. The variable SymbolValue is set to  $\text{paddedData} \wedge ((\text{SymbolValue} + 1) \ll \text{bits}) - 1$ .
7. The variable SymbolMaxBits is set to  $\text{SymbolMaxBits} - \text{bits}$ .

**Note:** bits may be equal to 0, in which case these ordered steps have no effect.

If disable\_cdf\_update is equal to 0, the cumulative distribution is updated as follows:



```

rate = 3 + ( cdf[ N ] > 15 ) + ( cdf[ N ] > 31 ) + Min( FloorLog2( N ), 2 )
tmp = 0
for ( i = 0; i < N - 1; i++ ) {
    tmp = ( i == symbol ) ? ( 1 << 15 ) : tmp
    if ( tmp < cdf[ i ] ) {
        cdf[ i ] -= ( ( cdf[ i ] - tmp ) >> rate )
    } else {
        cdf[ i ] += ( ( tmp - cdf[ i ] ) >> rate )
    }
}
cdf[ N ] += ( cdf[ N ] < 32 )

```

**Note:** The last entry of the cdf array is used to keep a count of the number of times the symbol has been decoded (up to a maximum of 32). This allows the cdf adaption rate to depend on the number of times the symbol has been decoded.

The return value from the function is given by symbol.

## 8.3. Parsing process for CDF encoded syntax elements

### 8.3.1. General

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to S.

The input to this process is the name of a syntax element.

Section 8.3.2 specifies how a CDF array is chosen for the syntax element. The variable cdf is set equal to a reference to this CDF array.

**Note:** The array must be passed by reference because read\_symbol will adjust the array contents.

The output of this process is the result of calling the function read\_symbol( cdf ).

### 8.3.2. Cdf selection process

The input to this process is the name of a syntax element.

The output of this process is a reference to a CDF array.

When the description in this section uses variables, these variables are taken to have the values defined by the syntax tables at the point that the syntax element is being decoded.

The probabilities depend on the syntax element as follows:

**use\_intrabc:** The cdf for use\_intrabc is given by TileIntrabcCdf.

**intra\_frame\_y\_mode:** The cdf for intra\_frame\_y\_mode is given by TileIntraFrameYModeCdf[ abovemode ][ leftmode ] where abovemode and leftmode are the intra modes used for the blocks immediately above and to the left of this block and are computed as:

```
abovemode = Intra_Mode_Context[ AvailU ? YModes[ MiRow - 1 ][ MiCol ] : DC_PRED ]
leftmode  = Intra_Mode_Context[ AvailL ? YModes[ MiRow ][ MiCol - 1 ] : DC_PRED ]
```

where Intra\_Mode\_Context is defined as follows:

```
Intra_Mode_Context[ INTRA_MODES ] = {
    0, 1, 2, 3, 4, 4, 4, 4, 3, 0, 1, 2, 0
}
```

**Note:** We are using a 2D array to store the YModes and UVModes for clarity. It is possible to reduce memory consumption by only storing one intra mode for each 4x4 horizontal and vertical position, i.e. to use two 1D arrays instead.

**y\_mode:** The cdf for y\_mode is given by TileYModeCdf[ ctx ] where the variable ctx is computed as Size\_Group[ MiSize ].

**uv\_mode:** The cdf for uv\_mode is derived as follows:

- If Lossless is equal to 1 and get\_plane\_residual\_size( MiSize, 1 ) is equal to BLOCK\_4X4, the cdf is given by TileUVModeCflAllowedCdf[ YMode ].
- Otherwise, if Lossless is equal to 0 and Max( Block\_Width[ MiSize ], Block\_Height[ MiSize ] ) <= 32, the cdf is given by TileUVModeCflAllowedCdf[ YMode ].
- Otherwise, the cdf is given by TileUVModeCflNotAllowedCdf[ YMode ].

**angle\_delta\_y:** The cdf for angle\_delta\_y is given by TileAngleDeltaCdf[YMode - V\_PRED].

**angle\_delta\_uv:** The cdf for angle\_delta\_uv is given by TileAngleDeltaCdf[UVMode - V\_PRED].

**partition:** The variable ctx is computed as follows:

```
bsl = Mi_Width_Log2[ bSize ]
above = AvailU && ( Mi_Width_Log2[ MiSizes[ r - 1 ][ c ] ] < bsl )
left = AvailL && ( Mi_Height_Log2[ MiSizes[ r ][ c - 1 ] ] < bsl )
ctx = left * 2 + above
```

The cdf is derived as follows:

- If bsl is equal to 1, the cdf is given by TilePartitionW8Cdf[ ctx ].

- Otherwise, if bsl is equal to 2, the cdf is given by TilePartitionW16Cdf[ ctx ].
- Otherwise, if bsl is equal to 3, the cdf is given by TilePartitionW32Cdf[ ctx ].
- Otherwise, if bsl is equal to 4, the cdf is given by TilePartitionW64Cdf[ ctx ].
- Otherwise (bsl is equal to 5), the cdf is given by TilePartitionW128Cdf[ ctx ].

**split\_or\_horz:** split\_or\_horz uses the same derivation for the variable ctx as for the syntax element partition.

The array partitionCdf is derived according to the cdf derivation procedure for the syntax element partition (note that bsl is never equal to 1 when decoding split\_or\_horz).

The cdf to return is given by an array of length 3 which is constructed as follows:

```

psum = ( partitionCdf[ PARTITION_VERT ] - partitionCdf[ PARTITION_VERT - 1 ] +
         partitionCdf[ PARTITION_SPLIT ] - partitionCdf[ PARTITION_SPLIT - 1 ] +
         partitionCdf[ PARTITION_HORZ_A ] - partitionCdf[ PARTITION_HORZ_A - 1 ] +
         partitionCdf[ PARTITION_VERT_A ] - partitionCdf[ PARTITION_VERT_A - 1 ] +
         partitionCdf[ PARTITION_VERT_B ] - partitionCdf[ PARTITION_VERT_B - 1 ] )
if ( bSize != BLOCK_128X128 )
    psum += partitionCdf[ PARTITION_VERT_4 ] - partitionCdf[ PARTITION_VERT_4 - 1 ] )
cdf[0] = ( 1 << 15 ) - psum
cdf[1] = 1 << 15
cdf[2] = 0

```

**Note:** The syntax element split\_or\_horz is not allowed to return a PARTITION\_VERT, so the probability for a vertical partition is assigned to the probability for the split partition.

**split\_or\_vert:** split\_or\_vert uses the same derivation for the variable ctx as for the syntax element partition.

The array partitionCdf is derived according to the cdf derivation procedure for the syntax element partition (note that bsl is never equal to 1 when decoding split\_or\_vert).

The cdf to return is given by an array of length 3 which is constructed as follows:

```

psum = ( partitionCdf[ PARTITION_HORZ ] - partitionCdf[ PARTITION_HORZ - 1 ] +
         partitionCdf[ PARTITION_SPLIT ] - partitionCdf[ PARTITION_SPLIT - 1 ] +
         partitionCdf[ PARTITION_HORZ_A ] - partitionCdf[ PARTITION_HORZ_A - 1 ] +
         partitionCdf[ PARTITION_HORZ_B ] - partitionCdf[ PARTITION_HORZ_B - 1 ] +
         partitionCdf[ PARTITION_VERT_A ] - partitionCdf[ PARTITION_VERT_A - 1 ] )
if ( bSize != BLOCK_128X128 )
    psum += partitionCdf[ PARTITION_HORZ_4 ] - partitionCdf[ PARTITION_HORZ_4 - 1 ]
cdf[0] = ( 1 << 15 ) - psum
cdf[1] = 1 << 15
cdf[2] = 0

```

**tx\_depth**: the cdf depends on the value of maxRectTxSize and ctx, where ctx is computed by:

```

maxTxWidth = Tx_Width[ maxRectTxSize ]
maxTxHeight = Tx_Height[ maxRectTxSize ]

if ( AvailU && IsInters[ MiRow - 1 ][ MiCol ] ) {
    aboveW = Block_Width[ MiSizes[ MiRow - 1 ][ MiCol ] ]
} else if ( AvailU ) {
    aboveW = get_above_tx_width( MiRow, MiCol )
} else {
    aboveW = 0
}

if ( AvailL && IsInters[ MiRow ][ MiCol - 1 ] ) {
    leftH = Block_Height[ MiSizes[ MiRow ][ MiCol - 1 ] ]
} else if ( AvailL ) {
    leftH = get_left_tx_height( MiRow, MiCol )
} else {
    leftH = 0
}

ctx = ( aboveW >= maxTxWidth ) + ( leftH >= maxTxHeight )

```

where get\_above\_tx\_width and get\_left\_tx\_height are functions defined as specified in the CDF selection process for txfm\_split.

The cdf to return is given by:

- TileTx64x64Cdf[ ctx ] if maxTxDepth is equal to 4.
- TileTx32x32Cdf[ ctx ] if maxTxDepth is equal to 3.
- TileTx16x16Cdf[ ctx ] if maxTxDepth is equal to 2.
- TileTx8x8Cdf[ ctx ] otherwise.

**txfm\_split**: the cdf is given by TileTxfmSplitCdf[ ctx ], where ctx is computed by:

```

above = get_above_tx_width( row, col ) < Tx_Width[ txSz ]
left  = get_left_tx_height( row, col ) < Tx_Height[ txSz ]
size  = Min( 64, Max( Block_Width[ MiSize ], Block_Height[ MiSize ] ) )
maxTxSz = find_tx_size( size, size )
txSzSqrUp = Tx_Size_Sqr_Up[ txSz ]
ctx = (txSzSqrUp != maxTxSz) * 3 +
      (TX_SIZES - 1 - maxTxSz) * 6 + above + left

```

where `get_above_tx_width` and `get_left_tx_height` are functions defined as follows:

```

get_above_tx_width( row, col ) {
    if ( row == MiRow ) {
        if ( !AvailU ) {
            return 64
        } else if ( Skips[ row - 1 ][ col ] && IsInters[ row - 1 ][ col ] ) {
            return Block_Width[ MiSizes[ row - 1 ][ col ] ]
        }
    }
    return Tx_Width[ InterTxSizes[ row - 1 ][ col ] ]
}

get_left_tx_height( row, col ) {
    if ( col == MiCol ) {
        if ( !AvailL ) {
            return 64
        } else if ( Skips[ row ][ col - 1 ] && IsInters[ row ][ col - 1 ] ) {
            return Block_Height[ MiSizes[ row ][ col - 1 ] ]
        }
    }
    return Tx_Height[ InterTxSizes[ row ][ col - 1 ] ]
}

```

**segment\_id:** The cdf is given by `TileSegmentIdCdf[ ctx ]` where `ctx` is computed by:

```

if ( prevUL < 0 )
    ctx = 0
else if ( (prevUL == prevU) && (prevUL == prevL) )
    ctx = 2
else if ( (prevUL == prevU) || (prevUL == prevL) || (prevU == prevL) )
    ctx = 1
else
    ctx = 0

```

**seg\_id\_predicted:** the cdf is given by `TileSegmentIdPredictedCdf[ ctx ]`, where `ctx` is computed by:

```
ctx = LeftSegPredContext[ MiRow ] + AboveSegPredContext[ MiCol ]
```

**new\_mv**: the cdf is given by TileNewMvCdf[ NewMvContext ].

**zero\_mv**: the cdf is given by TileZeroMvCdf[ ZeroMvContext ].

**ref\_mv**: the cdf is given by TileRefMvCdf[ RefMvContext ].

**drl\_mode**: the cdf is given by TileDrlModeCdf[ DrlCtxStack[ idx ] ].

**is\_inter**: the cdf is given by TileIsInterCdf[ ctx ] where ctx is computed by:

```
if ( AvailU && AvailL )
    ctx = (LeftIntra && AboveIntra) ? 3 : LeftIntra || AboveIntra
else if ( AvailU || AvailL )
    ctx = 2 * (AvailU ? AboveIntra : LeftIntra)
else
    ctx = 0
```

**use\_filter\_intra**: the cdf is given by TileFilterIntraCdf[ MiSize ].

**filter\_intra\_mode**: the cdf is given by TileFilterIntraModeCdf.

**comp\_mode**: the cdf is given by TileCompModeCdf[ ctx ] where ctx is computed by:

```

if ( AvailU && AvailL ) {
    if ( AboveSingle && LeftSingle )
        ctx = check_backward( AboveRefFrame[ 0 ] )
            ^ check_backward( LeftRefFrame[ 0 ] )
    else if ( AboveSingle )
        ctx = 2 + ( check_backward( AboveRefFrame[ 0 ] ) || AboveIntra )
    else if ( LeftSingle )
        ctx = 2 + ( check_backward( LeftRefFrame[ 0 ] ) || LeftIntra )
    else
        ctx = 4
} else if ( AvailU ) {
    if ( AboveSingle )
        ctx = check_backward( AboveRefFrame[ 0 ] )
    else
        ctx = 3
} else if ( AvailL ) {
    if ( LeftSingle )
        ctx = check_backward( LeftRefFrame[ 0 ] )
    else
        ctx = 3
} else {
    ctx = 1
}

```

where `check_backward` is a function specified as follows:

```

check_backward(refFrame) {
    return ( ( refFrame >= BWDREF_FRAME ) && ( refFrame <= ALTREF_FRAME ) )
}

```

**skip\_mode:** the cdf is given by `TileSkipModeCdf[ ctx ]` where `ctx` is computed by:

```

ctx = 0
if ( AvailU )
    ctx += SkipModes[ MiRow - 1 ][ MiCol ]
if ( AvailL )
    ctx += SkipModes[ MiRow ][ MiCol - 1 ]

```

**skip:** the cdf is given by `TileSkipCdf[ ctx ]` where `ctx` is computed by:

```

ctx = 0
if ( AvailU )
    ctx += Skips[ MiRow - 1 ][ MiCol ]
if ( AvailL )
    ctx += Skips[ MiRow ][ MiCol - 1 ]

```

**comp\_ref:** the cdf is given by TileCompRefCdf[ ctx ][ 0 ] where ctx is computed by:

```

last12Count = count_refs( LAST_FRAME ) + count_refs( LAST2_FRAME )
last3GoldCount = count_refs( LAST3_FRAME ) + count_refs( GOLDEN_FRAME )
ctx = ref_count_ctx( last12Count, last3GoldCount )

```

where count\_refs is defined as:

```

count_refs(frameType) {
    c = 0
    if ( AvailU ) {
        if ( AboveRefFrame[ 0 ] == frameType ) c++
        if ( AboveRefFrame[ 1 ] == frameType ) c++
    }
    if ( AvailL ) {
        if ( LeftRefFrame[ 0 ] == frameType ) c++
        if ( LeftRefFrame[ 1 ] == frameType ) c++
    }
    return c
}

```

and ref\_count\_ctx is defined as:

```

ref_count_ctx(counts0, counts1) {
    if ( counts0 < counts1 )
        return 0
    else if ( counts0 == counts1 )
        return 1
    else
        return 2
}

```

**comp\_ref\_p1:** the cdf is given by TileCompRefCdf[ ctx ][ 1 ] where ctx is computed by:



```

lastCount = count_refs( LAST_FRAME )
last2Count = count_refs( LAST2_FRAME )
ctx = ref_count_ctx( lastCount, last2Count )

```

where `count_refs` and `ref_count_ctx` are the same as given in the CDF selection process for `comp_ref`.

**comp\_ref\_p2:** the cdf is given by `TileCompRefCdf[ ctx ][ 2 ]` where `ctx` is computed by:

```

last3Count = count_refs( LAST3_FRAME )
goldCount = count_refs( GOLDEN_FRAME )
ctx = ref_count_ctx( last3Count, goldCount )

```

where `count_refs` and `ref_count_ctx` are the same as given in the CDF selection process for `comp_ref`.

**comp\_bwdref:** the cdf is given by `TileCompBwdRefCdf[ ctx ][ 0 ]` where `ctx` is computed by:

```

brfarf2Count = count_refs( BWDREF_FRAME ) + count_refs( ALTREF2_FRAME )
arfCount = count_refs( ALTREF_FRAME )
ctx = ref_count_ctx( brfarf2Count, arfCount )

```

where `count_refs` and `ref_count_ctx` are the same as given in the CDF selection process for `comp_ref`.

**comp\_bwdref\_p1:** the cdf is given by `TileCompBwdRefCdf[ ctx ][ 1 ]` where `ctx` is computed by:

```

brfCount = count_refs( BWDREF_FRAME )
arf2Count = count_refs( ALTREF2_FRAME )
ctx = ref_count_ctx( brfCount, arf2Count )

```

where `count_refs` and `ref_count_ctx` are the same as given in the CDF selection process for `comp_ref`.

**single\_ref\_p1:** the cdf is given by `TileSingleRefCdf[ ctx ][ 0 ]` where `ctx` is computed by:

```

fwdCount = count_refs( LAST_FRAME )
fwdCount += count_refs( LAST2_FRAME )
fwdCount += count_refs( LAST3_FRAME )
fwdCount += count_refs( GOLDEN_FRAME )
bwdCount = count_refs( BWDREF_FRAME )
bwdCount += count_refs( ALTREF2_FRAME )
bwdCount += count_refs( ALTREF_FRAME )
ctx = ref_count_ctx( fwdCount, bwdCount )

```

where `count_refs` and `ref_count_ctx` are the same as given in the CDF selection process for `comp_ref`.

**single\_ref\_p2:** the cdf is given by `TileSingleRefCdf[ ctx ][ 1 ]` where `ctx` is computed as in the CDF selection process for `comp_bwdref`.

**single\_ref\_p3:** the cdf is given by `TileSingleRefCdf[ ctx ][ 2 ]` where `ctx` is computed as in the CDF selection process for `comp_ref`.

**single\_ref\_p4:** the cdf is given by `TileSingleRefCdf[ ctx ][ 3 ]` where `ctx` is computed as in the CDF selection process for `comp_ref_p1`.

**single\_ref\_p5:** the cdf is given by `TileSingleRefCdf[ ctx ][ 4 ]` where `ctx` is computed as in the CDF selection process for `comp_ref_p2`.

**single\_ref\_p6:** the cdf is given by `TileSingleRefCdf[ ctx ][ 5 ]` where `ctx` is computed as in the CDF selection process for `comp_bwdref_p1`.

**compound\_mode:** the cdf is given by `TileCompoundModeCdf[ ctx ]` where `ctx` is computed by:

```
ctx = Compound_Mode_Ctx_Map[ RefMvContext >> 1 ][ Min(NewMvContext, COMP_NEWMV_CTXS - 1) ]
```

where `Compound_Mode_Ctx_Map` is defined as follows:

```
Compound_Mode_Ctx_Map[ 3 ][ COMP_NEWMV_CTXS ] = {
    { 0, 1, 1, 1, 1 },
    { 1, 2, 3, 4, 4 },
    { 4, 4, 5, 6, 7 }
}
```

**interp\_filter:** the cdf is given by `TileInterpFilterCdf[ ctx ]` where `ctx` is computed by:

```

ctx = ( ( dir & 1 ) * 2 + ( RefFrame[ 1 ] > INTRA_FRAME ) ) * 4
leftType = 3
aboveType = 3

if ( AvailL ) {
    if ( RefFrames[ MiRow ][ MiCol - 1 ][ 0 ] == RefFrame[ 0 ] ||
        RefFrames[ MiRow ][ MiCol - 1 ][ 1 ] == RefFrame[ 0 ] )
        leftType = InterpFilters[ MiRow ][ MiCol - 1 ][ dir ]
}

if ( AvailU ) {
    if ( RefFrames[ MiRow - 1 ][ MiCol ][ 0 ] == RefFrame[ 0 ] ||
        RefFrames[ MiRow - 1 ][ MiCol ][ 1 ] == RefFrame[ 0 ] )
        aboveType = InterpFilters[ MiRow - 1 ][ MiCol ][ dir ]
}

if ( leftType == aboveType )
    ctx += leftType
else if ( leftType == 3 )
    ctx += aboveType
else if ( aboveType == 3 )
    ctx += leftType
else
    ctx += 3

```

**motion\_mode:** the cdf is given by TileMotionModeCdf[ MiSize ].

**mv\_joint:** the cdf is given by TileMvJointCdf[ MvCtx ].

**mv\_sign:** the cdf is given by TileMvSignCdf[ MvCtx ][ comp ].

**mv\_class:** the cdf is given by TileMvClassCdf[ MvCtx ][ comp ].

**mv\_class0\_bit:** the cdf is given by TileMvClass0BitCdf[ MvCtx ][ comp ].

**mv\_class0\_fr:** the cdf is given by TileMvClass0FrCdf[ MvCtx ][ comp ][ mv\_class0\_bit ].

**mv\_class0\_hp:** the cdf is given by TileMvClass0HpCdf[ MvCtx ][ comp ].

**mv\_fr:** the cdf is given by TileMvFrCdf[ MvCtx ][ comp ].

**mv\_hp:** the cdf is given by TileMvHpCdf[ MvCtx ][ comp ].

**mv\_bit:** the cdf is given by TileMvBitCdf[ MvCtx ][ comp ][ i ].

**all\_zero:** the cdf is given by TileTxbSkipCdf[ txSzCtx ][ ctx ], where ctx is computed as follows:

```

maxX4 = MiCols
maxY4 = MiRows
if ( plane > 0 ) {
    maxX4 = maxX4 >> subsampling_x
    maxY4 = maxY4 >> subsampling_y
}

w = Tx_Width[txSz]
h = Tx_Height[txSz]

bsize = get_plane_residual_size( MiSize, plane )
bw = Block_Width[ bsize ]
bh = Block_Height[ bsize ]

if ( plane == 0 ) {
    top = 0
    left = 0
    for ( k = 0; k < w4; k++ ) {
        if ( x4 + k < maxX4 )
            top = Max( top, AboveLevelContext[ plane ][ x4 + k ] )
    }
    for ( k = 0; k < h4; k++ ) {
        if ( y4 + k < maxY4 )
            left = Max( left, LeftLevelContext[ plane ][ y4 + k ] )
    }
    top = Min( top, 255 )
    left = Min( left, 255 )
    if ( bw == w && bh == h ) {
        ctx = 0
    } else if ( top == 0 && left == 0 ) {
        ctx = 1
    } else if ( top == 0 || left == 0 ) {
        ctx = 2 + ( Max( top, left ) > 3 )
    } else if ( Max( top, left ) <= 3 ) {
        ctx = 4
    } else if ( Min( top, left ) <= 3 ) {
        ctx = 5
    } else {
        ctx = 6
    }
} else {
    above = 0
    left = 0
    for ( i = 0; i < w4; i++ ) {
        if ( x4 + i < maxX4 ) {
            above |= AboveLevelContext[ plane ][ x4 + i ]
            above |= AboveDcContext[ plane ][ x4 + i ]
        }
    }
    for ( i = 0; i < h4; i++ ) {

```

```

    if ( y4 + i < maxY4 ) {
        left |= LeftLevelContext[ plane ][ y4 + i ]
        left |= LeftDcContext[ plane ][ y4 + i ]
    }
}
ctx = ( above != 0 ) + ( left != 0 )
ctx += 7
if ( bw * bh > w * h )
    ctx += 3
}

```

**eob\_pt\_16:** the cdf is given by TileEobPt16Cdf[ ptype ][ ctx ], where ctx is computed as follows:

```

txType = compute_tx_type( plane, txSz, x4, y4 )
ctx = ( get_tx_class( txType ) == TX_CLASS_2D ) ? 0 : 1

```

where get\_tx\_class() is defined as in the CDF selection for coeff\_base.

**eob\_pt\_32:** the cdf is given by TileEobPt32Cdf[ ptype ][ ctx ], where ctx is computed as for eob\_pt\_16.

**eob\_pt\_64:** the cdf is given by TileEobPt64Cdf[ ptype ][ ctx ], where ctx is computed as for eob\_pt\_16.

**eob\_pt\_128:** the cdf is given by TileEobPt128Cdf[ ptype ][ ctx ], where ctx is computed as for eob\_pt\_16.

**eob\_pt\_256:** the cdf is given by TileEobPt256Cdf[ ptype ][ ctx ], where ctx is computed as for eob\_pt\_16.

**eob\_pt\_512:** the cdf is given by TileEobPt512Cdf[ ptype ].

**eob\_pt\_1024:** the cdf is given by TileEobPt1024Cdf[ ptype ].

**eob\_extra:** the cdf is given by TileEobExtraCdf[ txSzCtx ][ ptype ][ eobPt - 3 ].

**coeff\_base:** the cdf is given by TileCoeffBaseCdf[ txSzCtx ][ ptype ][ ctx ], where ctx is computed as follows:

```

ctx = get_coeff_base_ctx(txSz, plane, x4, y4, scan[c], c, 0)

```

where get\_coeff\_base\_ctx is defined as:

```

get_coeff_base_ctx( txSz, plane, blockX, blockY, pos, c, isEob ) {
    adjTxSz = Adjusted_Tx_Size[ txSz ]
    bwl = Tx_Width_Log2[ adjTxSz ]
    width = 1 << bwl
    height = Tx_Height[ adjTxSz ]
    txType = compute_tx_type( plane, txSz, blockX, blockY )
    if ( isEob ) {
        if ( c == 0 ) {
            return SIG_COEF_CONTEXTS - 4
        }
        if ( c <= (height << bwl) / 8 ) {
            return SIG_COEF_CONTEXTS - 3
        }
        if ( c <= (height << bwl) / 4 ) {
            return SIG_COEF_CONTEXTS - 2
        }
        return SIG_COEF_CONTEXTS - 1
    }
    txClass = get_tx_class( txType )
    row = pos >> bwl
    col = pos - (row << bwl)
    mag = 0

    for ( idx = 0; idx < SIG_REF_DIFF_OFFSET_NUM; idx++ ) {
        refRow = row + Sig_Ref_Diff_Offset[ txClass ][ idx ][ 0 ]
        refCol = col + Sig_Ref_Diff_Offset[ txClass ][ idx ][ 1 ]
        if ( refRow >= 0 &&
            refCol >= 0 &&
            refRow < height &&
            refCol < width ) {
            mag += Min( Abs( Quant[ (refRow << bwl) + refCol ] ), 3 )
        }
    }

    ctx = Min( ( mag + 1 ) >> 1, 4 )
    if ( txClass == TX_CLASS_2D ) {
        if ( row == 0 && col == 0 ) {
            return 0
        }
        return ctx + Coeff_Base_Ctx_Offset[ txSz ][ Min( row, 4 ) ][ Min( col, 4 ) ]
    }
    idx = ( txClass == TX_CLASS_VERT ) ? row : col
    return ctx + Coeff_Base_Pos_Ctx_Offset[ Min( idx, 2 ) ]
}

```

where get\_tx\_class is defined as:

```
get_tx_class( txType ) {  
    if ( ( txType == V_DCT ) ||  
        ( txType == V_ADST ) ||  
        ( txType == V_FLIPADST ) ) {  
        return TX_CLASS_VERT  
    } else if ( ( txType == H_DCT ) ||  
                ( txType == H_ADST ) ||  
                ( txType == H_FLIPADST ) ) {  
        return TX_CLASS_HORIZ  
    } else  
        return TX_CLASS_2D  
}
```

Coeff\_Base\_Ctx\_Offset is defined as:

```

Coeff_Base_Ctx_Offset[ TX_SIZES_ALL ][ 5 ][ 5 ] = {
{
  { 0, 1, 6, 6, 0 },
  { 1, 6, 6, 21, 0 },
  { 6, 6, 21, 21, 0 },
  { 6, 21, 21, 21, 0 },
  { 0, 0, 0, 0, 0 }
},
{
  { 0, 1, 6, 6, 21 },
  { 1, 6, 6, 21, 21 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 1, 6, 6, 21 },
  { 1, 6, 6, 21, 21 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 1, 6, 6, 21 },
  { 1, 6, 6, 21, 21 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 11, 11, 11, 0 },
  { 11, 11, 11, 11, 0 },
  { 6, 6, 21, 21, 0 },
  { 6, 21, 21, 21, 0 },
  { 21, 21, 21, 21, 0 }
},
{
  { 0, 16, 6, 6, 21 },
  { 16, 16, 6, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 0, 0, 0, 0, 0 }
}
},

```



```

{
  { 0, 11, 11, 11, 11 },
  { 11, 11, 11, 11, 11 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 16, 6, 6, 21 },
  { 16, 16, 6, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 }
},
{
  { 0, 11, 11, 11, 11 },
  { 11, 11, 11, 11, 11 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 16, 6, 6, 21 },
  { 16, 16, 6, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 }
},
{
  { 0, 11, 11, 11, 11 },
  { 11, 11, 11, 11, 11 },
  { 6, 6, 21, 21, 21 },
  { 6, 21, 21, 21, 21 },
  { 21, 21, 21, 21, 21 }
},
{
  { 0, 16, 6, 6, 21 },
  { 16, 16, 6, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 },
  { 16, 16, 21, 21, 21 }
},
{
  { 0, 11, 11, 11, 0 },
  { 11, 11, 11, 11, 0 },
  { 6, 6, 21, 21, 0 },
  { 6, 21, 21, 21, 0 },
  { 21, 21, 21, 21, 0 }
},
{

```

```

    { 0, 16, 6, 6, 21 },
    { 16, 16, 6, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 0, 0, 0, 0, 0 }
},
{
    { 0, 11, 11, 11, 11 },
    { 11, 11, 11, 11, 11 },
    { 6, 6, 21, 21, 21 },
    { 6, 21, 21, 21, 21 },
    { 21, 21, 21, 21, 21 }
},
{
    { 0, 16, 6, 6, 21 },
    { 16, 16, 6, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 16, 16, 21, 21, 21 }
},
{
    { 0, 11, 11, 11, 11 },
    { 11, 11, 11, 11, 11 },
    { 6, 6, 21, 21, 21 },
    { 6, 21, 21, 21, 21 },
    { 21, 21, 21, 21, 21 }
},
{
    { 0, 16, 6, 6, 21 },
    { 16, 16, 6, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 16, 16, 21, 21, 21 },
    { 16, 16, 21, 21, 21 }
}
}

```

and Coeff\_Base\_Pos\_Ctx\_Offset is defined as:

```

Coeff_Base_Pos_Ctx_Offset[ 3 ] = {
    SIG_COEF_CONTEXTS_2D,
    SIG_COEF_CONTEXTS_2D + 5,
    SIG_COEF_CONTEXTS_2D + 10
}

```

**coeff\_base\_eob:** the cdf is given by TileCoeffBaseEobCdf[ txSzCtx ][ ptype ][ ctx ], where ctx is computed as follows:

```
ctx = get_coeff_base_ctx(txSz, plane, x4, y4, scan[c], c, 1) - SIG_COEF_CONTEXTS +
SIG_COEF_CONTEXTS_EOB
```

where `get_coeff_base_ctx()` is as defined in the CDF selection for `coeff_base`.

**dc\_sign:** the cdf is given by `TileDcSignCdf[ ptype ][ ctx ]`, where `ctx` is computed as follows:

```
maxX4 = MiCols
maxY4 = MiRows
if ( plane > 0 ) {
    maxX4 = maxX4 >> subsampling_x
    maxY4 = maxY4 >> subsampling_y
}

dcSign = 0
for ( k = 0; k < w4; k++ ) {
    if ( x4 + k < maxX4 ) {
        sign = AboveDcContext[ plane ][ x4 + k ]
        if ( sign == 1 ) {
            dcSign--
        } else if ( sign == 2 ) {
            dcSign++
        }
    }
}
for ( k = 0; k < h4; k++ ) {
    if ( y4 + k < maxY4 ) {
        sign = LeftDcContext[ plane ][ y4 + k ]
        if ( sign == 1 ) {
            dcSign--
        } else if ( sign == 2 ) {
            dcSign++
        }
    }
}
if ( dcSign < 0 ) {
    ctx = 1
} else if ( dcSign > 0 ) {
    ctx = 2
} else {
    ctx = 0
}
```

**coeff\_br:** the cdf is given by `TileCoeffBrCdf[ Min( txSzCtx, TX_32X32 ) ][ ptype ][ ctx ]`, where `ctx` is computed as follows:

```

adjTxSz = Adjusted_Tx_Size[ txSz ]
bwl = Tx_Width_Log2[ adjTxSz ]
txw = Tx_Width[ adjTxSz ]
txh = Tx_Height[ adjTxSz ]
row = pos >> bwl
col = pos - (row << bwl)

mag = 0

txType = compute_tx_type( plane, txSz, x4, y4 )
txClass = get_tx_class( txType )

for ( idx = 0; idx < 3; idx++ ) {
    refRow = row + Mag_Ref_Offset_With_Tx_Class[ txClass ][ idx ][ 0 ]
    refCol = col + Mag_Ref_Offset_With_Tx_Class[ txClass ][ idx ][ 1 ]
    if ( refRow >= 0 &&
        refCol >= 0 &&
        refRow < txh &&
        refCol < (1 << bwl) ) {
        mag += Min( Quant[ refRow * txw + refCol ], COEFF_BASE_RANGE + NUM_BASE_LEVELS + 1 )
    }
}

mag = Min( ( mag + 1 ) >> 1, 6 )
if ( pos == 0 ) {
    ctx = mag
} else if ( txClass == 0 ) {
    if ( ( row < 2 ) && ( col < 2 ) ) {
        ctx = mag + 7
    } else {
        ctx = mag + 14
    }
} else {
    if ( txClass == 1 ) {
        if ( col == 0 ) {
            ctx = mag + 7
        } else {
            ctx = mag + 14
        }
    } else {
        if ( row == 0 ) {
            ctx = mag + 7
        } else {
            ctx = mag + 14
        }
    }
}
}

```

where `get_tx_class()` is defined as in the CDF selection for `coeff_base`, and `Mag_Ref_Offset_With_Tx_Class` is defined as:

```
Mag_Ref_Offset_With_Tx_Class[ 3 ][ 3 ][ 2 ] = {
  { { 0, 1 }, { 1, 0 }, { 1, 1 } },
  { { 0, 1 }, { 1, 0 }, { 0, 2 } },
  { { 0, 1 }, { 1, 0 }, { 2, 0 } }
}
```

**has\_palette\_y**: the cdf is given by TilePaletteYModeCdf[ bsizeCtx ][ ctx ] where ctx is computed as follows:

```
ctx = 0
if ( AvailU && PaletteSizes[ 0 ][ MiRow - 1 ][ MiCol ] > 0 )
  ctx += 1
if ( AvailL && PaletteSizes[ 0 ][ MiRow ][ MiCol - 1 ] > 0 )
  ctx += 1
```

**has\_palette\_uv**: the cdf is given by TilePaletteUVModeCdf[ ctx ] where ctx is computed as follows:

```
ctx = ( PaletteSizeY > 0 ) ? 1 : 0
```

**palette\_size\_y\_minus\_2**: the cdf is given by TilePaletteYSizeCdf[ bsizeCtx ].

**palette\_size\_uv\_minus\_2**: the cdf is given by TilePaletteUVSizeCdf[ bsizeCtx ].

**palette\_color\_idx\_y**: the cdf depends on PaletteSizeY, as follows:

PaletteSizeY	cdf
2	TilePaletteSize2YColorCdf[ ctx ]
3	TilePaletteSize3YColorCdf[ ctx ]
4	TilePaletteSize4YColorCdf[ ctx ]
5	TilePaletteSize5YColorCdf[ ctx ]
6	TilePaletteSize6YColorCdf[ ctx ]
7	TilePaletteSize7YColorCdf[ ctx ]
8	TilePaletteSize8YColorCdf[ ctx ]

where ctx is computed as follows:

```
ctx = Palette_Color_Context[ ColorContextHash ]
```

**palette\_color\_idx\_uv**: the cdf depends on PaletteSizeUV, as follows:

PaletteSizeUV	cdf
2	TilePaletteSize2UVColorCdf[ ctx ]
3	TilePaletteSize3UVColorCdf[ ctx ]
4	TilePaletteSize4UVColorCdf[ ctx ]
5	TilePaletteSize5UVColorCdf[ ctx ]
6	TilePaletteSize6UVColorCdf[ ctx ]
7	TilePaletteSize7UVColorCdf[ ctx ]
8	TilePaletteSize8UVColorCdf[ ctx ]

where ctx is computed as follows:

```
ctx = Palette_Color_Context[ ColorContextHash ]
```

**delta\_q\_abs**: the cdf is given by TileDeltaQCdf.

**delta\_lf\_abs**: the cdf is derived as follows:

- If delta\_lf\_multi is equal to 0, the cdf is given by TileDeltaLFCdf.
- Otherwise (delta\_lf\_multi is equal to 1), the cdf is given by TileDeltaLFMultiCdf[ i ].

**intra\_tx\_type**: the cdf depends on the variable set, as follows:

set	cdf
TX_SET_INTRA_1	TileIntraTxTypeSet1Cdf[ Tx_Size_Sqr[ txSz ] ][ intraDir ]
TX_SET_INTRA_2	TileIntraTxTypeSet2Cdf[ Tx_Size_Sqr[ txSz ] ][ intraDir ]

where the variable intraDir is derived as follows:

- If use\_filter\_intra is equal to 1, intraDir is set equal to Filter\_Intra\_Mode\_To\_Intra\_Dir[ filter\_intra\_mode ],
- Otherwise (use\_filter\_intra is equal to 0), intraDir is set equal to YMode.

The table Filter\_Intra\_Mode\_To\_Intra\_Dir is defined as:

```
Filter_Intra_Mode_To_Intra_Dir[ INTRA_FILTER_MODES ] = {
    DC_PRED, V_PRED, H_PRED, D157_PRED, DC_PRED
}
```

**inter\_tx\_type:** the cdf depends on the variable set, as follows:

set	cdf
TX_SET_INTER_1	TileInterTxTypeSet1Cdf[ Tx_Size_Sqr[ txSz ] ]
TX_SET_INTER_2	TileInterTxTypeSet2Cdf
TX_SET_INTER_3	TileInterTxTypeSet3Cdf[ Tx_Size_Sqr[ txSz ] ]

**comp\_ref\_type:** The cdf is given by TileCompRefTypeCdf[ ctx ], where ctx is computed as follows:

```

above0 = AboveRefFrame[ 0 ]
above1 = AboveRefFrame[ 1 ]
left0 = LeftRefFrame[ 0 ]
left1 = LeftRefFrame[ 1 ]
aboveCompInter = AvailU && !AboveIntra && !AboveSingle
leftCompInter = AvailL && !LeftIntra && !LeftSingle
aboveUniComp = aboveCompInter && is_samedir_ref_pair(above0, above1)
leftUniComp = leftCompInter && is_samedir_ref_pair(left0, left1)

if ( AvailU && !AboveIntra && AvailL && !LeftIntra ) {
    samedir = is_samedir_ref_pair(above0, left0)

    if ( !aboveCompInter && !leftCompInter ) {
        ctx = 1 + 2 * samedir
    } else if ( !aboveCompInter ) {
        if ( !leftUniComp )
            ctx = 1
        else
            ctx = 3 + samedir
    } else if ( !leftCompInter ) {
        if ( !aboveUniComp )
            ctx = 1
        else
            ctx = 3 + samedir
    } else {
        if ( !aboveUniComp && !leftUniComp )
            ctx = 0
        else if ( !aboveUniComp || !leftUniComp )
            ctx = 2
        else
            ctx = 3 + ((above0 == BWDREF_FRAME) == (left0 == BWDREF_FRAME))
    }
} else if ( AvailU && AvailL ) {
    if ( aboveCompInter )
        ctx = 1 + 2 * aboveUniComp
    else if ( leftCompInter )
        ctx = 1 + 2 * leftUniComp
    else
        ctx = 2
} else if ( aboveCompInter ) {
    ctx = 4 * aboveUniComp
} else if ( leftCompInter ) {
    ctx = 4 * leftUniComp
} else {
    ctx = 2
}

```

where `is_samedir_ref_pair` is defined as:



```

is_samedir_ref_pair(ref0, ref1) {
    return (ref0 >= BWDREF_FRAME) == (ref1 >= BWDREF_FRAME)
}

```

**uni\_comp\_ref:** The cdf is given by TileUniCompRefCdf[ ctx ][ 0 ], where ctx is computed as in the CDF selection process for single\_ref\_p1.

**uni\_comp\_ref\_p1:** The cdf is given by TileUniCompRefCdf[ ctx ][ 1 ], where ctx is computed as follows:

```

last2Count = count_refs( LAST2_FRAME )
last3GoldCount = count_refs( LAST3_FRAME ) + count_refs( GOLDEN_FRAME )
ctx = ref_count_ctx( last2Count, last3GoldCount )

```

where count\_refs and ref\_count\_ctx are the same as given in the CDF selection process for comp\_ref.

**uni\_comp\_ref\_p2:** The cdf is given by TileUniCompRefCdf[ ctx ][ 2 ], where ctx is computed as in the CDF selection process for comp\_ref\_p2.

**comp\_group\_idx:** The cdf is given by TileCompGroupIdxCdf[ ctx ], where ctx is computed as follows:

```

ctx = 0
if ( AvailU ) {
    if ( !AboveSingle )
        ctx += CompGroupIdxs[ MiRow - 1 ][ MiCol ]
    else if ( AboveRefFrame[ 0 ] == ALTREF_FRAME )
        ctx += 3
}
if ( AvailL ) {
    if ( !LeftSingle )
        ctx += CompGroupIdxs[ MiRow ][ MiCol - 1 ]
    else if ( LeftRefFrame[ 0 ] == ALTREF_FRAME )
        ctx += 3
}
ctx = Min( 5, ctx )

```

**compound\_idx:** The cdf is given by TileCompoundIdxCdf[ ctx ], where ctx is computed as follows:

```

fwd = Abs( get_relative_dist( OrderHints[ RefFrame[ 0 ] ], OrderHint ) )
bck = Abs( get_relative_dist( OrderHints[ RefFrame[ 1 ] ], OrderHint ) )
ctx = ( fwd == bck ) ? 3 : 0
if ( AvailU ) {
    if ( !AboveSingle )
        ctx += CompoundIdxs[ MiRow - 1 ][ MiCol ]
    else if ( AboveRefFrame[ 0 ] == ALTREF_FRAME )
        ctx++
}
if ( AvailL ) {
    if ( !LeftSingle )
        ctx += CompoundIdxs[ MiRow ][ MiCol - 1 ]
    else if ( LeftRefFrame[ 0 ] == ALTREF_FRAME )
        ctx++
}

```

**compound\_type:** The cdf is given by TileCompoundTypeCdf[ MiSize ].

**interintra:** The cdf is given by TileInterIntraCdf[ ctx ], where ctx is computed as follows:

```
ctx = Size_Group[ MiSize ] - 1
```

**interintra\_mode:** The cdf is given by TileInterIntraModeCdf[ ctx ], where ctx is computed as follows:

```
ctx = Size_Group[ MiSize ] - 1
```

**wedge\_index:** The cdf is given by TileWedgeIndexCdf[ MiSize ].

**wedge\_interintra:** The cdf is given by TileWedgeInterIntraCdf[ MiSize ].

**use\_obmc:** The cdf is given by TileUseObmcCdf[ MiSize ].

**cfl\_alpha\_signs:** The cdf is given by TileCflSignCdf.

**cfl\_alpha\_u:** The cdf is given by TileCflAlphaCdf[ ctx ], where ctx is obtained from the following table:

cfl_alpha_signs	ctx
0	N/A
1	N/A
2	0
3	1

cfl_alpha_signs	ctx
4	2
5	3
6	4
7	5

**Note:** N/A is used to indicate that no context is needed as the sign is zero and no value is decoded.

or computed as follows:

$$\text{ctx} = (\text{signU} - 1) * 3 + \text{signV}$$

**Note:** As shown in the previous table, the variable ctx produced by this calculation will be equal to cfl\_alpha\_signs - 2.

**cfl\_alpha\_v:** The cdf is given by TileCflAlphaCdf[ ctx ], where ctx is obtained from the following table:

cfl_alpha_signs	ctx
0	0
1	3
2	N/A
3	1
4	4
5	N/A
6	2
7	5

**Note:** N/A is used to indicate that no context is needed as the sign is zero and no value is decoded.

or computed as follows:

$$\text{ctx} = (\text{signV} - 1) * 3 + \text{signU}$$

**use\_wiener:** The cdf is given by TileUseWienerCdf.

**use\_sgrproj:** The cdf is given by TileUseSgrprojCdf.

**restoration\_type:** The cdf is given by TileRestorationTypeCdf.

## 9. Additional tables

### 9.1. General

This section contains tables that do not naturally fit in the main sections of the Specification.

### 9.2. Scan tables

This section defines the scan order for different types of transform. Each table is named in the form `<type>_Scan_<w>x<h>`, and contains an ordered list of positions within a rectangle of width `w` and height `h`. Each position is calculated as `w * y + x`.

The following table lists pairs of scan tables which are transposes of each other. A table `T_wxh` is a transpose of another table `T_hxw` if the following function returns 1:

```
is_transpose( ) {
    for ( pos = 0; pos < w * h; pos++ ) {
        x1 = T_wxh[ pos ] % w
        y1 = T_wxh[ pos ] / w
        x2 = T_hxw[ pos ] % h
        y2 = T_hxw[ pos ] / h
        if ( x1 != y2 || y1 != x2 )
            return 0
    }
    return 1
}
```

Table	Transpose
Default_Scan_4x8	Default_Scan_8x4
Default_Scan_16x32	Default_Scan_32x16
Default_Scan_16x4	Default_Scan_4x16
Default_Scan_16x8	Default_Scan_8x16
Default_Scan_32x8	Default_Scan_8x32
Mcol_Scan_4x4	Mrow_Scan_4x4
Mcol_Scan_4x8	Mrow_Scan_8x4
Mcol_Scan_8x8	Mrow_Scan_8x8
Mcol_Scan_16x8	Mrow_Scan_8x16
Mcol_Scan_16x16	Mrow_Scan_16x16
Mcol_Scan_16x4	Mrow_Scan_4x16

Table	Transpose
Mrow_Scan_4x4	Mcol_Scan_4x4
Mrow_Scan_4x8	Mcol_Scan_8x4
Mrow_Scan_8x8	Mcol_Scan_8x8
Mrow_Scan_16x8	Mcol_Scan_8x16
Mrow_Scan_16x16	Mcol_Scan_16x16
Mrow_Scan_16x4	Mcol_Scan_4x16

```
Default_Scan_4x4[ 16 ] = {
    0, 1, 4, 8,
    5, 2, 3, 6,
    9, 12, 13, 10,
    7, 11, 14, 15
}
```

```
Mcol_Scan_4x4[ 16 ] = {
    0, 4, 8, 12,
    1, 5, 9, 13,
    2, 6, 10, 14,
    3, 7, 11, 15
}
```

```
Mrow_Scan_4x4[ 16 ] = {
    0, 1, 2, 3,
    4, 5, 6, 7,
    8, 9, 10, 11,
    12, 13, 14, 15
}
```

```
Default_Scan_4x8[ 32 ] = {
    0, 1, 4, 2, 5, 8, 3, 6, 9, 12, 7, 10, 13, 16, 11, 14,
    17, 20, 15, 18, 21, 24, 19, 22, 25, 28, 23, 26, 29, 27, 30, 31
}
```

```
Mcol_Scan_4x8[ 32 ] = {
    0, 4, 8, 12, 16, 20, 24, 28, 1, 5, 9, 13, 17, 21, 25, 29,
    2, 6, 10, 14, 18, 22, 26, 30, 3, 7, 11, 15, 19, 23, 27, 31
}
```

```
Mrow_Scan_4x8[ 32 ] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
}
```

```
Default_Scan_8x4[ 32 ] = {
    0, 8, 1, 16, 9, 2, 24, 17, 10, 3, 25, 18, 11, 4, 26, 19,
    12, 5, 27, 20, 13, 6, 28, 21, 14, 7, 29, 22, 15, 30, 23, 31
}
```

```
Mcol_Scan_8x4[ 32 ] = {
    0, 8, 16, 24, 1, 9, 17, 25, 2, 10, 18, 26, 3, 11, 19, 27,
    4, 12, 20, 28, 5, 13, 21, 29, 6, 14, 22, 30, 7, 15, 23, 31
}
```

```
Mrow_Scan_8x4[ 32 ] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
}
```

```
Default_Scan_8x8[ 64 ] = {
    0, 1, 8, 16, 9, 2, 3, 10,
    17, 24, 32, 25, 18, 11, 4, 5,
    12, 19, 26, 33, 40, 48, 41, 34,
    27, 20, 13, 6, 7, 14, 21, 28,
    35, 42, 49, 56, 57, 50, 43, 36,
    29, 22, 15, 23, 30, 37, 44, 51,
    58, 59, 52, 45, 38, 31, 39, 46,
    53, 60, 61, 54, 47, 55, 62, 63
}
```

```

Mcol_Scan_8x8[ 64 ] = {
    0, 8, 16, 24, 32, 40, 48, 56,
    1, 9, 17, 25, 33, 41, 49, 57,
    2, 10, 18, 26, 34, 42, 50, 58,
    3, 11, 19, 27, 35, 43, 51, 59,
    4, 12, 20, 28, 36, 44, 52, 60,
    5, 13, 21, 29, 37, 45, 53, 61,
    6, 14, 22, 30, 38, 46, 54, 62,
    7, 15, 23, 31, 39, 47, 55, 63
}

```

```

Mrow_Scan_8x8[ 64 ] = {
    0, 1, 2, 3, 4, 5, 6, 7,
    8, 9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23,
    24, 25, 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
}

```

```

Default_Scan_8x16[ 128 ] = {
    0, 1, 8, 2, 9, 16, 3, 10, 17, 24, 4, 11, 18, 25, 32,
    5, 12, 19, 26, 33, 40, 6, 13, 20, 27, 34, 41, 48, 7, 14,
    21, 28, 35, 42, 49, 56, 15, 22, 29, 36, 43, 50, 57, 64, 23,
    30, 37, 44, 51, 58, 65, 72, 31, 38, 45, 52, 59, 66, 73, 80,
    39, 46, 53, 60, 67, 74, 81, 88, 47, 54, 61, 68, 75, 82, 89,
    96, 55, 62, 69, 76, 83, 90, 97, 104, 63, 70, 77, 84, 91, 98,
    105, 112, 71, 78, 85, 92, 99, 106, 113, 120, 79, 86, 93, 100, 107,
    114, 121, 87, 94, 101, 108, 115, 122, 95, 102, 109, 116, 123, 103, 110,
    117, 124, 111, 118, 125, 119, 126, 127
}

```

```

Mcol_Scan_8x16[ 128 ] = {
    0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120,
    1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113, 121,
    2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82, 90, 98, 106, 114, 122,
    3, 11, 19, 27, 35, 43, 51, 59, 67, 75, 83, 91, 99, 107, 115, 123,
    4, 12, 20, 28, 36, 44, 52, 60, 68, 76, 84, 92, 100, 108, 116, 124,
    5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93, 101, 109, 117, 125,
    6, 14, 22, 30, 38, 46, 54, 62, 70, 78, 86, 94, 102, 110, 118, 126,
    7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119, 127
}

```



```

Mrow_Scan_8x16[ 128 ] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
    15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 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, 103, 104,
    105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
    120, 121, 122, 123, 124, 125, 126, 127
}

```

```

Default_Scan_16x8[ 128 ] = {
    0, 16, 1, 32, 17, 2, 48, 33, 18, 3, 64, 49, 34, 19, 4, 80,
    65, 50, 35, 20, 5, 96, 81, 66, 51, 36, 21, 6, 112, 97, 82, 67,
    52, 37, 22, 7, 113, 98, 83, 68, 53, 38, 23, 8, 114, 99, 84, 69,
    54, 39, 24, 9, 115, 100, 85, 70, 55, 40, 25, 10, 116, 101, 86, 71,
    56, 41, 26, 11, 117, 102, 87, 72, 57, 42, 27, 12, 118, 103, 88, 73,
    58, 43, 28, 13, 119, 104, 89, 74, 59, 44, 29, 14, 120, 105, 90, 75,
    60, 45, 30, 15, 121, 106, 91, 76, 61, 46, 31, 122, 107, 92, 77, 62,
    47, 123, 108, 93, 78, 63, 124, 109, 94, 79, 125, 110, 95, 126, 111, 127
}

```

```

Mcol_Scan_16x8[ 128 ] = {
    0, 16, 32, 48, 64, 80, 96, 112, 1, 17, 33, 49, 65, 81, 97, 113,
    2, 18, 34, 50, 66, 82, 98, 114, 3, 19, 35, 51, 67, 83, 99, 115,
    4, 20, 36, 52, 68, 84, 100, 116, 5, 21, 37, 53, 69, 85, 101, 117,
    6, 22, 38, 54, 70, 86, 102, 118, 7, 23, 39, 55, 71, 87, 103, 119,
    8, 24, 40, 56, 72, 88, 104, 120, 9, 25, 41, 57, 73, 89, 105, 121,
    10, 26, 42, 58, 74, 90, 106, 122, 11, 27, 43, 59, 75, 91, 107, 123,
    12, 28, 44, 60, 76, 92, 108, 124, 13, 29, 45, 61, 77, 93, 109, 125,
    14, 30, 46, 62, 78, 94, 110, 126, 15, 31, 47, 63, 79, 95, 111, 127
}

```

```

Mrow_Scan_16x8[ 128 ] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
    15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 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, 103, 104,
    105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
    120, 121, 122, 123, 124, 125, 126, 127
}

```

```

Default_Scan_16x16[ 256 ] = {
    0,  1,  16, 32, 17, 2,  3,  18, 33, 48, 64, 49, 34, 19,  4,  5,
    20, 35, 50, 65, 80, 96, 81, 66, 51, 36, 21, 6,  7,  22, 37, 52,
    67, 82, 97, 112, 128, 113, 98, 83, 68, 53, 38, 23, 8,  9,  24, 39,
    54, 69, 84, 99, 114, 129, 144, 160, 145, 130, 115, 100, 85, 70, 55, 40,
    25, 10, 11, 26, 41, 56, 71, 86, 101, 116, 131, 146, 161, 176, 192, 177,
    162, 147, 132, 117, 102, 87, 72, 57, 42, 27, 12, 13, 28, 43, 58, 73,
    88, 103, 118, 133, 148, 163, 178, 193, 208, 224, 209, 194, 179, 164, 149, 134,
    119, 104, 89, 74, 59, 44, 29, 14, 15, 30, 45, 60, 75, 90, 105, 120,
    135, 150, 165, 180, 195, 210, 225, 240, 241, 226, 211, 196, 181, 166, 151, 136,
    121, 106, 91, 76, 61, 46, 31, 47, 62, 77, 92, 107, 122, 137, 152, 167,
    182, 197, 212, 227, 242, 243, 228, 213, 198, 183, 168, 153, 138, 123, 108, 93,
    78, 63, 79, 94, 109, 124, 139, 154, 169, 184, 199, 214, 229, 244, 245, 230,
    215, 200, 185, 170, 155, 140, 125, 110, 95, 111, 126, 141, 156, 171, 186, 201,
    216, 231, 246, 247, 232, 217, 202, 187, 172, 157, 142, 127, 143, 158, 173, 188,
    203, 218, 233, 248, 249, 234, 219, 204, 189, 174, 159, 175, 190, 205, 220, 235,
    250, 251, 236, 221, 206, 191, 207, 222, 237, 252, 253, 238, 223, 239, 254, 255
}

```

```

Mcol_Scan_16x16[ 256 ] = {
    0,  16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240,
    1,  17, 33, 49, 65, 81, 97, 113, 129, 145, 161, 177, 193, 209, 225, 241,
    2,  18, 34, 50, 66, 82, 98, 114, 130, 146, 162, 178, 194, 210, 226, 242,
    3,  19, 35, 51, 67, 83, 99, 115, 131, 147, 163, 179, 195, 211, 227, 243,
    4,  20, 36, 52, 68, 84, 100, 116, 132, 148, 164, 180, 196, 212, 228, 244,
    5,  21, 37, 53, 69, 85, 101, 117, 133, 149, 165, 181, 197, 213, 229, 245,
    6,  22, 38, 54, 70, 86, 102, 118, 134, 150, 166, 182, 198, 214, 230, 246,
    7,  23, 39, 55, 71, 87, 103, 119, 135, 151, 167, 183, 199, 215, 231, 247,
    8,  24, 40, 56, 72, 88, 104, 120, 136, 152, 168, 184, 200, 216, 232, 248,
    9,  25, 41, 57, 73, 89, 105, 121, 137, 153, 169, 185, 201, 217, 233, 249,
    10, 26, 42, 58, 74, 90, 106, 122, 138, 154, 170, 186, 202, 218, 234, 250,
    11, 27, 43, 59, 75, 91, 107, 123, 139, 155, 171, 187, 203, 219, 235, 251,
    12, 28, 44, 60, 76, 92, 108, 124, 140, 156, 172, 188, 204, 220, 236, 252,
    13, 29, 45, 61, 77, 93, 109, 125, 141, 157, 173, 189, 205, 221, 237, 253,
    14, 30, 46, 62, 78, 94, 110, 126, 142, 158, 174, 190, 206, 222, 238, 254,
    15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255
}

```

```

Mrow_Scan_16x16[ 256 ] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
    15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 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, 103, 104,
    105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
    120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
    135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
    150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
    165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
    180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
    195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
    210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
    225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
    240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
    255
}

```

```

Default_Scan_16x32[ 512 ] = {
    0,  1,  16,  2,  17,  32,  3,  18,  33,  48,  4,  19,  34,  49,  64,
    5,  20,  35,  50,  65,  80,  6,  21,  36,  51,  66,  81,  96,  7,  22,
    37,  52,  67,  82,  97,  112, 8,  23,  38,  53,  68,  83,  98,  113, 128,
    9,  24,  39,  54,  69,  84,  99,  114, 129, 144, 10, 25, 40, 55, 70,
    85, 100, 115, 130, 145, 160, 11, 26, 41, 56, 71, 86, 101, 116, 131,
    146, 161, 176, 12, 27, 42, 57, 72, 87, 102, 117, 132, 147, 162, 177,
    192, 13, 28, 43, 58, 73, 88, 103, 118, 133, 148, 163, 178, 193, 208,
    14, 29, 44, 59, 74, 89, 104, 119, 134, 149, 164, 179, 194, 209, 224,
    15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225,
    240, 31, 46, 61, 76, 91, 106, 121, 136, 151, 166, 181, 196, 211, 226,
    241, 256, 47, 62, 77, 92, 107, 122, 137, 152, 167, 182, 197, 212, 227,
    242, 257, 272, 63, 78, 93, 108, 123, 138, 153, 168, 183, 198, 213, 228,
    243, 258, 273, 288, 79, 94, 109, 124, 139, 154, 169, 184, 199, 214, 229,
    244, 259, 274, 289, 304, 95, 110, 125, 140, 155, 170, 185, 200, 215, 230,
    245, 260, 275, 290, 305, 320, 111, 126, 141, 156, 171, 186, 201, 216, 231,
    246, 261, 276, 291, 306, 321, 336, 127, 142, 157, 172, 187, 202, 217, 232,
    247, 262, 277, 292, 307, 322, 337, 352, 143, 158, 173, 188, 203, 218, 233,
    248, 263, 278, 293, 308, 323, 338, 353, 368, 159, 174, 189, 204, 219, 234,
    249, 264, 279, 294, 309, 324, 339, 354, 369, 384, 175, 190, 205, 220, 235,
    250, 265, 280, 295, 310, 325, 340, 355, 370, 385, 400, 191, 206, 221, 236,
    251, 266, 281, 296, 311, 326, 341, 356, 371, 386, 401, 416, 207, 222, 237,
    252, 267, 282, 297, 312, 327, 342, 357, 372, 387, 402, 417, 432, 223, 238,
    253, 268, 283, 298, 313, 328, 343, 358, 373, 388, 403, 418, 433, 448, 239,
    254, 269, 284, 299, 314, 329, 344, 359, 374, 389, 404, 419, 434, 449, 464,
    255, 270, 285, 300, 315, 330, 345, 360, 375, 390, 405, 420, 435, 450, 465,
    480, 271, 286, 301, 316, 331, 346, 361, 376, 391, 406, 421, 436, 451, 466,
    481, 496, 287, 302, 317, 332, 347, 362, 377, 392, 407, 422, 437, 452, 467,
    482, 497, 303, 318, 333, 348, 363, 378, 393, 408, 423, 438, 453, 468, 483,
    498, 319, 334, 349, 364, 379, 394, 409, 424, 439, 454, 469, 484, 499, 335,
    350, 365, 380, 395, 410, 425, 440, 455, 470, 485, 500, 351, 366, 381, 396,
    411, 426, 441, 456, 471, 486, 501, 367, 382, 397, 412, 427, 442, 457, 472,
    487, 502, 383, 398, 413, 428, 443, 458, 473, 488, 503, 399, 414, 429, 444,
    459, 474, 489, 504, 415, 430, 445, 460, 475, 490, 505, 431, 446, 461, 476,
    491, 506, 447, 462, 477, 492, 507, 463, 478, 493, 508, 479, 494, 509, 495,
    510, 511
}

```

```

Default_Scan_32x16[ 512 ] = {
    0,  32,  1,  64,  33,  2,  96,  65,  34,  3,  128, 97,  66,  35,  4,
    160, 129, 98,  67,  36,  5,  192, 161, 130, 99,  68,  37,  6,  224, 193,
    162, 131, 100, 69,  38,  7,  256, 225, 194, 163, 132, 101, 70,  39,  8,
    288, 257, 226, 195, 164, 133, 102, 71,  40,  9,  320, 289, 258, 227, 196,
    165, 134, 103, 72,  41,  10, 352, 321, 290, 259, 228, 197, 166, 135, 104,
    73,  42,  11,  384, 353, 322, 291, 260, 229, 198, 167, 136, 105, 74,  43,
    12,  416, 385, 354, 323, 292, 261, 230, 199, 168, 137, 106, 75,  44,  13,
    448, 417, 386, 355, 324, 293, 262, 231, 200, 169, 138, 107, 76,  45,  14,
    480, 449, 418, 387, 356, 325, 294, 263, 232, 201, 170, 139, 108, 77,  46,
    15,  481, 450, 419, 388, 357, 326, 295, 264, 233, 202, 171, 140, 109, 78,
    47,  16,  482, 451, 420, 389, 358, 327, 296, 265, 234, 203, 172, 141, 110,
    79,  48,  17,  483, 452, 421, 390, 359, 328, 297, 266, 235, 204, 173, 142,
    111, 80,  49,  18,  484, 453, 422, 391, 360, 329, 298, 267, 236, 205, 174,
    143, 112, 81,  50,  19,  485, 454, 423, 392, 361, 330, 299, 268, 237, 206,
    175, 144, 113, 82,  51,  20,  486, 455, 424, 393, 362, 331, 300, 269, 238,
    207, 176, 145, 114, 83,  52,  21,  487, 456, 425, 394, 363, 332, 301, 270,
    239, 208, 177, 146, 115, 84,  53,  22,  488, 457, 426, 395, 364, 333, 302,
    271, 240, 209, 178, 147, 116, 85,  54,  23,  489, 458, 427, 396, 365, 334,
    303, 272, 241, 210, 179, 148, 117, 86,  55,  24,  490, 459, 428, 397, 366,
    335, 304, 273, 242, 211, 180, 149, 118, 87,  56,  25,  491, 460, 429, 398,
    367, 336, 305, 274, 243, 212, 181, 150, 119, 88,  57,  26,  492, 461, 430,
    399, 368, 337, 306, 275, 244, 213, 182, 151, 120, 89,  58,  27,  493, 462,
    431, 400, 369, 338, 307, 276, 245, 214, 183, 152, 121, 90,  59,  28,  494,
    463, 432, 401, 370, 339, 308, 277, 246, 215, 184, 153, 122, 91,  60,  29,
    495, 464, 433, 402, 371, 340, 309, 278, 247, 216, 185, 154, 123, 92,  61,
    30,  496, 465, 434, 403, 372, 341, 310, 279, 248, 217, 186, 155, 124, 93,
    62,  31,  497, 466, 435, 404, 373, 342, 311, 280, 249, 218, 187, 156, 125,
    94,  63,  498, 467, 436, 405, 374, 343, 312, 281, 250, 219, 188, 157, 126,
    95,  499, 468, 437, 406, 375, 344, 313, 282, 251, 220, 189, 158, 127, 500,
    469, 438, 407, 376, 345, 314, 283, 252, 221, 190, 159, 501, 470, 439, 408,
    377, 346, 315, 284, 253, 222, 191, 502, 471, 440, 409, 378, 347, 316, 285,
    254, 223, 503, 472, 441, 410, 379, 348, 317, 286, 255, 504, 473, 442, 411,
    380, 349, 318, 287, 505, 474, 443, 412, 381, 350, 319, 506, 475, 444, 413,
    382, 351, 507, 476, 445, 414, 383, 508, 477, 446, 415, 509, 478, 447, 510,
    479, 511
}

```

```

Default_Scan_32x32[ 1024 ] = {
    0,  1,  32,  64,  33,  2,  3,  34,  65,  96,  128,  97,  66,
    35,  4,  5,  36,  67,  98,  129,  160,  192,  161,  130,  99,  68,
    37,  6,  7,  38,  69,  100,  131,  162,  193,  224,  256,  225,  194,
    163,  132,  101,  70,  39,  8,  9,  40,  71,  102,  133,  164,  195,
    226,  257,  288,  320,  289,  258,  227,  196,  165,  134,  103,  72,  41,
    10,  11,  42,  73,  104,  135,  166,  197,  228,  259,  290,  321,  352,
    384,  353,  322,  291,  260,  229,  198,  167,  136,  105,  74,  43,  12,
    13,  44,  75,  106,  137,  168,  199,  230,  261,  292,  323,  354,  385,
    416,  448,  417,  386,  355,  324,  293,  262,  231,  200,  169,  138,  107,
    76,  45,  14,  15,  46,  77,  108,  139,  170,  201,  232,  263,  294,
    325,  356,  387,  418,  449,  480,  512,  481,  450,  419,  388,  357,  326,
    295,  264,  233,  202,  171,  140,  109,  78,  47,  16,  17,  48,  79,
    110,  141,  172,  203,  234,  265,  296,  327,  358,  389,  420,  451,  482,
    513,  544,  576,  545,  514,  483,  452,  421,  390,  359,  328,  297,  266,
    235,  204,  173,  142,  111,  80,  49,  18,  19,  50,  81,  112,  143,
    174,  205,  236,  267,  298,  329,  360,  391,  422,  453,  484,  515,  546,
    577,  608,  640,  609,  578,  547,  516,  485,  454,  423,  392,  361,  330,
    299,  268,  237,  206,  175,  144,  113,  82,  51,  20,  21,  52,  83,
    114,  145,  176,  207,  238,  269,  300,  331,  362,  393,  424,  455,  486,
    517,  548,  579,  610,  641,  672,  704,  673,  642,  611,  580,  549,  518,
    487,  456,  425,  394,  363,  332,  301,  270,  239,  208,  177,  146,  115,
    84,  53,  22,  23,  54,  85,  116,  147,  178,  209,  240,  271,  302,
    333,  364,  395,  426,  457,  488,  519,  550,  581,  612,  643,  674,  705,
    736,  768,  737,  706,  675,  644,  613,  582,  551,  520,  489,  458,  427,
    396,  365,  334,  303,  272,  241,  210,  179,  148,  117,  86,  55,  24,
    25,  56,  87,  118,  149,  180,  211,  242,  273,  304,  335,  366,  397,
    428,  459,  490,  521,  552,  583,  614,  645,  676,  707,  738,  769,  800,
    832,  801,  770,  739,  708,  677,  646,  615,  584,  553,  522,  491,  460,
    429,  398,  367,  336,  305,  274,  243,  212,  181,  150,  119,  88,  57,
    26,  27,  58,  89,  120,  151,  182,  213,  244,  275,  306,  337,  368,
    399,  430,  461,  492,  523,  554,  585,  616,  647,  678,  709,  740,  771,
    802,  833,  864,  896,  865,  834,  803,  772,  741,  710,  679,  648,  617,
    586,  555,  524,  493,  462,  431,  400,  369,  338,  307,  276,  245,  214,
    183,  152,  121,  90,  59,  28,  29,  60,  91,  122,  153,  184,  215,
    246,  277,  308,  339,  370,  401,  432,  463,  494,  525,  556,  587,  618,
    649,  680,  711,  742,  773,  804,  835,  866,  897,  928,  960,  929,  898,
    867,  836,  805,  774,  743,  712,  681,  650,  619,  588,  557,  526,  495,
    464,  433,  402,  371,  340,  309,  278,  247,  216,  185,  154,  123,  92,
    61,  30,  31,  62,  93,  124,  155,  186,  217,  248,  279,  310,  341,
    372,  403,  434,  465,  496,  527,  558,  589,  620,  651,  682,  713,  744,
    775,  806,  837,  868,  899,  930,  961,  992,  993,  962,  931,  900,  869,
    838,  807,  776,  745,  714,  683,  652,  621,  590,  559,  528,  497,  466,
    435,  404,  373,  342,  311,  280,  249,  218,  187,  156,  125,  94,  63,
    95,  126,  157,  188,  219,  250,  281,  312,  343,  374,  405,  436,  467,
    498,  529,  560,  591,  622,  653,  684,  715,  746,  777,  808,  839,  870,
    901,  932,  963,  994,  995,  964,  933,  902,  871,  840,  809,  778,  747,
    716,  685,  654,  623,  592,  561,  530,  499,  468,  437,  406,  375,  344,
    313,  282,  251,  220,  189,  158,  127,  159,  190,  221,  252,  283,  314,
    345,  376,  407,  438,  469,  500,  531,  562,  593,  624,  655,  686,  717,

```

```

748, 779, 810, 841, 872, 903, 934, 965, 996, 997, 966, 935, 904,
873, 842, 811, 780, 749, 718, 687, 656, 625, 594, 563, 532, 501,
470, 439, 408, 377, 346, 315, 284, 253, 222, 191, 223, 254, 285,
316, 347, 378, 409, 440, 471, 502, 533, 564, 595, 626, 657, 688,
719, 750, 781, 812, 843, 874, 905, 936, 967, 998, 999, 968, 937,
906, 875, 844, 813, 782, 751, 720, 689, 658, 627, 596, 565, 534,
503, 472, 441, 410, 379, 348, 317, 286, 255, 287, 318, 349, 380,
411, 442, 473, 504, 535, 566, 597, 628, 659, 690, 721, 752, 783,
814, 845, 876, 907, 938, 969, 1000, 1001, 970, 939, 908, 877, 846,
815, 784, 753, 722, 691, 660, 629, 598, 567, 536, 505, 474, 443,
412, 381, 350, 319, 351, 382, 413, 444, 475, 506, 537, 568, 599,
630, 661, 692, 723, 754, 785, 816, 847, 878, 909, 940, 971, 1002,
1003, 972, 941, 910, 879, 848, 817, 786, 755, 724, 693, 662, 631,
600, 569, 538, 507, 476, 445, 414, 383, 415, 446, 477, 508, 539,
570, 601, 632, 663, 694, 725, 756, 787, 818, 849, 880, 911, 942,
973, 1004, 1005, 974, 943, 912, 881, 850, 819, 788, 757, 726, 695,
664, 633, 602, 571, 540, 509, 478, 447, 479, 510, 541, 572, 603,
634, 665, 696, 727, 758, 789, 820, 851, 882, 913, 944, 975, 1006,
1007, 976, 945, 914, 883, 852, 821, 790, 759, 728, 697, 666, 635,
604, 573, 542, 511, 543, 574, 605, 636, 667, 698, 729, 760, 791,
822, 853, 884, 915, 946, 977, 1008, 1009, 978, 947, 916, 885, 854,
823, 792, 761, 730, 699, 668, 637, 606, 575, 607, 638, 669, 700,
731, 762, 793, 824, 855, 886, 917, 948, 979, 1010, 1011, 980, 949,
918, 887, 856, 825, 794, 763, 732, 701, 670, 639, 671, 702, 733,
764, 795, 826, 857, 888, 919, 950, 981, 1012, 1013, 982, 951, 920,
889, 858, 827, 796, 765, 734, 703, 735, 766, 797, 828, 859, 890,
921, 952, 983, 1014, 1015, 984, 953, 922, 891, 860, 829, 798, 767,
799, 830, 861, 892, 923, 954, 985, 1016, 1017, 986, 955, 924, 893,
862, 831, 863, 894, 925, 956, 987, 1018, 1019, 988, 957, 926, 895,
927, 958, 989, 1020, 1021, 990, 959, 991, 1022, 1023
}

```

```

Default_Scan_4x16[ 64 ] = {
    0,  1,  4,  2,  5,  8,  3,  6,  9, 12, 7, 10, 13, 16, 11, 14,
    17, 20, 15, 18, 21, 24, 19, 22, 25, 28, 23, 26, 29, 32, 27, 30,
    33, 36, 31, 34, 37, 40, 35, 38, 41, 44, 39, 42, 45, 48, 43, 46,
    49, 52, 47, 50, 53, 56, 51, 54, 57, 60, 55, 58, 61, 59, 62, 63
}

```

```

Mcol_Scan_4x16[ 64 ] = {
    0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60,
    1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61,
    2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62,
    3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63
}

```

```

Mrow_Scan_4x16[ 64 ] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 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
}

```

```

Default_Scan_16x4[ 64 ] = {
    0,  16, 1,  32, 17, 2,  48, 33, 18, 3,  49, 34, 19, 4,  50, 35,
    20, 5,  51, 36, 21, 6,  52, 37, 22, 7,  53, 38, 23, 8,  54, 39,
    24, 9,  55, 40, 25, 10, 56, 41, 26, 11, 57, 42, 27, 12, 58, 43,
    28, 13, 59, 44, 29, 14, 60, 45, 30, 15, 61, 46, 31, 62, 47, 63
}

```

```

Mcol_Scan_16x4[ 64 ] = {
    0,  16, 32, 48, 1,  17, 33, 49, 2,  18, 34, 50, 3,  19, 35, 51,
    4,  20, 36, 52, 5,  21, 37, 53, 6,  22, 38, 54, 7,  23, 39, 55,
    8,  24, 40, 56, 9,  25, 41, 57, 10, 26, 42, 58, 11, 27, 43, 59,
    12, 28, 44, 60, 13, 29, 45, 61, 14, 30, 46, 62, 15, 31, 47, 63
}

```

```

Mrow_Scan_16x4[ 64 ] = {
    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 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
}

```



```

Default_Scan_8x32[ 256 ] = {
    0,  1,  8,  2,  9,  16, 3,  10, 17, 24, 4,  11, 18, 25, 32,
    5,  12, 19, 26, 33, 40, 6,  13, 20, 27, 34, 41, 48, 7,  14,
    21, 28, 35, 42, 49, 56, 15, 22, 29, 36, 43, 50, 57, 64, 23,
    30, 37, 44, 51, 58, 65, 72, 31, 38, 45, 52, 59, 66, 73, 80,
    39, 46, 53, 60, 67, 74, 81, 88, 47, 54, 61, 68, 75, 82, 89,
    96, 55, 62, 69, 76, 83, 90, 97, 104, 63, 70, 77, 84, 91, 98,
    105, 112, 71, 78, 85, 92, 99, 106, 113, 120, 79, 86, 93, 100, 107,
    114, 121, 128, 87, 94, 101, 108, 115, 122, 129, 136, 95, 102, 109, 116,
    123, 130, 137, 144, 103, 110, 117, 124, 131, 138, 145, 152, 111, 118, 125,
    132, 139, 146, 153, 160, 119, 126, 133, 140, 147, 154, 161, 168, 127, 134,
    141, 148, 155, 162, 169, 176, 135, 142, 149, 156, 163, 170, 177, 184, 143,
    150, 157, 164, 171, 178, 185, 192, 151, 158, 165, 172, 179, 186, 193, 200,
    159, 166, 173, 180, 187, 194, 201, 208, 167, 174, 181, 188, 195, 202, 209,
    216, 175, 182, 189, 196, 203, 210, 217, 224, 183, 190, 197, 204, 211, 218,
    225, 232, 191, 198, 205, 212, 219, 226, 233, 240, 199, 206, 213, 220, 227,
    234, 241, 248, 207, 214, 221, 228, 235, 242, 249, 215, 222, 229, 236, 243,
    250, 223, 230, 237, 244, 251, 231, 238, 245, 252, 239, 246, 253, 247, 254,
    255
}

```

```

Default_Scan_32x8[ 256 ] = {
    0,  32, 1,  64, 33, 2,  96, 65, 34, 3,  128, 97, 66, 35, 4,
    160, 129, 98, 67, 36, 5,  192, 161, 130, 99, 68, 37, 6,  224, 193,
    162, 131, 100, 69, 38, 7,  225, 194, 163, 132, 101, 70, 39, 8,  226,
    195, 164, 133, 102, 71, 40, 9,  227, 196, 165, 134, 103, 72, 41, 10,
    228, 197, 166, 135, 104, 73, 42, 11, 229, 198, 167, 136, 105, 74, 43,
    12, 230, 199, 168, 137, 106, 75, 44, 13, 231, 200, 169, 138, 107, 76,
    45, 14, 232, 201, 170, 139, 108, 77, 46, 15, 233, 202, 171, 140, 109,
    78, 47, 16, 234, 203, 172, 141, 110, 79, 48, 17, 235, 204, 173, 142,
    111, 80, 49, 18, 236, 205, 174, 143, 112, 81, 50, 19, 237, 206, 175,
    144, 113, 82, 51, 20, 238, 207, 176, 145, 114, 83, 52, 21, 239, 208,
    177, 146, 115, 84, 53, 22, 240, 209, 178, 147, 116, 85, 54, 23, 241,
    210, 179, 148, 117, 86, 55, 24, 242, 211, 180, 149, 118, 87, 56, 25,
    243, 212, 181, 150, 119, 88, 57, 26, 244, 213, 182, 151, 120, 89, 58,
    27, 245, 214, 183, 152, 121, 90, 59, 28, 246, 215, 184, 153, 122, 91,
    60, 29, 247, 216, 185, 154, 123, 92, 61, 30, 248, 217, 186, 155, 124,
    93, 62, 31, 249, 218, 187, 156, 125, 94, 63, 250, 219, 188, 157, 126,
    95, 251, 220, 189, 158, 127, 252, 221, 190, 159, 253, 222, 191, 254, 223,
    255
}

```

## 9.3. Conversion tables

This section defines the constant lookup tables used to convert between different representations.

For a block size  $x$  (with values having the same interpretation as for the variable `subSize`), `Mi_Width_Log2[  $x$  ]` gives the base 2 logarithm of the width of the block in units of 4 samples.

```
Mi_Width_Log2[ BLOCK_SIZES ] = {
    0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3,
    4, 4, 4, 5, 5, 0, 2, 1, 3, 2, 4
}
```

For a block size  $x$ ,  $Mi\_Height\_Log2[x]$  gives the base 2 logarithm of the height of the block in units of 4 samples.

```
Mi_Height_Log2[ BLOCK_SIZES ] = {
    0, 1, 0, 1, 2, 1, 2, 3, 2, 3, 4,
    3, 4, 5, 4, 5, 2, 0, 3, 1, 4, 2
}
```

For a block size  $x$ ,  $Num\_4x4\_Blocks\_Wide[x]$  gives the width of the block in units of 4 samples.

```
Num_4x4_Blocks_Wide[ BLOCK_SIZES ] = {
    1, 1, 2, 2, 2, 4, 4, 4, 8, 8, 8,
    16, 16, 16, 32, 32, 1, 4, 2, 8, 4, 16
}
```

For a block size  $x$ ,  $Block\_Width[x]$  gives the width of the block in units of samples.  $Block\_Width[x]$  is defined to be equal to  $4 * Num\_4x4\_Blocks\_Wide[x]$ .

For a block size  $x$ ,  $Num\_4x4\_Blocks\_High[x]$  gives the the height of the block in units of 4 samples.

```
Num_4x4_Blocks_High[ BLOCK_SIZES ] = {
    1, 2, 1, 2, 4, 2, 4, 8, 4, 8, 16,
    8, 16, 32, 16, 32, 4, 1, 8, 2, 16, 4
}
```

For a block size  $x$ ,  $Block\_Height[x]$  gives the width of the block in units of samples.  $Block\_Height[x]$  is defined to be equal to  $4 * Num\_4x4\_Blocks\_High[x]$ .

$Size\_Group$  is used to map a block size into a context for intra syntax elements.

```
Size_Group[ BLOCK_SIZES ] = {
    0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3,
    3, 3, 3, 3, 3, 0, 0, 1, 1, 2, 2
}
```

For a luma block size  $x$ ,  $Max\_Tx\_Size\_Rect[x]$  returns the largest transform size that can be used for blocks of size  $x$  (this can be either square or rectangular).

```

Max_Tx_Size_Rect[ BLOCK_SIZES ] = {
    TX_4X4, TX_4X8, TX_8X4, TX_8X8,
    TX_8X16, TX_16X8, TX_16X16, TX_16X32,
    TX_32X16, TX_32X32, TX_32X64, TX_64X32,
    TX_64X64, TX_64X64, TX_64X64, TX_64X64,
    TX_4X16, TX_16X4, TX_8X32, TX_32X8,
    TX_16X64, TX_64X16
}

```

For a square block size  $x$ , and a partition type  $p$ , `Partition_Subsize[  $p$  ][  $x$  ]` returns the size of the sub-blocks used by this partition. (If the partition produces blocks of different sizes, then the table contains the largest sub-block size.)

The table will never get accessed for rectangular block sizes.

```

Partition_Subsize[ 10 ][ BLOCK_SIZES ] = {
{
    BLOCK_4X4,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X8,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X16,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X32,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X64,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_128X128,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
    BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X4,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X8,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X16,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X32,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_128X64,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
    BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_4X8,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X16,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X32,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X64,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X128,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
    BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_4X4,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X8,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X16,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X32,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X64,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
    BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X4,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X8,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X16,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X32,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_128X64,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
    BLOCK_INVALID,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X4,
    BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X8,

```

```

BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X16,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X32,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_128X64,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
                                BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_4X8,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X16,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X32,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X64,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X128,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
                                BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_4X8,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X16,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X32,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X64,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X128,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
                                BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X4,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_32X8,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_64X16,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}, {
                                BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_4X16,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_8X32,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_16X64,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID,
BLOCK_INVALID, BLOCK_INVALID, BLOCK_INVALID
}
}

```

```

Split_Tx_Size[ TX_SIZES_ALL ] = {
    TX_4X4,
    TX_4X4,
    TX_8X8,
    TX_16X16,
    TX_32X32,
    TX_4X4,
    TX_4X4,
    TX_8X8,
    TX_8X8,
    TX_16X16,
    TX_16X16,
    TX_32X32,
    TX_32X32
    TX_4X8,
    TX_8X4,
    TX_8X16,
    TX_16X8,
    TX_16X32,
    TX_32X16
}

```

```

Mode_To_Txfrm[ UV_INTRA_MODES_CFL_ALLOWED ] = {
    DCT_DCT, // DC
    ADST_DCT, // V
    DCT_ADST, // H
    DCT_DCT, // D45
    ADST_ADST, // D135
    ADST_DCT, // D117
    DCT_ADST, // D153
    DCT_ADST, // D207
    ADST_DCT, // D63
    ADST_ADST, // SMOOTH
    ADST_DCT, // SMOOTH_V
    DCT_ADST, // SMOOTH_H
    ADST_ADST, // TM
    DCT_DCT, // UV_CFL_PRED
}

```

```

Palette_Color_Context[ PALETTE_MAX_COLOR_CONTEXT_HASH + 1 ] =
{ -1, -1, 0, -1, -1, 4, 3, 2, 1 }

```

**Note:** The negative numbers in the array `Palette_Color_Context` indicate values that will never be accessed.

```
Palette_Color_Hash_Multipliers[ PALETTE_NUM_NEIGHBORS ] = { 1, 2, 2 }
```

```
Sm_Weights_Tx_4x4  [ 4 ] = { 255, 149, 85, 64 }
Sm_Weights_Tx_8x8  [ 8 ] = { 255, 197, 146, 105, 73, 50, 37, 32 }
Sm_Weights_Tx_16x16[ 16 ] = { 255, 225, 196, 170, 145, 123, 102, 84, 68, 54, 43, 33, 26, 20,
17, 16 }
Sm_Weights_Tx_32x32[ 32 ] = { 255, 240, 225, 210, 196, 182, 169, 157, 145, 133, 122, 111, 101, 92,
83, 74,
                                66, 59, 52, 45, 39, 34, 29, 25, 21, 17, 14, 12, 10, 9,
8, 8 }
Sm_Weights_Tx_64x64[ 64 ] = { 255, 248, 240, 233, 225, 218, 210, 203, 196, 189, 182, 176, 169, 163,
156,
                                150, 144, 138, 133, 127, 121, 116, 111, 106, 101, 96, 91, 86, 82, 77,
73, 69,
                                65, 61, 57, 54, 50, 47, 44, 41, 38, 35, 32, 29, 27, 25, 22, 20, 18, 16,
15,
                                13, 12, 10, 9, 8, 7, 6, 6, 5, 5, 4, 4, 4 }
```

```
Mode_To_Angle[ INTRA_MODES ] = { 0, 90, 180, 45, 135, 113, 157, 203, 67, 0, 0, 0, 0 }
```

```
Dr_Intra_Derivative[ 90 ] = {
    0, 0, 0, 1023, 0, 0, 547, 0, 0, 372, 0, 0, 0, 0,
    273, 0, 0, 215, 0, 0, 178, 0, 0, 151, 0, 0, 132, 0, 0,
    116, 0, 0, 102, 0, 0, 0, 90, 0, 0, 80, 0, 0, 71, 0, 0,
    64, 0, 0, 57, 0, 0, 51, 0, 0, 45, 0, 0, 40, 0, 0,
    35, 0, 0, 31, 0, 0, 27, 0, 0, 23, 0, 0, 19, 0, 0,
    15, 0, 0, 0, 0, 11, 0, 0, 7, 0, 0, 3, 0, 0
}
```

```

Intra_Filter_Taps[ INTRA_FILTER_MODES ][ 8 ][ 7 ] = {
{
    { -6, 10, 0, 0, 0, 12, 0 },
    { -5, 2, 10, 0, 0, 9, 0 },
    { -3, 1, 1, 10, 0, 7, 0 },
    { -3, 1, 1, 2, 10, 5, 0 },
    { -4, 6, 0, 0, 0, 2, 12 },
    { -3, 2, 6, 0, 0, 2, 9 },
    { -3, 2, 2, 6, 0, 2, 7 },
    { -3, 1, 2, 2, 6, 3, 5 },
},
{
    { -10, 16, 0, 0, 0, 10, 0 },
    { -6, 0, 16, 0, 0, 6, 0 },
    { -4, 0, 0, 16, 0, 4, 0 },
    { -2, 0, 0, 0, 16, 2, 0 },
    { -10, 16, 0, 0, 0, 0, 10 },
    { -6, 0, 16, 0, 0, 0, 6 },
    { -4, 0, 0, 16, 0, 0, 4 },
    { -2, 0, 0, 0, 16, 0, 2 },
},
{
    { -8, 8, 0, 0, 0, 16, 0 },
    { -8, 0, 8, 0, 0, 16, 0 },
    { -8, 0, 0, 8, 0, 16, 0 },
    { -8, 0, 0, 0, 8, 16, 0 },
    { -4, 4, 0, 0, 0, 0, 16 },
    { -4, 0, 4, 0, 0, 0, 16 },
    { -4, 0, 0, 4, 0, 0, 16 },
    { -4, 0, 0, 0, 4, 0, 16 },
},
{
    { -2, 8, 0, 0, 0, 10, 0 },
    { -1, 3, 8, 0, 0, 6, 0 },
    { -1, 2, 3, 8, 0, 4, 0 },
    { 0, 1, 2, 3, 8, 2, 0 },
    { -1, 4, 0, 0, 0, 3, 10 },
    { -1, 3, 4, 0, 0, 4, 6 },
    { -1, 2, 3, 4, 0, 4, 4 },
    { -1, 2, 2, 3, 4, 3, 3 },
},
{
    { -12, 14, 0, 0, 0, 14, 0 },
    { -10, 0, 14, 0, 0, 12, 0 },
    { -9, 0, 0, 14, 0, 11, 0 },
    { -8, 0, 0, 0, 14, 10, 0 },
    { -10, 12, 0, 0, 0, 0, 14 },
    { -9, 1, 12, 0, 0, 0, 12 },
    { -8, 0, 0, 12, 0, 1, 11 },
    { -7, 0, 0, 1, 12, 1, 9 },
}
}

```



```

    }
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ) (with the same interpretation as for the `TxSize` variable), `Tx_Size_Sqr[  $t$  ]` returns a square tx size with side length  $\text{Min}(w, h)$ .

```

Tx_Size_Sqr[ TX_SIZES_ALL ] = {
    TX_4X4,
    TX_8X8,
    TX_16X16,
    TX_32X32,
    TX_64X64,
    TX_4X4,
    TX_4X4,
    TX_8X8,
    TX_8X8,
    TX_16X16,
    TX_16X16,
    TX_32X32,
    TX_32X32,
    TX_4X4,
    TX_4X4,
    TX_8X8,
    TX_8X8,
    TX_16X16,
    TX_16X16
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ), `Tx_Size_Sqr_Up[  $t$  ]` returns a square tx size with side length  $\text{Max}(w, h)$ .

```

Tx_Size_Sqr_Up[ TX_SIZES_ALL ] = {
    TX_4X4,
    TX_8X8,
    TX_16X16,
    TX_32X32,
    TX_64X64,
    TX_8X8,
    TX_8X8,
    TX_16X16,
    TX_16X16,
    TX_32X32,
    TX_32X32,
    TX_64X64,
    TX_64X64,
    TX_16X16,
    TX_16X16,
    TX_32X32,
    TX_32X32,
    TX_64X64,
    TX_64X64
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ),  $Tx\_Width[t]$  returns  $w$ .

```

Tx_Width[ TX_SIZES_ALL ] = {
    4, 8, 16, 32, 64, 4, 8, 8, 16, 16, 32, 32, 64, 4, 16, 8, 32, 16, 64
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ),  $Tx\_Height[t]$  returns  $h$ .

```

Tx_Height[ TX_SIZES_ALL ] = {
    4, 8, 16, 32, 64, 8, 4, 16, 8, 32, 16, 64, 32, 16, 4, 32, 8, 64, 16
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ),  $Tx\_Width\_Log2[t]$  returns the base 2 logarithm of  $w$ .

```

Tx_Width_Log2[ TX_SIZES_ALL ] = {
    2, 3, 4, 5, 6, 2, 3, 3, 4, 4, 5, 5, 6, 2, 4, 3, 5, 4, 6
}

```

For a transform size  $t$  (of width  $w$  and height  $h$ ),  $Tx\_Height\_Log2[t]$  returns the base 2 logarithm of  $h$ .

```
Tx_Height_Log2[ TX_SIZES_ALL ] = {
    2, 3, 4, 5, 6, 3, 2, 4, 3, 5, 4, 6, 5, 4, 2, 5, 3, 6, 4
}
```

```
Wedge_Bits[ BLOCK_SIZES ] = {
    0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0,
    0, 0, 0, 0, 0, 0, 4, 4, 0, 0
}
```

For a transform size *t* (of width *w* and height *h*), `Transpose_Tx_Size[ t ]` returns a transform size with width *h* and height *w*.

```
Transpose_Tx_Size[ TX_SIZES_ALL ] = {
    TX_4X4, TX_8X8, TX_16X16, TX_32X32, TX_64X64, TX_8X4, TX_4X8,
    TX_16X8, TX_8X16, TX_32X16, TX_16X32, TX_64X32, TX_32X64,
    TX_16X4, TX_4X16, TX_32X8, TX_8X32, TX_64X16, TX_16X64
}
```

For a transform type *t*, `Transpose_Tx_Type[ t ]` returns a transform type with the horizontal and vertical operations transposed.

```
Transpose_Tx_Type[ TX_TYPES ] = {
    DCT_DCT, DCT_ADST, ADST_DCT, ADST_ADST, DCT_FLIPADST, FLIPADST_DCT,
    FLIPADST_FLIPADST, FLIPADST_ADST, ADST_FLIPADST, IDTX, H_DCT, V_DCT,
    H_ADST, V_ADST, H_FLIPADST, V_FLIPADST,
}
```

```
Sig_Ref_Diff_Offset[ 3 ][ SIG_REF_DIFF_OFFSET_NUM ][ 2 ] = {
    {
        { 0, 1 }, { 1, 0 }, { 1, 1 }, { 0, 2 }, { 2, 0 }
    },
    {
        { 0, 1 }, { 1, 0 }, { 0, 2 }, { 0, 3 }, { 0, 4 }
    },
    {
        { 0, 1 }, { 1, 0 }, { 2, 0 }, { 3, 0 }, { 4, 0 }
    }
}
```

```
Adjusted_Tx_Size[ TX_SIZES_ALL ] = {
    TX_4X4,
    TX_8X8,
    TX_16X16,
    TX_32X32,
    TX_32X32,
    TX_4X8,
    TX_8X4,
    TX_8X16,
    TX_16X8,
    TX_16X32,
    TX_32X16,
    TX_32X32,
    TX_32X32,
    TX_4X16,
    TX_16X4,
    TX_8X32,
    TX_32X8,
    TX_16X32,
    TX_32X16
}
```

The array `Gaussian_Sequence` contains random samples from a Gaussian distribution with zero mean and standard deviation of about 512 clipped to the range of `[-2048, 2047]` and rounded to the nearest multiple of 4.

```

Gaussian_Sequence[ 2048 ] = {
    56,    568,   -180,   172,   124,   -84,   172,   -64,   -900,   24,   820,
    224,   1248,   996,   272,    -8,   -916,  -388,  -732,  -104,  -188,   800,
    112,   -652,  -320,  -376,   140,  -252,   492,  -168,   44,   -788,   588,
   -584,   500,  -228,   12,   680,   272,  -476,   972,  -100,   652,   368,
    432,  -196,  -720,  -192,  1000,  -332,   652,  -136,  -552,  -604,    -4,
    192,  -220,  -136,  1000,   -52,   372,   -96,  -624,   124,   -24,   396,
    540,   -12,  -104,   640,   464,   244,  -208,   -84,   368,  -528,  -740,
    248,  -968,  -848,   608,   376,   -60,  -292,   -40,  -156,   252,  -292,
    248,   224,  -280,   400,  -244,   244,   -60,    76,   -80,   212,   532,
    340,   128,   -36,   824,  -352,   -60,  -264,   -96,  -612,   416,  -704,
    220,  -204,   640,  -160,  1220,  -408,   900,   336,    20,  -336,   -96,
   -792,   304,    48,   -28,  -1232, -1172,  -448,   104,  -292,  -520,   244,
    60,   -948,    0,   -708,   268,   108,   356,  -548,   488,  -344,  -136,
    488,  -196,  -224,   656,  -236, -1128,    60,    4,   140,   276,  -676,
   -376,   168,  -108,   464,    8,   564,    64,   240,   308,  -300,  -400,
   -456,  -136,    56,   120,  -408,  -116,   436,   504,  -232,   328,   844,
   -164,   -84,   784,  -168,   232,  -224,   348,  -376,   128,   568,    96,
  -1244,  -288,   276,   848,   832,  -360,   656,   464,  -384,  -332,  -356,
    728,  -388,   160,  -192,   468,   296,   224,   140,  -776,  -100,   280,
    4,   196,   44,   -36,  -648,   932,   16,  1428,   28,   528,   808,
    772,    20,   268,   88,  -332,  -284,   124,  -384,  -448,   208,  -228,
  -1044,  -328,   660,   380,  -148,  -300,   588,   240,   540,   28,   136,
   -88,  -436,   256,   296, -1000,  1400,    0,   -48,  1056,  -136,   264,
   -528, -1108,   632,  -484,  -592,  -344,   796,   124,  -668,  -768,   388,
  1296,  -232,  -188,  -200,  -288,   -4,   308,   100,  -168,   256,  -500,
    204,  -508,   648,  -136,   372,  -272,  -120, -1004,  -552,  -548,  -384,
    548,  -296,   428,  -108,   -8,  -912,  -324,  -224,   -88,  -112,  -220,
   -100,   996,  -796,   548,   360,  -216,   180,   428,  -200,  -212,   148,
    96,   148,   284,   216,  -412,  -320,   120,  -300,  -384,  -604,  -572,
   -332,   -8,  -180,  -176,   696,   116,   -88,   628,    76,    44,  -516,
    240,  -208,   -40,   100,  -592,   344,  -308,  -452,  -228,    20,   916,
  -1752,  -136,  -340,  -804,   140,    40,   512,   340,   248,   184,  -492,
    896,  -156,   932,  -628,   328,  -688,  -448,  -616,  -752,  -100,   560,
  -1020,   180,  -800,   -64,    76,   576,  1068,   396,   660,   552,  -108,
   -28,   320,  -628,   312,   -92,   -92,  -472,   268,    16,   560,   516,
  -672,   -52,   492,  -100,   260,   384,   284,   292,   304,  -148,    88,
  -152,  1012,  1064,  -228,   164,  -376,  -684,   592,  -392,   156,   196,
  -524,   -64,  -884,   160,  -176,   636,   648,   404,  -396,  -436,   864,
    424,  -728,   988,  -604,   904,  -592,   296,  -224,   536,  -176,  -920,
    436,   -48,  1176,  -884,   416,  -776,  -824,  -884,   524,  -548,  -564,
   -68,  -164,   -96,   692,   364,  -692, -1012,   -68,   260,  -480,   876,
  -1116,   452,  -332,  -352,   892, -1088,  1220,  -676,    12,  -292,   244,
    496,   372,   -32,   280,   200,   112,  -440,   -96,    24,  -644,  -184,
    56,  -432,   224,  -980,   272,  -260,   144,  -436,   420,   356,   364,
   -528,    76,   172,  -744,  -368,   404,  -752,  -416,   684,  -688,    72,
    540,   416,    92,   444,   480,   -72, -1416,   164, -1172,   -68,    24,
    424,   264,  1040,   128,  -912,  -524,  -356,    64,   876,   -12,    4,
   -88,   532,   272,  -524,   320,   276,  -508,   940,    24,  -400,  -120,
    756,    60,   236,  -412,   100,   376,  -484,   400,  -100,  -740,  -108,

```

```

-260, 328, -268, 224, -200, -416, 184, -604, -564, -20, 296,
60, 892, -888, 60, 164, 68, -760, 216, -296, 904, -336,
-28, 404, -356, -568, -208, -1480, -512, 296, 328, -360, -164,
-1560, -776, 1156, -428, 164, -504, -112, 120, -216, -148, -264,
308, 32, 64, -72, 72, 116, 176, -64, -272, 460, -536,
-784, -280, 348, 108, -752, -132, 524, -540, -776, 116, -296,
-1196, -288, -560, 1040, -472, 116, -848, -1116, 116, 636, 696,
284, -176, 1016, 204, -864, -648, -248, 356, 972, -584, -204,
264, 880, 528, -24, -184, 116, 448, -144, 828, 524, 212,
-212, 52, 12, 200, 268, -488, -404, -880, 824, -672, -40,
908, -248, 500, 716, -576, 492, -576, 16, 720, -108, 384,
124, 344, 280, 576, -500, 252, 104, -308, 196, -188, -8,
1268, 296, 1032, -1196, 436, 316, 372, -432, -200, -660, 704,
-224, 596, -132, 268, 32, -452, 884, 104, -1008, 424, -1348,
-280, 4, -1168, 368, 476, 696, 300, -8, 24, 180, -592,
-196, 388, 304, 500, 724, -160, 244, -84, 272, -256, -420,
320, 208, -144, -156, 156, 364, 452, 28, 540, 316, 220,
-644, -248, 464, 72, 360, 32, -388, 496, -680, -48, 208,
-116, -408, 60, -604, -392, 548, -840, 784, -460, 656, -544,
-388, -264, 908, -800, -628, -612, -568, 572, -220, 164, 288,
-16, -308, 308, -112, -636, -760, 280, -668, 432, 364, 240,
-196, 604, 340, 384, 196, 592, -44, -500, 432, -580, -132,
636, -76, 392, 4, -412, 540, 508, 328, -356, -36, 16,
-220, -64, -248, -60, 24, -192, 368, 1040, 92, -24, -1044,
-32, 40, 104, 148, 192, -136, -520, 56, -816, -224, 732,
392, 356, 212, -80, -424, -1008, -324, 588, -1496, 576, 460,
-816, -848, 56, -580, -92, -1372, -112, -496, 200, 364, 52,
-140, 48, -48, -60, 84, 72, 40, 132, -356, -268, -104,
-284, -404, 732, -520, 164, -304, -540, 120, 328, -76, -460,
756, 388, 588, 236, -436, -72, -176, -404, -316, -148, 716,
-604, 404, -72, -88, -888, -68, 944, 88, -220, -344, 960,
472, 460, -232, 704, 120, 832, -228, 692, -508, 132, -476,
844, -748, -364, -44, 1116, -1104, -1056, 76, 428, 552, -692,
60, 356, 96, -384, -188, -612, -576, 736, 508, 892, 352,
-1132, 504, -24, -352, 324, 332, -600, -312, 292, 508, -144,
-8, 484, 48, 284, -260, -240, 256, -100, -292, -204, -44,
472, -204, 908, -188, -1000, -256, 92, 1164, -392, 564, 356,
652, -28, -884, 256, 484, -192, 760, -176, 376, -524, -452,
-436, 860, -736, 212, 124, 504, -476, 468, 76, -472, 552,
-692, -944, -620, 740, -240, 400, 132, 20, 192, -196, 264,
-668, -1012, -60, 296, -316, -828, 76, -156, 284, -768, -448,
-832, 148, 248, 652, 616, 1236, 288, -328, -400, -124, 588,
220, 520, -696, 1032, 768, -740, -92, -272, 296, 448, -464,
412, -200, 392, 440, -200, 264, -152, -260, 320, 1032, 216,
320, -8, -64, 156, -1016, 1084, 1172, 536, 484, -432, 132,
372, -52, -256, 84, 116, -352, 48, 116, 304, -384, 412,
924, -300, 528, 628, 180, 648, 44, -980, -220, 1320, 48,
332, 748, 524, -268, -720, 540, -276, 564, -344, -208, -196,
436, 896, 88, -392, 132, 80, -964, -288, 568, 56, -48,
-456, 888, 8, 552, -156, -292, 948, 288, 128, -716, -292,

```

```

1192, -152, 876, 352, -600, -260, -812, -468, -28, -120, -32,
-44, 1284, 496, 192, 464, 312, -76, -516, -380, -456, -1012,
-48, 308, -156, 36, 492, -156, -808, 188, 1652, 68, -120,
-116, 316, 160, -140, 352, 808, -416, 592, 316, -480, 56,
528, -204, -568, 372, -232, 752, -344, 744, -4, 324, -416,
-600, 768, 268, -248, -88, -132, -420, -432, 80, -288, 404,
-316, -1216, -588, 520, -108, 92, -320, 368, -480, -216, -92,
1688, -300, 180, 1020, -176, 820, -68, -228, -260, 436, -904,
20, 40, -508, 440, -736, 312, 332, 204, 760, -372, 728,
96, -20, -632, -520, -560, 336, 1076, -64, -532, 776, 584,
192, 396, -728, -520, 276, -188, 80, -52, -612, -252, -48,
648, 212, -688, 228, -52, -260, 428, -412, -272, -404, 180,
816, -796, 48, 152, 484, -88, -216, 988, 696, 188, -528,
648, -116, -180, 316, 476, 12, -564, 96, 476, -252, -364,
-376, -392, 556, -256, -576, 260, -352, 120, -16, -136, -260,
-492, 72, 556, 660, 580, 616, 772, 436, 424, -32, -324,
-1268, 416, -324, -80, 920, 160, 228, 724, 32, -516, 64,
384, 68, -128, 136, 240, 248, -204, -68, 252, -932, -120,
-480, -628, -84, 192, 852, -404, -288, -132, 204, 100, 168,
-68, -196, -868, 460, 1080, 380, -80, 244, 0, 484, -888,
64, 184, 352, 600, 460, 164, 604, -196, 320, -64, 588,
-184, 228, 12, 372, 48, -848, -344, 224, 208, -200, 484,
128, -20, 272, -468, -840, 384, 256, -720, -520, -464, -580,
112, -120, 644, -356, -208, -608, -528, 704, 560, -424, 392,
828, 40, 84, 200, -152, 0, -144, 584, 280, -120, 80,
-556, -972, -196, -472, 724, 80, 168, -32, 88, 160, -688,
0, 160, 356, 372, -776, 740, -128, 676, -248, -480, 4,
-364, 96, 544, 232, -1032, 956, 236, 356, 20, -40, 300,
24, -676, -596, 132, 1120, -104, 532, -1096, 568, 648, 444,
508, 380, 188, -376, -604, 1488, 424, 24, 756, -220, -192,
716, 120, 920, 688, 168, 44, -460, 568, 284, 1144, 1160,
600, 424, 888, 656, -356, -320, 220, 316, -176, -724, -188,
-816, -628, -348, -228, -380, 1012, -452, -660, 736, 928, 404,
-696, -72, -268, -892, 128, 184, -344, -780, 360, 336, 400,
344, 428, 548, -112, 136, -228, -216, -820, -516, 340, 92,
-136, 116, -300, 376, -244, 100, -316, -520, -284, -12, 824,
164, -548, -180, -128, 116, -924, -828, 268, -368, -580, 620,
192, 160, 0, -1676, 1068, 424, -56, -360, 468, -156, 720,
288, -528, 556, -364, 548, -148, 504, 316, 152, -648, -620,
-684, -24, -376, -384, -108, -920, -1032, 768, 180, -264, -508,
-1268, -260, -60, 300, -240, 988, 724, -376, -576, -212, -736,
556, 192, 1092, -620, -880, 376, -56, -4, -216, -32, 836,
268, 396, 1332, 864, -600, 100, 56, -412, -92, 356, 180,
884, -468, -436, 292, -388, -804, -704, -840, 368, -348, 140,
-724, 1536, 940, 372, 112, -372, 436, -480, 1136, 296, -32,
-228, 132, -48, -220, 868, -1016, -60, -1044, -464, 328, 916,
244, 12, -736, -296, 360, 468, -376, -108, -92, 788, 368,
-56, 544, 400, -672, -420, 728, 16, 320, 44, -284, -380,
-796, 488, 132, 204, -596, -372, 88, -152, -908, -636, -572,
-624, -116, -692, -200, -56, 276, -88, 484, -324, 948, 864,

```

```

1000, -456, -184, -276, 292, -296, 156, 676, 320, 160, 908,
-84, -1236, -288, -116, 260, -372, -644, 732, -756, -96, 84,
344, -520, 348, -688, 240, -84, 216, -1044, -136, -676, -396,
-1500, 960, -40, 176, 168, 1516, 420, -504, -344, -364, -360,
1216, -940, -380, -212, 252, -660, -708, 484, -444, -152, 928,
-120, 1112, 476, -260, 560, -148, -344, 108, -196, 228, -288,
504, 560, -328, -88, 288, -1008, 460, -228, 468, -836, -196,
76, 388, 232, 412, -1168, -716, -644, 756, -172, -356, -504,
116, 432, 528, 48, 476, -168, -608, 448, 160, -532, -272,
28, -676, -12, 828, 980, 456, 520, 104, -104, 256, -344,
-4, -28, -368, -52, -524, -572, -556, -200, 768, 1124, -208,
-512, 176, 232, 248, -148, -888, 604, -600, -304, 804, -156,
-212, 488, -192, -804, -256, 368, -360, -916, -328, 228, -240,
-448, -472, 856, -556, -364, 572, -12, -156, -368, -340, 432,
252, -752, -152, 288, 268, -580, -848, -592, 108, -76, 244,
312, -716, 592, -80, 436, 360, 4, -248, 160, 516, 584,
732, 44, -468, -280, -292, -156, -588, 28, 308, 912, 24,
124, 156, 180, -252, 944, -924, -772, -520, -428, -624, 300,
-212, -1144, 32, -724, 800, -1128, -212, -1288, -848, 180, -416,
440, 192, -576, -792, -76, -1080, 80, -532, -352, -132, 380,
-820, 148, 1112, 128, 164, 456, 700, -924, 144, -668, -384,
648, -832, 508, 552, -52, -100, -656, 208, -568, 748, -88,
680, 232, 300, 192, -408, -1012, -152, -252, -268, 272, -876,
-664, -648, -332, -136, 16, 12, 1152, -28, 332, -536, 320,
-672, -460, -316, 532, -260, 228, -40, 1052, -816, 180, 88,
-496, -556, -672, -368, 428, 92, 356, 404, -408, 252, 196,
-176, -556, 792, 268, 32, 372, 40, 96, -332, 328, 120,
372, -900, -40, 472, -264, -592, 952, 128, 656, 112, 664,
-232, 420, 4, -344, -464, 556, 244, -416, -32, 252, 0,
-412, 188, -696, 508, -476, 324, -1096, 656, -312, 560, 264,
-136, 304, 160, -64, -580, 248, 336, -720, 560, -348, -288,
-276, -196, -500, 852, -544, -236, -1128, -992, -776, 116, 56,
52, 860, 884, 212, -12, 168, 1020, 512, -552, 924, -148,
716, 188, 164, -340, -520, -184, 880, -152, -680, -208, -1156,
-300, -528, -472, 364, 100, -744, -1056, -32, 540, 280, 144,
-676, -32, -232, -280, -224, 96, 568, -76, 172, 148, 148,
104, 32, -296, -32, 788, -80, 32, -16, 280, 288, 944,
428, -484
}

```

## 9.4. Default CDF tables

This section contains the default values for the cumulative distributions.



```

Default_Intra_Frame_Y_Mode_Cdf[ INTRA_MODE_CONTEXTS ][ INTRA_MODE_CONTEXTS ][ INTRA_MODES + 1 ] = {
{
{ 15588, 17027, 19338, 20218, 20682, 21110, 21825, 23244,
  24189, 28165, 29093, 30466, 32768, 0 },
{ 12016, 18066, 19516, 20303, 20719, 21444, 21888, 23032,
  24434, 28658, 30172, 31409, 32768, 0 },
{ 10052, 10771, 22296, 22788, 23055, 23239, 24133, 25620,
  26160, 29336, 29929, 31567, 32768, 0 },
{ 14091, 15406, 16442, 18808, 19136, 19546, 19998, 22096,
  24746, 29585, 30958, 32462, 32768, 0 },
{ 12122, 13265, 15603, 16501, 18609, 20033, 22391, 25583,
  26437, 30261, 31073, 32475, 32768, 0 }
},
{
{ 10023, 19585, 20848, 21440, 21832, 22760, 23089, 24023,
  25381, 29014, 30482, 31436, 32768, 0 },
{ 5983, 24099, 24560, 24886, 25066, 25795, 25913, 26423,
  27610, 29905, 31276, 31794, 32768, 0 },
{ 7444, 12781, 20177, 20728, 21077, 21607, 22170, 23405,
  24469, 27915, 29090, 30492, 32768, 0 },
{ 8537, 14689, 15432, 17087, 17408, 18172, 18408, 19825,
  24649, 29153, 31096, 32210, 32768, 0 },
{ 7543, 14231, 15496, 16195, 17905, 20717, 21984, 24516,
  26001, 29675, 30981, 31994, 32768, 0 }
},
{
{ 12613, 13591, 21383, 22004, 22312, 22577, 23401, 25055,
  25729, 29538, 30305, 32077, 32768, 0 },
{ 9687, 13470, 18506, 19230, 19604, 20147, 20695, 22062,
  23219, 27743, 29211, 30907, 32768, 0 },
{ 6183, 6505, 26024, 26252, 26366, 26434, 27082, 28354, 28555,
  30467, 30794, 32086, 32768, 0 },
{ 10718, 11734, 14954, 17224, 17565, 17924, 18561, 21523,
  23878, 28975, 30287, 32252, 32768, 0 },
{ 9194, 9858, 16501, 17263, 18424, 19171, 21563, 25961, 26561,
  30072, 30737, 32463, 32768, 0 }
},
{
{ 12602, 14399, 15488, 18381, 18778, 19315, 19724, 21419,
  25060, 29696, 30917, 32409, 32768, 0 },
{ 8203, 13821, 14524, 17105, 17439, 18131, 18404, 19468,
  25225, 29485, 31158, 32342, 32768, 0 },
{ 8451, 9731, 15004, 17643, 18012, 18425, 19070, 21538, 24605,
  29118, 30078, 32018, 32768, 0 },
{ 7714, 9048, 9516, 16667, 16817, 16994, 17153, 18767, 26743,
  30389, 31536, 32528, 32768, 0 },
{ 8843, 10280, 11496, 15317, 16652, 17943, 19108, 22718,
  25769, 29953, 30983, 32485, 32768, 0 }
},
}

```

```

{ 12578, 13671, 15979, 16834, 19075, 20913, 22989, 25449,
  26219, 30214, 31150, 32477, 32768, 0 },
{ 9563, 13626, 15080, 15892, 17756, 20863, 22207, 24236,
  25380, 29653, 31143, 32277, 32768, 0 },
{ 8356, 8901, 17616, 18256, 19350, 20106, 22598, 25947, 26466,
  29900, 30523, 32261, 32768, 0 },
{ 10835, 11815, 13124, 16042, 17018, 18039, 18947, 22753,
  24615, 29489, 30883, 32482, 32768, 0 },
{ 7618, 8288, 9859, 10509, 15386, 18657, 22903, 28776, 29180,
  31355, 31802, 32593, 32768, 0 }
}
}

```

```

Default_Y_Mode_Cdf[ BLOCK_SIZE_GROUPS ][ INTRA_MODES + 1 ] = {
{ 22801, 23489, 24293, 24756,
  25601, 26123, 26606, 27418,
  27945, 29228, 29685, 30349,
  32768, 0 },
{ 18673, 19845, 22631, 23318,
  23950, 24649, 25527, 27364,
  28152, 29701, 29984, 30852,
  32768, 0 },
{ 19770, 20979, 23396, 23939,
  24241, 24654, 25136, 27073,
  27830, 29360, 29730, 30659,
  32768, 0 },
{ 20155, 21301, 22838, 23178,
  23261, 23533, 23703, 24804,
  25352, 26575, 27016, 28049,
  32768, 0 }
}

```

```

Default_Uv_Mode_Cfl_Not_Allowed_Cdf[ INTRA_MODES ][ UV_INTRA_MODES_CFL_NOT_ALLOWED + 1 ] = {
  { 22631, 24152, 25378, 25661, 25986, 26520, 27055, 27923,
    28244, 30059, 30941, 31961, 32768, 0 },
  { 9513, 26881, 26973, 27046, 27118, 27664, 27739, 27824,
    28359, 29505, 29800, 31796, 32768, 0 },
  { 9845, 9915, 28663, 28704, 28757, 28780, 29198, 29822, 29854,
    30764, 31777, 32029, 32768, 0 },
  { 13639, 13897, 14171, 25331, 25606, 25727, 25953, 27148,
    28577, 30612, 31355, 32493, 32768, 0 },
  { 9764, 9835, 9930, 9954, 25386, 27053, 27958, 28148, 28243,
    31101, 31744, 32363, 32768, 0 },
  { 11825, 13589, 13677, 13720, 15048, 29213, 29301, 29458,
    29711, 31161, 31441, 32550, 32768, 0 },
  { 14175, 14399, 16608, 16821, 17718, 17775, 28551, 30200,
    30245, 31837, 32342, 32667, 32768, 0 },
  { 12885, 13038, 14978, 15590, 15673, 15748, 16176, 29128,
    29267, 30643, 31961, 32461, 32768, 0 },
  { 12026, 13661, 13874, 15305, 15490, 15726, 15995, 16273,
    28443, 30388, 30767, 32416, 32768, 0 },
  { 19052, 19840, 20579, 20916, 21150, 21467, 21885, 22719,
    23174, 28861, 30379, 32175, 32768, 0 },
  { 18627, 19649, 20974, 21219, 21492, 21816, 22199, 23119,
    23527, 27053, 31397, 32148, 32768, 0 },
  { 17026, 19004, 19997, 20339, 20586, 21103, 21349, 21907,
    22482, 25896, 26541, 31819, 32768, 0 },
  { 12124, 13759, 14959, 14992, 15007, 15051, 15078, 15166,
    15255, 15753, 16039, 16606, 32768, 0 }
}

```

```

Default_Uv_Mode_Cfl_Allowed_Cdf[ INTRA_MODES ][ UV_INTRA_MODES_CFL_ALLOWED + 1 ] = {
  { 10407, 11208, 12900, 13181, 13823, 14175, 14899, 15656,
    15986, 20086, 20995, 22455, 24212, 32768, 0 },
  { 4532, 19780, 20057, 20215, 20428, 21071, 21199, 21451,
    22099, 24228, 24693, 27032, 29472, 32768, 0 },
  { 5273, 5379, 20177, 20270, 20385, 20439, 20949, 21695, 21774,
    23138, 24256, 24703, 26679, 32768, 0 },
  { 6740, 7167, 7662, 14152, 14536, 14785, 15034, 16741, 18371,
    21520, 22206, 23389, 24182, 32768, 0 },
  { 4987, 5368, 5928, 6068, 19114, 20315, 21857, 22253, 22411,
    24911, 25380, 26027, 26376, 32768, 0 },
  { 5370, 6889, 7247, 7393, 9498, 21114, 21402, 21753, 21981,
    24780, 25386, 26517, 27176, 32768, 0 },
  { 4816, 4961, 7204, 7326, 8765, 8930, 20169, 20682, 20803,
    23188, 23763, 24455, 24940, 32768, 0 },
  { 6608, 6740, 8529, 9049, 9257, 9356, 9735, 18827, 19059,
    22336, 23204, 23964, 24793, 32768, 0 },
  { 5998, 7419, 7781, 8933, 9255, 9549, 9753, 10417, 18898,
    22494, 23139, 24764, 25989, 32768, 0 },
  { 10660, 11298, 12550, 12957, 13322, 13624, 14040, 15004,
    15534, 20714, 21789, 23443, 24861, 32768, 0 },
  { 10522, 11530, 12552, 12963, 13378, 13779, 14245, 15235,
    15902, 20102, 22696, 23774, 25838, 32768, 0 },
  { 10099, 10691, 12639, 13049, 13386, 13665, 14125, 15163,
    15636, 19676, 20474, 23519, 25208, 32768, 0 },
  { 3144, 5087, 7382, 7504, 7593, 7690, 7801, 8064, 8232, 9248,
    9875, 10521, 29048, 32768, 0 }
}

```

```

Default_Angle_Delta_Cdf[ DIRECTIONAL_MODES ][ (2 * MAX_ANGLE_DELTA + 1) + 1 ] = {
  { 2180, 5032, 7567, 22776, 26989, 30217, 32768, 0 },
  { 2301, 5608, 8801, 23487, 26974, 30330, 32768, 0 },
  { 3780, 11018, 13699, 19354, 23083, 31286, 32768, 0 },
  { 4581, 11226, 15147, 17138, 21834, 28397, 32768, 0 },
  { 1737, 10927, 14509, 19588, 22745, 28823, 32768, 0 },
  { 2664, 10176, 12485, 17650, 21600, 30495, 32768, 0 },
  { 2240, 11096, 15453, 20341, 22561, 28917, 32768, 0 },
  { 3605, 10428, 12459, 17676, 21244, 30655, 32768, 0 }
}

```

```

Default_Intrabc_Cdf[ 2 + 1 ] = { 30531, 32768, 0 }

```

```

Default_Partition_W8_Cdf[ PARTITION_CONTEXTS ][ 5 ] = {
    { 19132, 25510, 30392, 32768, 0 },
    { 13928, 19855, 28540, 32768, 0 },
    { 12522, 23679, 28629, 32768, 0 },
    { 9896, 18783, 25853, 32768, 0 },
}

```

```

Default_Partition_W16_Cdf[ PARTITION_CONTEXTS ][ 11 ] = {
    { 15597, 20929, 24571, 26706, 27664, 28821, 29601, 30571, 31902, 32768, 0 },
    { 7925, 11043, 16785, 22470, 23971, 25043, 26651, 28701, 29834, 32768, 0 },
    { 5414, 13269, 15111, 20488, 22360, 24500, 25537, 26336, 32117, 32768, 0 },
    { 2662, 6362, 8614, 20860, 23053, 24778, 26436, 27829, 31171, 32768, 0 }
}

```

```

Default_Partition_W32_Cdf[ PARTITION_CONTEXTS ][ 11 ] = {
    { 18462, 20920, 23124, 27647, 28227, 29049, 29519, 30178, 31544, 32768, 0 },
    { 7689, 9060, 12056, 24992, 25660, 26182, 26951, 28041, 29052, 32768, 0 },
    { 6015, 9009, 10062, 24544, 25409, 26545, 27071, 27526, 32047, 32768, 0 },
    { 1394, 2208, 2796, 28614, 29061, 29466, 29840, 30185, 31899, 32768, 0 }
}

```

```

Default_Partition_W64_Cdf[ PARTITION_CONTEXTS ][ 11 ] = {
    { 20137, 21547, 23078, 29566, 29837, 30261, 30524, 30892, 31724, 32768, 0 },
    { 6732, 7490, 9497, 27944, 28250, 28515, 28969, 29630, 30104, 32768, 0 },
    { 5945, 7663, 8348, 28683, 29117, 29749, 30064, 30298, 32238, 32768, 0 },
    { 870, 1212, 1487, 31198, 31394, 31574, 31743, 31881, 32332, 32768, 0 }
}

```

```

Default_Partition_W128_Cdf[ PARTITION_CONTEXTS ][ 9 ] = {
    { 27899, 28219, 28529, 32484, 32539, 32619, 32639, 32768, 0 },
    { 6607, 6990, 8268, 32060, 32219, 32338, 32371, 32768, 0 },
    { 5429, 6676, 7122, 32027, 32227, 32531, 32582, 32768, 0 },
    { 711, 966, 1172, 32448, 32538, 32617, 32664, 32768, 0 }
}

```

```

Default_Tx_8x8_Cdf[ TX_SIZE_CONTEXTS ][ MAX_TX_DEPTH + 1 ] = {
    { 19968, 32768, 0 },
    { 19968, 32768, 0 },
    { 24320, 32768, 0 }
}

```

```

Default_Tx_16x16_Cdf[ TX_SIZE_CONTEXTS ][ MAX_TX_DEPTH + 2 ] = {
    { 12272, 30172, 32768, 0 },
    { 12272, 30172, 32768, 0 },
    { 18677, 30848, 32768, 0 }
}

```

```

Default_Tx_32x32_Cdf[ TX_SIZE_CONTEXTS ][ MAX_TX_DEPTH + 2 ] = {
    { 12986, 15180, 32768, 0 },
    { 12986, 15180, 32768, 0 },
    { 24302, 25602, 32768, 0 }
}

```

```

Default_Tx_64x64_Cdf[ TX_SIZE_CONTEXTS ][ MAX_TX_DEPTH + 2 ] = {
    { 5782, 11475, 32768, 0 },
    { 5782, 11475, 32768, 0 },
    { 16803, 22759, 32768, 0 }
}

```

```

Default_Txfm_Split_Cdf[ TXFM_PARTITION_CONTEXTS ][ 3 ] = {
    { 28581, 32768, 0 }, { 23846, 32768, 0 }, { 20847, 32768, 0 },
    { 24315, 32768, 0 }, { 18196, 32768, 0 }, { 12133, 32768, 0 },
    { 18791, 32768, 0 }, { 10887, 32768, 0 }, { 11005, 32768, 0 },
    { 27179, 32768, 0 }, { 20004, 32768, 0 }, { 11281, 32768, 0 },
    { 26549, 32768, 0 }, { 19308, 32768, 0 }, { 14224, 32768, 0 },
    { 28015, 32768, 0 }, { 21546, 32768, 0 }, { 14400, 32768, 0 },
    { 28165, 32768, 0 }, { 22401, 32768, 0 }, { 16088, 32768, 0 }
}

```

```

Default_Filter_Intra_Mode_Cdf[ 6 ] = { 8949, 12776, 17211, 29558, 32768, 0 }

```

```

Default_Filter_Intra_Cdf[ BLOCK_SIZES ][ 3 ] = {
    { 4621, 32768, 0 }, { 6743, 32768, 0 }, { 5893, 32768, 0 },
    { 7866, 32768, 0 }, { 12551, 32768, 0 }, { 9394, 32768, 0 },
    { 12408, 32768, 0 }, { 14301, 32768, 0 }, { 12756, 32768, 0 },
    { 22343, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 12770, 32768, 0 }, { 10368, 32768, 0 },
    { 20229, 32768, 0 }, { 18101, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }
}

```

**Note:** Indices 10 to 15 and 20 to 21 inclusive are never used in the first dimension of the Default\_Filter\_Intra\_Cdf CDF table.

```
Default_Segment_Id_Cdf[ SEGMENT_ID_CONTEXTS ][ MAX_SEGMENTS + 1 ] = {
  { 5622, 7893, 16093, 18233, 27809, 28373, 32533, 32768, 0 },
  { 14274, 18230, 22557, 24935, 29980, 30851, 32344, 32768, 0 },
  { 27527, 28487, 28723, 28890, 32397, 32647, 32679, 32768, 0 }
}
```

```
Default_Segment_Id_Predicted_Cdf[ SEGMENT_ID_PREDICTED_CONTEXTS ][ 3 ] = {
  { 128 * 128, 32768, 0 },
  { 128 * 128, 32768, 0 },
  { 128 * 128, 32768, 0 }
}
```

```
Default_Mv_Class0_Hp_Cdf[ 3 ] = {
  160*128, 32768, 0
}
```

```
Default_Mv_Hp_Cdf[ 3 ] = {
  128*128, 32768, 0
}
```

```
Default_Mv_Sign_Cdf[3] = {
  128*128, 32768, 0
}
```

```
Default_Mv_Bit_Cdf[ MV_OFFSET_BITS ][ 3 ] = {
  { 136*128, 32768, 0 },
  { 140*128, 32768, 0 },
  { 148*128, 32768, 0 },
  { 160*128, 32768, 0 },
  { 176*128, 32768, 0 },
  { 192*128, 32768, 0 },
  { 224*128, 32768, 0 },
  { 234*128, 32768, 0 },
  { 234*128, 32768, 0 },
  { 240*128, 32768, 0 }
}
```

```
Default_Mv_Class0_Bit_Cdf[ 3 ] = {
    216*128, 32768, 0
}
```

```
Default_New_Mv_Cdf[ NEW_MV_CONTEXTS ][ 3 ] = {
    { 24035, 32768, 0 },
    { 16630, 32768, 0 },
    { 15339, 32768, 0 },
    { 8386, 32768, 0 },
    { 12222, 32768, 0 },
    { 4676, 32768, 0 }
}
```

```
Default_Zero_Mv_Cdf[ ZERO_MV_CONTEXTS ][ 3 ] = {
    { 2175, 32768, 0 },
    { 1054, 32768, 0 }
}
```

```
Default_Ref_Mv_Cdf[ REF_MV_CONTEXTS ][ 3 ] = {
    { 23974, 32768, 0 },
    { 24188, 32768, 0 },
    { 17848, 32768, 0 },
    { 28622, 32768, 0 },
    { 24312, 32768, 0 },
    { 19923, 32768, 0 }
}
```

```
Default_Dr1_Mode_Cdf[ DRL_MODE_CONTEXTS ][ 3 ] = {
    { 13104, 32768, 0 },
    { 24560, 32768, 0 },
    { 18945, 32768, 0 }
}
```

```
Default_Is_Inter_Cdf[ IS_INTER_CONTEXTS ][ 3 ] = {
    { 806, 32768, 0 },
    { 16662, 32768, 0 },
    { 20186, 32768, 0 },
    { 26538, 32768, 0 }
}
```



```
Default_Comp_Mode_Cdf[ COMP_INTER_CONTEXTS ][ 3 ] = {
    { 26828, 32768, 0 },
    { 24035, 32768, 0 },
    { 12031, 32768, 0 },
    { 10640, 32768, 0 },
    { 2901, 32768, 0 }
}
```

```
Default_Skip_Mode_Cdf[ SKIP_MODE_CONTEXTS ][ 3 ] = {
    {32621, 32768, 0},
    {20708, 32768, 0},
    {8127, 32768, 0}
}
```

```
Default_Skip_Cdf[ SKIP_CONTEXTS ][ 3 ] = {
    { 31671, 32768, 0 },
    { 16515, 32768, 0 },
    { 4576, 32768, 0 }
}
```

```
Default_Comp_Ref_Cdf[ REF_CONTEXTS ][ FWD_REFS - 1 ][ 3 ] = {
    { { 4946, 32768, 0 },
      { 9468, 32768, 0 },
      { 1503, 32768, 0 } },
    { { 19891, 32768, 0 },
      { 22441, 32768, 0 },
      { 15160, 32768, 0 } },
    { { 30731, 32768, 0 },
      { 31059, 32768, 0 },
      { 27544, 32768, 0 } }
}
```

```
Default_Comp_Bwd_Ref_Cdf[ REF_CONTEXTS ][ BWD_REFS - 1 ][ 3 ] = {
    { { 2235, 32768, 0 }, { 1423, 32768, 0 } },
    { { 17182, 32768, 0 }, { 15175, 32768, 0 } },
    { { 30606, 32768, 0 }, { 30489, 32768, 0 } }
}
```

```

Default_Single_Ref_Cdf[ REF_CONTEXTS ][ SINGLE_REFS - 1 ][ 3 ] = {
  { { 4897, 32768, 0 }, { 1555, 32768, 0 }, { 4236, 32768, 0 },
    { 8650, 32768, 0 }, { 904, 32768, 0 }, { 1444, 32768, 0 } },
  { { 16973, 32768, 0 }, { 16751, 32768, 0 }, { 19647, 32768, 0 },
    { 24773, 32768, 0 }, { 11014, 32768, 0 }, { 15087, 32768, 0 } },
  { { 29744, 32768, 0 }, { 30279, 32768, 0 }, { 31194, 32768, 0 },
    { 31895, 32768, 0 }, { 26875, 32768, 0 }, { 30304, 32768, 0 } }
}

```

```

Default_Compound_Mode_Cdf[ COMPOUND_MODE_CONTEXTS ][ COMPOUND_MODES + 1 ] = {
  { 7760, 13823, 15808, 17641, 19156, 20666, 26891, 32768, 0 },
  { 10730, 19452, 21145, 22749, 24039, 25131, 28724, 32768, 0 },
  { 10664, 20221, 21588, 22906, 24295, 25387, 28436, 32768, 0 },
  { 13298, 16984, 20471, 24182, 25067, 25736, 26422, 32768, 0 },
  { 18904, 23325, 25242, 27432, 27898, 28258, 30758, 32768, 0 },
  { 10725, 17454, 20124, 22820, 24195, 25168, 26046, 32768, 0 },
  { 17125, 24273, 25814, 27492, 28214, 28704, 30592, 32768, 0 },
  { 13046, 23214, 24505, 25942, 27435, 28442, 29330, 32768, 0 }
}

```

```

Default_Interp_Filter_Cdf[ INTERP_FILTER_CONTEXTS ][ INTERP_FILTERS + 1 ] = {
  { 31935, 32720, 32768, 0 },
  { 5568, 32719, 32768, 0 },
  { 422, 2938, 32768, 0 },
  { 28244, 32608, 32768, 0 },
  { 31206, 31953, 32768, 0 },
  { 4862, 32121, 32768, 0 },
  { 770, 1152, 32768, 0 },
  { 20889, 25637, 32768, 0 },
  { 31910, 32724, 32768, 0 },
  { 4120, 32712, 32768, 0 },
  { 305, 2247, 32768, 0 },
  { 27403, 32636, 32768, 0 },
  { 31022, 32009, 32768, 0 },
  { 2963, 32093, 32768, 0 },
  { 601, 943, 32768, 0 },
  { 14969, 21398, 32768, 0 }
}

```

```

Default_Motion_Mode_Cdf[ BLOCK_SIZES ][ MOTION_MODES + 1 ] = {
  { 10923, 21845, 32768, 0 },
  { 10923, 21845, 32768, 0 },
  { 10923, 21845, 32768, 0 },
  { 7651, 24760, 32768, 0 },
  { 4738, 24765, 32768, 0 },
  { 5391, 25528, 32768, 0 },
  { 19419, 26810, 32768, 0 },
  { 5123, 23606, 32768, 0 },
  { 11606, 24308, 32768, 0 },
  { 26260, 29116, 32768, 0 },
  { 20360, 28062, 32768, 0 },
  { 21679, 26830, 32768, 0 },
  { 29516, 30701, 32768, 0 },
  { 28898, 30397, 32768, 0 },
  { 30878, 31335, 32768, 0 },
  { 32507, 32558, 32768, 0 },
  { 10923, 21845, 32768, 0 },
  { 10923, 21845, 32768, 0 },
  { 28799, 31390, 32768, 0 },
  { 26431, 30774, 32768, 0 },
  { 28973, 31594, 32768, 0 },
  { 29742, 31203, 32768, 0 }
}

```

**Note:** Indices 0 to 2 and 16 to 17 inclusive are never used in the first dimension of the Default\_Motion\_Mode\_Cdf CDF table.

```

Default_Mv_Joint_Cdf[ MV_JOINTS + 1 ] = {
  4096, 11264, 19328, 32768, 0
}

```

```

Default_Mv_Class_Cdf[ 2 ][ MV_CLASSES + 1 ] = {
  { 28672, 30976, 31858, 32320,
    32551, 32656, 32740, 32757,
    32762, 32767, 32768, 0 },
  { 28672, 30976, 31858, 32320,
    32551, 32656, 32740, 32757,
    32762, 32767, 32768, 0 }
}

```

```

Default_Mv_Class0_Fr_Cdf[ 2 ][ CLASS0_SIZE ][ MV_JOINTS + 1 ] = {
    { { 16384, 24576, 26624, 32768, 0 },
      { 12288, 21248, 24128, 32768, 0 } },
    { { 16384, 24576, 26624, 32768, 0 },
      { 12288, 21248, 24128, 32768, 0 } },
}

```

```

Default_Mv_Fr_Cdf[ 2 ][ MV_JOINTS + 1 ] = {
    { 8192, 17408, 21248, 32768, 0 },
    { 8192, 17408, 21248, 32768, 0 },
}

```

```

Default_Palette_Y_Size_Cdf[ PALETTE_BLOCK_SIZE_CONTEXTS ][ PALETTE_SIZES + 1 ] = {
    { 7952, 13000, 18149, 21478, 25527, 29241, 32768, 0 },
    { 7139, 11421, 16195, 19544, 23666, 28073, 32768, 0 },
    { 7788, 12741, 17325, 20500, 24315, 28530, 32768, 0 },
    { 8271, 14064, 18246, 21564, 25071, 28533, 32768, 0 },
    { 12725, 19180, 21863, 24839, 27535, 30120, 32768, 0 },
    { 9711, 14888, 16923, 21052, 25661, 27875, 32768, 0 },
    { 14940, 20797, 21678, 24186, 27033, 28999, 32768, 0 }
}

```

```

Default_Palette_Uv_Size_Cdf[ PALETTE_BLOCK_SIZE_CONTEXTS ][ PALETTE_SIZES + 1 ] = {
    { 8713, 19979, 27128, 29609, 31331, 32272, 32768, 0 },
    { 5839, 15573, 23581, 26947, 29848, 31700, 32768, 0 },
    { 4426, 11260, 17999, 21483, 25863, 29430, 32768, 0 },
    { 3228, 9464, 14993, 18089, 22523, 27420, 32768, 0 },
    { 3768, 8886, 13091, 17852, 22495, 27207, 32768, 0 },
    { 2464, 8451, 12861, 21632, 25525, 28555, 32768, 0 },
    { 1269, 5435, 10433, 18963, 21700, 25865, 32768, 0 }
}

```

```

Default_Palette_Size_2_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 3 ] = {
    { 28710, 32768, 0 },
    { 16384, 32768, 0 },
    { 10553, 32768, 0 },
    { 27036, 32768, 0 },
    { 31603, 32768, 0 }
}

Default_Palette_Size_3_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 4 ] = {
    { 27877, 30490, 32768, 0 },
    { 11532, 25697, 32768, 0 },
    { 6544, 30234, 32768, 0 },
    { 23018, 28072, 32768, 0 },
    { 31915, 32385, 32768, 0 }
}

Default_Palette_Size_4_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 5 ] = {
    { 25572, 28046, 30045, 32768, 0 },
    { 9478, 21590, 27256, 32768, 0 },
    { 7248, 26837, 29824, 32768, 0 },
    { 19167, 24486, 28349, 32768, 0 },
    { 31400, 31825, 32250, 32768, 0 }
}

Default_Palette_Size_5_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 6 ] = {
    { 24779, 26955, 28576, 30282, 32768, 0 },
    { 8669, 20364, 24073, 28093, 32768, 0 },
    { 4255, 27565, 29377, 31067, 32768, 0 },
    { 19864, 23674, 26716, 29530, 32768, 0 },
    { 31646, 31893, 32147, 32426, 32768, 0 }
}

Default_Palette_Size_6_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 7 ] = {
    { 23132, 25407, 26970, 28435, 30073, 32768, 0 },
    { 7443, 17242, 20717, 24762, 27982, 32768, 0 },
    { 6300, 24862, 26944, 28784, 30671, 32768, 0 },
    { 18916, 22895, 25267, 27435, 29652, 32768, 0 },
    { 31270, 31550, 31808, 32059, 32353, 32768, 0 }
}

Default_Palette_Size_7_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 8 ] = {
    { 23105, 25199, 26464, 27684, 28931, 30318, 32768, 0 },
    { 6950, 15447, 18952, 22681, 25567, 28563, 32768, 0 },
    { 7560, 23474, 25490, 27203, 28921, 30708, 32768, 0 },
    { 18544, 22373, 24457, 26195, 28119, 30045, 32768, 0 },
    { 31198, 31451, 31670, 31882, 32123, 32391, 32768, 0 }
}

Default_Palette_Size_8_Y_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 9 ] = {
    { 21689, 23883, 25163, 26352, 27506, 28827, 30195, 32768, 0 },

```

```
{ 6892, 15385, 17840, 21606, 24287, 26753, 29204, 32768, 0 },  
{ 5651, 23182, 25042, 26518, 27982, 29392, 30900, 32768, 0 },  
{ 19349, 22578, 24418, 25994, 27524, 29031, 30448, 32768, 0 },  
{ 31028, 31270, 31504, 31705, 31927, 32153, 32392, 32768, 0 }  
}
```

```

Default_Palette_Size_2_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 3 ] = {
    { 29089, 32768, 0 },
    { 16384, 32768, 0 },
    { 8713, 32768, 0 },
    { 29257, 32768, 0 },
    { 31610, 32768, 0 }
}

Default_Palette_Size_3_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 4 ] = {
    { 25257, 29145, 32768, 0 },
    { 12287, 27293, 32768, 0 },
    { 7033, 27960, 32768, 0 },
    { 20145, 25405, 32768, 0 },
    { 30608, 31639, 32768, 0 }
}

Default_Palette_Size_4_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 5 ] = {
    { 24210, 27175, 29903, 32768, 0 },
    { 9888, 22386, 27214, 32768, 0 },
    { 5901, 26053, 29293, 32768, 0 },
    { 18318, 22152, 28333, 32768, 0 },
    { 30459, 31136, 31926, 32768, 0 }
}

Default_Palette_Size_5_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 6 ] = {
    { 22980, 25479, 27781, 29986, 32768, 0 },
    { 8413, 21408, 24859, 28874, 32768, 0 },
    { 2257, 29449, 30594, 31598, 32768, 0 },
    { 19189, 21202, 25915, 28620, 32768, 0 },
    { 31844, 32044, 32281, 32518, 32768, 0 }
}

Default_Palette_Size_6_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 7 ] = {
    { 22217, 24567, 26637, 28683, 30548, 32768, 0 },
    { 7307, 16406, 19636, 24632, 28424, 32768, 0 },
    { 4441, 25064, 26879, 28942, 30919, 32768, 0 },
    { 17210, 20528, 23319, 26750, 29582, 32768, 0 },
    { 30674, 30953, 31396, 31735, 32207, 32768, 0 }
}

Default_Palette_Size_7_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 8 ] = {
    { 21239, 23168, 25044, 26962, 28705, 30506, 32768, 0 },
    { 6545, 15012, 18004, 21817, 25503, 28701, 32768, 0 },
    { 3448, 26295, 27437, 28704, 30126, 31442, 32768, 0 },
    { 15889, 18323, 21704, 24698, 26976, 29690, 32768, 0 },
    { 30988, 31204, 31479, 31734, 31983, 32325, 32768, 0 }
}

Default_Palette_Size_8_Uv_Color_Cdf[ PALETTE_COLOR_CONTEXTS ][ 9 ] = {
    { 21442, 23288, 24758, 26246, 27649, 28980, 30563, 32768, 0 },

```

```
{ 5863, 14933, 17552, 20668, 23683, 26411, 29273, 32768, 0 },
{ 3415, 25810, 26877, 27990, 29223, 30394, 31618, 32768, 0 },
{ 17965, 20084, 22232, 23974, 26274, 28402, 30390, 32768, 0 },
{ 31190, 31329, 31516, 31679, 31825, 32026, 32322, 32768, 0 }
}
```

```
Default_Palette_Y_Mode_Cdf[ PALETTE_BLOCK_SIZE_CONTEXTS ][ PALETTE_Y_MODE_CONTEXTS ][ 3 ] = {
  { { 31676, 32768, 0 }, { 3419, 32768, 0 }, { 1261, 32768, 0 } },
  { { 31912, 32768, 0 }, { 2859, 32768, 0 }, { 980, 32768, 0 } },
  { { 31823, 32768, 0 }, { 3400, 32768, 0 }, { 781, 32768, 0 } },
  { { 32030, 32768, 0 }, { 3561, 32768, 0 }, { 904, 32768, 0 } },
  { { 32309, 32768, 0 }, { 7337, 32768, 0 }, { 1462, 32768, 0 } },
  { { 32265, 32768, 0 }, { 4015, 32768, 0 }, { 1521, 32768, 0 } },
  { { 32450, 32768, 0 }, { 7946, 32768, 0 }, { 129, 32768, 0 } }
}
```

```
Default_Palette_Uv_Mode_Cdf[ PALETTE_UV_MODE_CONTEXTS ][ 3 ] = {
  { 32461, 32768, 0 }, { 21488, 32768, 0 }
}
```

```
Default_Delta_Q_Cdf[ DELTA_Q_SMALL + 2 ] = {
  28160, 32120, 32677, 32768, 0
}
```

```
Default_Delta_Lf_Cdf[ DELTA_LF_SMALL + 2 ] = {
  28160, 32120, 32677, 32768, 0
}
```



```

Default_Intra_Tx_Type_Set1_Cdf[ 2 ][ INTRA_MODES ][ 8 ] = {
    {
        { 1535, 8035, 9461, 12751, 23467, 27825, 32768, 0 },
        { 564, 3335, 9709, 10870, 18143, 28094, 32768, 0 },
        { 672, 3247, 3676, 11982, 19415, 23127, 32768, 0 },
        { 5279, 13885, 15487, 18044, 23527, 30252, 32768, 0 },
        { 4423, 6074, 7985, 10416, 25693, 29298, 32768, 0 },
        { 1486, 4241, 9460, 10662, 16456, 27694, 32768, 0 },
        { 439, 2838, 3522, 6737, 18058, 23754, 32768, 0 },
        { 1190, 4233, 4855, 11670, 20281, 24377, 32768, 0 },
        { 1045, 4312, 8647, 10159, 18644, 29335, 32768, 0 },
        { 202, 3734, 4747, 7298, 17127, 24016, 32768, 0 },
        { 447, 4312, 6819, 8884, 16010, 23858, 32768, 0 },
        { 277, 4369, 5255, 8905, 16465, 22271, 32768, 0 },
        { 3409, 5436, 10599, 15599, 19687, 24040, 32768, 0 }
    },
    {
        { 1870, 13742, 14530, 16498, 23770, 27698, 32768, 0 },
        { 326, 8796, 14632, 15079, 19272, 27486, 32768, 0 },
        { 484, 7576, 7712, 14443, 19159, 22591, 32768, 0 },
        { 1126, 15340, 15895, 17023, 20896, 30279, 32768, 0 },
        { 655, 4854, 5249, 5913, 22099, 27138, 32768, 0 },
        { 1299, 6458, 8885, 9290, 14851, 25497, 32768, 0 },
        { 311, 5295, 5552, 6885, 16107, 22672, 32768, 0 },
        { 883, 8059, 8270, 11258, 17289, 21549, 32768, 0 },
        { 741, 7580, 9318, 10345, 16688, 29046, 32768, 0 },
        { 110, 7406, 7915, 9195, 16041, 23329, 32768, 0 },
        { 363, 7974, 9357, 10673, 15629, 24474, 32768, 0 },
        { 153, 7647, 8112, 9936, 15307, 19996, 32768, 0 },
        { 3511, 6332, 11165, 15335, 19323, 23594, 32768, 0 }
    }
}

```

```

Default_Intra_Tx_Type_Set2_Cdf[ 3 ][ INTRA_MODES ][ 6 ] = {
    {
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
    },
    {
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
        { 6554, 13107, 19661, 26214, 32768, 0 },
    },
    {
        { 1127, 12814, 22772, 27483, 32768, 0 },
        { 145, 6761, 11980, 26667, 32768, 0 },
        { 362, 5887, 11678, 16725, 32768, 0 },
        { 385, 15213, 18587, 30693, 32768, 0 },
        { 25, 2914, 23134, 27903, 32768, 0 },
        { 60, 4470, 11749, 23991, 32768, 0 },
        { 37, 3332, 14511, 21448, 32768, 0 },
        { 157, 6320, 13036, 17439, 32768, 0 },
        { 119, 6719, 12906, 29396, 32768, 0 },
        { 47, 5537, 12576, 21499, 32768, 0 },
        { 269, 6076, 11258, 23115, 32768, 0 },
        { 83, 5615, 12001, 17228, 32768, 0 },
        { 1968, 5556, 12023, 18547, 32768, 0 }
    }
}

```

```

Default_Inter_Tx_Type_Set1_Cdf[ 2 ][ 17 ] = {
    { 4458, 5560, 7695, 9709, 13330, 14789, 17537, 20266, 21504,
      22848, 23934, 25474, 27727, 28915, 30631, 32768, 0 },
    { 1645, 2573, 4778, 5711, 7807, 8622, 10522, 15357, 17674,
      20408, 22517, 25010, 27116, 28856, 30749, 32768, 0 },
    { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432,
      20480, 22528, 24576, 26624, 28672, 30720, 32768, 0 },
    { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432,
      20480, 22528, 24576, 26624, 28672, 30720, 32768, 0 }
}

```

```

Default_Inter_Tx_Type_Set2_Cdf[ 13 ] = {
    770, 2421, 5225, 12907, 15819, 18927, 21561, 24089, 26595,
    28526, 30529, 32768, 0
}

```

```

Default_Inter_Tx_Type_Set3_Cdf[ 4 ][ 3 ] = {
    { 16384, 32768, 0 },
    { 4167, 32768, 0 },
    { 1998, 32768, 0 },
    { 748, 32768, 0 }
}

```

```

Default_Compound_Idx_Cdf[ COMPOUND_IDX_CONTEXTS ][ 3 ] = {
    { 18244, 32768, 0 },
    { 12865, 32768, 0 },
    { 7053, 32768, 0 },
    { 13259, 32768, 0 },
    { 9334, 32768, 0 },
    { 4644, 32768, 0 }
}

```

```

Default_Comp_Group_Idx_Cdf[ COMP_GROUP_IDX_CONTEXTS ][ 3 ] = {
    { 26607, 32768, 0 },
    { 22891, 32768, 0 },
    { 18840, 32768, 0 },
    { 24594, 32768, 0 },
    { 19934, 32768, 0 },
    { 22674, 32768, 0 }
}

```

```

Default_Compound_Type_Cdf[ BLOCK_SIZES ][ COMPOUND_TYPES + 1 ] = {
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 23431, 32768, 0 },
  { 13171, 32768, 0 },
  { 11470, 32768, 0 },
  { 9770, 32768, 0 },
  { 9100, 32768, 0 },
  { 8233, 32768, 0 },
  { 6172, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 11820, 32768, 0 },
  { 7701, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 }
}

```

**Note:** Indices 0 to 2, 10 to 17, and 20 to 21 inclusive are never used in the first dimension of the Default\_Compound\_Type\_Cdf CDF table.

```

Default_Inter_Intra_Cdf[ BLOCK_SIZE_GROUPS - 1 ][ 3 ] = {
  { 26887, 32768, 0 },
  { 27597, 32768, 0 },
  { 30237, 32768, 0 }
}

```

```

Default_Inter_Intra_Mode_Cdf[ BLOCK_SIZE_GROUPS - 1 ][ INTERINTRA_MODES + 1 ] = {
  { 1875, 11082, 27332, 32768, 0 },
  { 2473, 9996, 26388, 32768, 0 },
  { 4238, 11537, 25926, 32768, 0 }
}

```

```

Default_Wedge_Index_Cdf[ BLOCK_SIZES ][ 16+1 ] = {
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2438, 4440, 6599, 8663, 11005, 12874, 15751, 18094, 20359, 22362, 24127, 25702, 27752, 29450,
    31171, 32768, 0 },
  { 806, 3266, 6005, 6738, 7218, 7367, 7771, 14588, 16323, 17367, 18452, 19422, 22839, 26127, 29629,
    32768, 0 },
  { 2779, 3738, 4683, 7213, 7775, 8017, 8655, 14357, 17939, 21332, 24520, 27470, 29456, 30529, 31656,
    32768, 0 },
  { 1684, 3625, 5675, 7108, 9302, 11274, 14429, 17144, 19163, 20961, 22884, 24471, 26719, 28714,
    30877, 32768, 0 },
  { 1142, 3491, 6277, 7314, 8089, 8355, 9023, 13624, 15369, 16730, 18114, 19313, 22521, 26012, 29550,
    32768, 0 },
  { 2742, 4195, 5727, 8035, 8980, 9336, 10146, 14124, 17270, 20533, 23434, 25972, 27944, 29570,
    31416, 32768, 0 },
  { 1727, 3948, 6101, 7796, 9841, 12344, 15766, 18944, 20638, 22038, 23963, 25311, 26988, 28766,
    31012, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 154, 987, 1925, 2051, 2088, 2111, 2151, 23033, 23703, 24284, 24985, 25684, 27259, 28883, 30911,
    32768, 0 },
  { 1135, 1322, 1493, 2635, 2696, 2737, 2770, 21016, 22935, 25057, 27251, 29173, 30089, 30960, 31933,
    32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 },
  { 2048, 4096, 6144, 8192, 10240, 12288, 14336, 16384, 18432, 20480, 22528, 24576, 26624, 28672,
    30720, 32768, 0 }
}

```

**Note:** Indices 0 to 2, 10 to 17, and 20 to 21 inclusive are never used in the first dimension of the Default\_Wedge\_Index\_Cdf CDF table.

[illegible]

**Note:** Only indices 3 to 9 inclusive are used in the first dimension of the Default\_Wedge\_Inter\_Intra\_Cdf CDF table.

```

Default_Use_Obmc_Cdf[ BLOCK_SIZES ][ 3 ] = {
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 10437, 32768, 0 },
  { 9371, 32768, 0 },
  { 9301, 32768, 0 },
  { 17432, 32768, 0 },
  { 14423, 32768, 0 },
  { 15142, 32768, 0 },
  { 25817, 32768, 0 },
  { 22823, 32768, 0 },
  { 22083, 32768, 0 },
  { 30128, 32768, 0 },
  { 31014, 32768, 0 },
  { 31560, 32768, 0 },
  { 32638, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 23664, 32768, 0 },
  { 20901, 32768, 0 },
  { 24008, 32768, 0 },
  { 26879, 32768, 0 }
}

```

**Note:** Indices 0 to 2 and 16 to 17 inclusive are never used in the first dimension of the Default\_Use\_Obmc\_Cdf CDF table.

```

Default_Comp_Ref_Type_Cdf[ COMP_REF_TYPE_CONTEXTS ][ 3 ] = {
  { 1198, 32768, 0 },
  { 2070, 32768, 0 },
  { 9166, 32768, 0 },
  { 7499, 32768, 0 },
  { 22475, 32768, 0 }
}

```

```

Default_Uni_Comp_Ref_Cdf[ REF_CONTEXTS ][ UNIDIR_COMP_REFS - 1 ][ 3 ] = {
  { { 5284, 32768, 0 }, { 3865, 32768, 0 }, { 3128, 32768, 0 } },
  { { 23152, 32768, 0 }, { 14173, 32768, 0 }, { 15270, 32768, 0 } },
  { { 31774, 32768, 0 }, { 25120, 32768, 0 }, { 26710, 32768, 0 } }
}

```

```
Default_Cfl_Sign_Cdf[ CFL_JOINT_SIGNS + 1 ] = {
    1418, 2123, 13340, 18405, 26972, 28343, 32294, 32768, 0
}
```

```
Default_Cfl_Alpha_Cdf[ CFL_ALPHA_CONTEXTS ][ CFL_ALPHABET_SIZE + 1 ] = {
    { 7637, 20719, 31401, 32481, 32657, 32688, 32692, 32696, 32700,
      32704, 32708, 32712, 32716, 32720, 32724, 32768, 0 },
    { 14365, 23603, 28135, 31168, 32167, 32395, 32487, 32573, 32620,
      32647, 32668, 32672, 32676, 32680, 32684, 32768, 0 },
    { 11532, 22380, 28445, 31360, 32349, 32523, 32584, 32649, 32673,
      32677, 32681, 32685, 32689, 32693, 32697, 32768, 0 },
    { 26990, 31402, 32282, 32571, 32692, 32696, 32700, 32704, 32708,
      32712, 32716, 32720, 32724, 32728, 32732, 32768, 0 },
    { 17248, 26058, 28904, 30608, 31305, 31877, 32126, 32321, 32394,
      32464, 32516, 32560, 32576, 32593, 32622, 32768, 0 },
    { 14738, 21678, 25779, 27901, 29024, 30302, 30980, 31843, 32144,
      32413, 32520, 32594, 32622, 32656, 32660, 32768, 0 }
}
```

```
Default_Use_Wiener_Cdf[ 2 + 1 ] = {
    11570, 32768, 0
}
```

```
Default_Use_Sgrproj_Cdf[ 2 + 1 ] = {
    16855, 32768, 0
}
```

```
Default_Restoration_Type_Cdf[ RESTORE_SWITCHABLE + 1 ] = {
    9413, 22581, 32768, 0
}
```



```

Default_Txb_Skip_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ TXB_SKIP_CONTEXTS ][ 3 ] = {
{
  { { 31849, 32768, 0 },
    { 5892, 32768, 0 },
    { 12112, 32768, 0 },
    { 21935, 32768, 0 },
    { 20289, 32768, 0 },
    { 27473, 32768, 0 },
    { 32487, 32768, 0 },
    { 7654, 32768, 0 },
    { 19473, 32768, 0 },
    { 29984, 32768, 0 },
    { 9961, 32768, 0 },
    { 30242, 32768, 0 },
    { 32117, 32768, 0 } },
  { { 31548, 32768, 0 },
    { 1549, 32768, 0 },
    { 10130, 32768, 0 },
    { 16656, 32768, 0 },
    { 18591, 32768, 0 },
    { 26308, 32768, 0 },
    { 32537, 32768, 0 },
    { 5403, 32768, 0 },
    { 18096, 32768, 0 },
    { 30003, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 29957, 32768, 0 },
    { 5391, 32768, 0 },
    { 18039, 32768, 0 },
    { 23566, 32768, 0 },
    { 22431, 32768, 0 },
    { 25822, 32768, 0 },
    { 32197, 32768, 0 },
    { 3778, 32768, 0 },
    { 15336, 32768, 0 },
    { 28981, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 17920, 32768, 0 },
    { 1818, 32768, 0 },
    { 7282, 32768, 0 },
    { 25273, 32768, 0 },
    { 10923, 32768, 0 },
    { 31554, 32768, 0 },
    { 32624, 32768, 0 },
    { 1366, 32768, 0 },
    { 15628, 32768, 0 },
  }
}

```

```

    { 30462, 32768, 0 },
    { 146, 32768, 0 },
    { 5132, 32768, 0 },
    { 31657, 32768, 0 } },
  { { 6308, 32768, 0 },
    { 117, 32768, 0 },
    { 1638, 32768, 0 },
    { 2161, 32768, 0 },
    { 16384, 32768, 0 },
    { 10923, 32768, 0 },
    { 30247, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } }
},
{
  { { 30371, 32768, 0 },
    { 7570, 32768, 0 },
    { 13155, 32768, 0 },
    { 20751, 32768, 0 },
    { 20969, 32768, 0 },
    { 27067, 32768, 0 },
    { 32013, 32768, 0 },
    { 5495, 32768, 0 },
    { 17942, 32768, 0 },
    { 28280, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 31782, 32768, 0 },
    { 1836, 32768, 0 },
    { 10689, 32768, 0 },
    { 17604, 32768, 0 },
    { 21622, 32768, 0 },
    { 27518, 32768, 0 },
    { 32399, 32768, 0 },
    { 4419, 32768, 0 },
    { 16294, 32768, 0 },
    { 28345, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 31901, 32768, 0 },
    { 10311, 32768, 0 },
    { 18047, 32768, 0 },
    { 24806, 32768, 0 },
    { 23288, 32768, 0 },

```

```

    { 27914, 32768, 0 },
    { 32296, 32768, 0 },
    { 4215, 32768, 0 },
    { 15756, 32768, 0 },
    { 28341, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 26726, 32768, 0 },
    { 1045, 32768, 0 },
    { 11703, 32768, 0 },
    { 20590, 32768, 0 },
    { 18554, 32768, 0 },
    { 25970, 32768, 0 },
    { 31938, 32768, 0 },
    { 5583, 32768, 0 },
    { 21313, 32768, 0 },
    { 29390, 32768, 0 },
    { 641, 32768, 0 },
    { 22265, 32768, 0 },
    { 31452, 32768, 0 } },
  { { 26584, 32768, 0 },
    { 188, 32768, 0 },
    { 8847, 32768, 0 },
    { 24519, 32768, 0 },
    { 22938, 32768, 0 },
    { 30583, 32768, 0 },
    { 32608, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } }
},
{
  { { 29614, 32768, 0 },
    { 9068, 32768, 0 },
    { 12924, 32768, 0 },
    { 19538, 32768, 0 },
    { 17737, 32768, 0 },
    { 24619, 32768, 0 },
    { 30642, 32768, 0 },
    { 4119, 32768, 0 },
    { 16026, 32768, 0 },
    { 25657, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 31957, 32768, 0 },

```

$$\begin{aligned} & \{ 3230, 32768, 0 \}, \\ & \{ 11153, 32768, 0 \}, \\ & \{ 18123, 32768, 0 \}, \\ & \{ 20143, 32768, 0 \}, \\ & \{ 26536, 32768, 0 \}, \\ & \{ 31986, 32768, 0 \}, \\ & \{ 3050, 32768, 0 \}, \\ & \{ 14603, 32768, 0 \}, \\ & \{ 25155, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \} \}, \\ \{ & \{ 32363, 32768, 0 \}, \\ & \{ 10692, 32768, 0 \}, \\ & \{ 19090, 32768, 0 \}, \\ & \{ 24357, 32768, 0 \}, \\ & \{ 24442, 32768, 0 \}, \\ & \{ 28312, 32768, 0 \}, \\ & \{ 32169, 32768, 0 \}, \\ & \{ 3648, 32768, 0 \}, \\ & \{ 15690, 32768, 0 \}, \\ & \{ 26815, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \} \}, \\ \{ & \{ 30669, 32768, 0 \}, \\ & \{ 3832, 32768, 0 \}, \\ & \{ 11663, 32768, 0 \}, \\ & \{ 18889, 32768, 0 \}, \\ & \{ 19782, 32768, 0 \}, \\ & \{ 23313, 32768, 0 \}, \\ & \{ 31330, 32768, 0 \}, \\ & \{ 5124, 32768, 0 \}, \\ & \{ 18719, 32768, 0 \}, \\ & \{ 28468, 32768, 0 \}, \\ & \{ 3082, 32768, 0 \}, \\ & \{ 20982, 32768, 0 \}, \\ & \{ 29443, 32768, 0 \} \}, \\ \{ & \{ 28573, 32768, 0 \}, \\ & \{ 3183, 32768, 0 \}, \\ & \{ 17802, 32768, 0 \}, \\ & \{ 25977, 32768, 0 \}, \\ & \{ 26677, 32768, 0 \}, \\ & \{ 27832, 32768, 0 \}, \\ & \{ 32387, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \\ & \{ 16384, 32768, 0 \}, \end{aligned}$$

```

    { 16384, 32768, 0 } }
},
{
  { { 26887, 32768, 0 },
    { 6729, 32768, 0 },
    { 10361, 32768, 0 },
    { 17442, 32768, 0 },
    { 15045, 32768, 0 },
    { 22478, 32768, 0 },
    { 29072, 32768, 0 },
    { 2713, 32768, 0 },
    { 11861, 32768, 0 },
    { 20773, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 31903, 32768, 0 },
    { 2044, 32768, 0 },
    { 7528, 32768, 0 },
    { 14618, 32768, 0 },
    { 16182, 32768, 0 },
    { 24168, 32768, 0 },
    { 31037, 32768, 0 },
    { 2786, 32768, 0 },
    { 11194, 32768, 0 },
    { 20155, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32510, 32768, 0 },
    { 8430, 32768, 0 },
    { 17318, 32768, 0 },
    { 24154, 32768, 0 },
    { 23674, 32768, 0 },
    { 28789, 32768, 0 },
    { 32139, 32768, 0 },
    { 3440, 32768, 0 },
    { 13117, 32768, 0 },
    { 22702, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 31671, 32768, 0 },
    { 2056, 32768, 0 },
    { 11746, 32768, 0 },
    { 16852, 32768, 0 },
    { 18635, 32768, 0 },
    { 24715, 32768, 0 },
    { 31484, 32768, 0 },
    { 4656, 32768, 0 },

```

```

    { 16074, 32768, 0 },
    { 24704, 32768, 0 },
    { 1806, 32768, 0 },
    { 14645, 32768, 0 },
    { 25336, 32768, 0 } },
{ { 31539, 32768, 0 },
  { 8433, 32768, 0 },
  { 20576, 32768, 0 },
  { 27904, 32768, 0 },
  { 27852, 32768, 0 },
  { 30026, 32768, 0 },
  { 32441, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 },
  { 16384, 32768, 0 } }
}
}

```

```

Default_Eob_Pt_16_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 6 ] = {
{ { { 840, 1039, 1980, 4895, 32768, 0 },
    { 370, 671, 1883, 4471, 32768, 0 } },
  { { 3247, 4950, 9688, 14563, 32768, 0 },
    { 1904, 3354, 7763, 14647, 32768, 0 } } },
{ { { 2125, 2551, 5165, 8946, 32768, 0 },
    { 513, 765, 1859, 6339, 32768, 0 } },
  { { 7637, 9498, 14259, 19108, 32768, 0 },
    { 2497, 4096, 8866, 16993, 32768, 0 } } },
{ { { 4016, 4897, 8881, 14968, 32768, 0 },
    { 716, 1105, 2646, 10056, 32768, 0 } },
  { { 11139, 13270, 18241, 23566, 32768, 0 },
    { 3192, 5032, 10297, 19755, 32768, 0 } } },
{ { { 6708, 8958, 14746, 22133, 32768, 0 },
    { 1222, 2074, 4783, 15410, 32768, 0 } },
  { { 19575, 21766, 26044, 29709, 32768, 0 },
    { 7297, 10767, 19273, 28194, 32768, 0 } } }
}

```

```

Default_Eob_Pt_32_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 7 ] = {
  { { { 400, 520, 977, 2102, 6542, 32768, 0 },
      { 210, 405, 1315, 3326, 7537, 32768, 0 } },
    { { { 2636, 4273, 7588, 11794, 20401, 32768, 0 },
      { 1786, 3179, 6902, 11357, 19054, 32768, 0 } } },
    { { { 989, 1249, 2019, 4151, 10785, 32768, 0 },
      { 313, 441, 1099, 2917, 8562, 32768, 0 } },
      { { { 8394, 10352, 13932, 18855, 26014, 32768, 0 },
        { 2578, 4124, 8181, 13670, 24234, 32768, 0 } } },
      { { { 2515, 3003, 4452, 8162, 16041, 32768, 0 },
        { 574, 821, 1836, 5089, 13128, 32768, 0 } } },
      { { { 13468, 16303, 20361, 25105, 29281, 32768, 0 },
        { 3542, 5502, 10415, 16760, 25644, 32768, 0 } } } },
    { { { { 4617, 5709, 8446, 13584, 23135, 32768, 0 },
          { 1156, 1702, 3675, 9274, 20539, 32768, 0 } },
        { { 22086, 24282, 27010, 29770, 31743, 32768, 0 },
          { 7699, 10897, 20891, 26926, 31628, 32768, 0 } } } } }
}

```

```

Default_Eob_Pt_64_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 8 ] = {
  { { { { 329, 498, 1101, 1784, 3265, 7758, 32768, 0 },
        { 335, 730, 1459, 5494, 8755, 12997, 32768, 0 } },
      { { { 3505, 5304, 10086, 13814, 17684, 23370, 32768, 0 },
        { 1563, 2700, 4876, 10911, 14706, 22480, 32768, 0 } } },
      { { { { 1260, 1446, 2253, 3712, 6652, 13369, 32768, 0 },
            { 401, 605, 1029, 2563, 5845, 12626, 32768, 0 } },
        { { { 8609, 10612, 14624, 18714, 22614, 29024, 32768, 0 },
          { 1923, 3127, 5867, 9703, 14277, 27100, 32768, 0 } } } },
      { { { { 2374, 2772, 4583, 7276, 12288, 19706, 32768, 0 },
            { 497, 810, 1315, 3000, 7004, 15641, 32768, 0 } },
        { { { 15050, 17126, 21410, 24886, 28156, 30726, 32768, 0 },
          { 4034, 6290, 10235, 14982, 21214, 28491, 32768, 0 } } } },
      { { { { 6307, 7541, 12060, 16358, 22553, 27865, 32768, 0 },
            { 1289, 2320, 3971, 7926, 14153, 24291, 32768, 0 } },
        { { { 24212, 25708, 28268, 30035, 31307, 32049, 32768, 0 },
          { 8726, 12378, 19409, 26450, 30038, 32462, 32768, 0 } } } } }
}

```

```

Default_Eob_Pt_128_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 9 ] = {
  { { { 219, 482, 1140, 2091, 3680, 6028, 12586, 32768, 0 },
      { 371, 699, 1254, 4830, 9479, 12562, 17497, 32768, 0 } },
    { { { 5245, 7456, 12880, 15852, 20033, 23932, 27608, 32768, 0 },
      { 2054, 3472, 5869, 14232, 18242, 20590, 26752, 32768, 0 } } },
    { { { 685, 933, 1488, 2714, 4766, 8562, 19254, 32768, 0 },
      { 217, 352, 618, 2303, 5261, 9969, 17472, 32768, 0 } },
      { { { 8045, 11200, 15497, 19595, 23948, 27408, 30938, 32768, 0 },
        { 2310, 4160, 7471, 14997, 17931, 20768, 30240, 32768, 0 } } },
      { { { 1366, 1738, 2527, 5016, 9355, 15797, 24643, 32768, 0 },
        { 354, 558, 944, 2760, 7287, 14037, 21779, 32768, 0 } },
        { { { 13627, 16246, 20173, 24429, 27948, 30415, 31863, 32768, 0 },
          { 6275, 9889, 14769, 23164, 27988, 30493, 32272, 32768, 0 } } },
          { { { 3472, 4885, 7489, 12481, 18517, 24536, 29635, 32768, 0 },
            { 886, 1731, 3271, 8469, 15569, 22126, 28383, 32768, 0 } },
            { { { 24313, 26062, 28385, 30107, 31217, 31898, 32345, 32768, 0 },
              { 9165, 13282, 21150, 30286, 31894, 32571, 32712, 32768, 0 } } } } }
}

```

```

Default_Eob_Pt_256_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 10 ] = {
  { { { { 310, 584, 1887, 3589, 6168, 8611, 11352, 15652, 32768, 0 },
        { 998, 1850, 2998, 5604, 17341, 19888, 22899, 25583, 32768, 0 } },
      { { { 2520, 3240, 5952, 8870, 12577, 17558, 19954, 24168, 32768, 0 },
        { 2203, 4130, 7435, 10739, 20652, 23681, 25609, 27261, 32768, 0 } } },
      { { { 1448, 2109, 4151, 6263, 9329, 13260, 17944, 23300, 32768, 0 },
        { 399, 1019, 1749, 3038, 10444, 15546, 22739, 27294, 32768, 0 } },
        { { { 6402, 8148, 12623, 15072, 18728, 22847, 26447, 29377, 32768, 0 },
          { 1674, 3252, 5734, 10159, 22397, 23802, 24821, 30940, 32768, 0 } } } },
      { { { { 3089, 3920, 6038, 9460, 14266, 19881, 25766, 29176, 32768, 0 },
        { 1084, 2358, 3488, 5122, 11483, 18103, 26023, 29799, 32768, 0 } },
        { { { 11514, 13794, 17480, 20754, 24361, 27378, 29492, 31277, 32768, 0 },
          { 6571, 9610, 15516, 21826, 29092, 30829, 31842, 32708, 32768, 0 } } } },
      { { { { 5348, 7113, 11820, 15924, 22106, 26777, 30334, 31757, 32768, 0 },
        { 2453, 4474, 6307, 8777, 16474, 22975, 29000, 31547, 32768, 0 } },
        { { { 23110, 24597, 27140, 28894, 30167, 30927, 31392, 32094, 32768, 0 },
          { 9998, 17661, 25178, 28097, 31308, 32038, 32403, 32695, 32768, 0 } } } } }
}

```



```

Default_Eob_Pt_512_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 11 ] = {
  { { 641, 983, 3707, 5430, 10234, 14958, 18788, 23412, 26061, 32768, 0 },
    { 5095, 6446, 9996, 13354, 16017, 17986, 20919, 26129, 29140, 32768, 0 } },
  { { 1230, 2278, 5035, 7776, 11871, 15346, 19590, 24584, 28749, 32768, 0 },
    { 7265, 9979, 15819, 19250, 21780, 23846, 26478, 28396, 31811, 32768, 0 } },
  { { 2624, 3936, 6480, 9686, 13979, 17726, 23267, 28410, 31078, 32768, 0 },
    { 12015, 14769, 19588, 22052, 24222, 25812, 27300, 29219, 32114, 32768, 0 } },
  { { 5927, 7809, 10923, 14597, 19439, 24135, 28456, 31142, 32060, 32768, 0 },
    { 21093, 23043, 25742, 27658, 29097, 29716, 30073, 30820, 31956, 32768, 0 } }
}

```

```

Default_Eob_Pt_1024_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 12 ] = {
  { { 393, 421, 751, 1623, 3160, 6352, 13345, 18047, 22571, 25830, 32768, 0 },
    { 1865, 1988, 2930, 4242, 10533, 16538, 21354, 27255, 28546, 31784, 32768, 0 } },
  { { 696, 948, 3145, 5702, 9706, 13217, 17851, 21856, 25692, 28034, 32768, 0 },
    { 2672, 3591, 9330, 17084, 22725, 24284, 26527, 28027, 28377, 30876, 32768, 0 } },
  { { 2784, 3831, 7041, 10521, 14847, 18844, 23155, 26682, 29229, 31045, 32768, 0 },
    { 9577, 12466, 17739, 20750, 22061, 23215, 24601, 25483, 25843, 32056, 32768, 0 } },
  { { 6698, 8334, 11961, 15762, 20186, 23862, 27434, 29326, 31082, 32050, 32768, 0 },
    { 20569, 22426, 25569, 26859, 28053, 28913, 29486, 29724, 29807, 32570, 32768, 0 } }
}

```

```

Default_Eob_Extra_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ EOB_COEF_CONTEXTS ][ 3 ] = {
  { { { { 16961, 32768, 0 }, { 17223, 32768, 0 }, { 7621, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      },
    { { { 19069, 32768, 0 }, { 22525, 32768, 0 }, { 13377, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      }
    } },
  { { { { 20401, 32768, 0 }, { 17025, 32768, 0 }, { 12845, 32768, 0 },
        { 12873, 32768, 0 }, { 14094, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      },
    { { { 20681, 32768, 0 }, { 20701, 32768, 0 }, { 15250, 32768, 0 },
        { 15017, 32768, 0 }, { 14928, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      }
    } },
  { { { { 23905, 32768, 0 }, { 17194, 32768, 0 }, { 16170, 32768, 0 },
        { 17695, 32768, 0 }, { 13826, 32768, 0 }, { 15810, 32768, 0 },
        { 12036, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      },
    { { { 23959, 32768, 0 }, { 20799, 32768, 0 }, { 19021, 32768, 0 },
        { 16203, 32768, 0 }, { 17886, 32768, 0 }, { 14144, 32768, 0 },
        { 12010, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      }
    } },
  { { { { 27399, 32768, 0 }, { 16327, 32768, 0 }, { 18071, 32768, 0 },
        { 19584, 32768, 0 }, { 20721, 32768, 0 }, { 18432, 32768, 0 },
        { 19560, 32768, 0 }, { 10150, 32768, 0 }, { 8805, 32768, 0 }
      },
    { { { 24932, 32768, 0 }, { 20833, 32768, 0 }, { 12027, 32768, 0 },
        { 16670, 32768, 0 }, { 19914, 32768, 0 }, { 15106, 32768, 0 },
        { 17662, 32768, 0 }, { 13783, 32768, 0 }, { 28756, 32768, 0 }
      }
    } },
  { { { { 23406, 32768, 0 }, { 21845, 32768, 0 }, { 18432, 32768, 0 },
        { 16384, 32768, 0 }, { 17096, 32768, 0 }, { 12561, 32768, 0 },
        { 17320, 32768, 0 }, { 22395, 32768, 0 }, { 21370, 32768, 0 }
      },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      }
    } },
  { { { { 17471, 32768, 0 }, { 20223, 32768, 0 }, { 11357, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      },
    { { { 20335, 32768, 0 }, { 21667, 32768, 0 }, { 14818, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
        { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
      }
    } },
  { { { { 20430, 32768, 0 }, { 20662, 32768, 0 }, { 15367, 32768, 0 },

```

```

    { 16970, 32768, 0 }, { 14657, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  },
  { { 22117, 32768, 0 }, { 22028, 32768, 0 }, { 18650, 32768, 0 },
    { 16042, 32768, 0 }, { 15885, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } },
{ { { 22409, 32768, 0 }, { 21012, 32768, 0 }, { 15650, 32768, 0 },
    { 17395, 32768, 0 }, { 15469, 32768, 0 }, { 20205, 32768, 0 },
    { 19511, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  },
  { { 24220, 32768, 0 }, { 22480, 32768, 0 }, { 17737, 32768, 0 },
    { 18916, 32768, 0 }, { 19268, 32768, 0 }, { 18412, 32768, 0 },
    { 18844, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } },
{ { { 25991, 32768, 0 }, { 20314, 32768, 0 }, { 17731, 32768, 0 },
    { 19678, 32768, 0 }, { 18649, 32768, 0 }, { 17307, 32768, 0 },
    { 21798, 32768, 0 }, { 17549, 32768, 0 }, { 15630, 32768, 0 }
  },
  { { 26585, 32768, 0 }, { 21469, 32768, 0 }, { 20432, 32768, 0 },
    { 17735, 32768, 0 }, { 19280, 32768, 0 }, { 15235, 32768, 0 },
    { 20297, 32768, 0 }, { 22471, 32768, 0 }, { 28997, 32768, 0 }
  } },
{ { { 26605, 32768, 0 }, { 11304, 32768, 0 }, { 16726, 32768, 0 },
    { 16560, 32768, 0 }, { 20866, 32768, 0 }, { 23524, 32768, 0 },
    { 19878, 32768, 0 }, { 13469, 32768, 0 }, { 23084, 32768, 0 }
  },
  { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } } },
{ { { { 18983, 32768, 0 }, { 20512, 32768, 0 }, { 14885, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  },
  { { 20090, 32768, 0 }, { 19444, 32768, 0 }, { 17286, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } },
{ { { 19139, 32768, 0 }, { 21487, 32768, 0 }, { 18959, 32768, 0 },
    { 20910, 32768, 0 }, { 19089, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  },
  { { 20536, 32768, 0 }, { 20664, 32768, 0 }, { 20625, 32768, 0 },
    { 19123, 32768, 0 }, { 14862, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } },
{ { { 19833, 32768, 0 }, { 21502, 32768, 0 }, { 17485, 32768, 0 },
    { 20267, 32768, 0 }, { 18353, 32768, 0 }, { 23329, 32768, 0 },
    { 21478, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  }

```

```

    },
    { { 22041, 32768, 0 }, { 23434, 32768, 0 }, { 20001, 32768, 0 },
      { 20554, 32768, 0 }, { 20951, 32768, 0 }, { 20145, 32768, 0 },
      { 15562, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    } },
  { { { 23312, 32768, 0 }, { 21607, 32768, 0 }, { 16526, 32768, 0 },
      { 18957, 32768, 0 }, { 18034, 32768, 0 }, { 18934, 32768, 0 },
      { 24247, 32768, 0 }, { 16921, 32768, 0 }, { 17080, 32768, 0 }
    },
    { { 26579, 32768, 0 }, { 24910, 32768, 0 }, { 18637, 32768, 0 },
      { 19800, 32768, 0 }, { 20388, 32768, 0 }, { 9887, 32768, 0 },
      { 15642, 32768, 0 }, { 30198, 32768, 0 }, { 24721, 32768, 0 }
    } },
  { { { 26998, 32768, 0 }, { 16737, 32768, 0 }, { 17838, 32768, 0 },
      { 18922, 32768, 0 }, { 19515, 32768, 0 }, { 18636, 32768, 0 },
      { 17333, 32768, 0 }, { 15776, 32768, 0 }, { 22658, 32768, 0 }
    },
    { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    } } },
  { { { 20177, 32768, 0 }, { 20789, 32768, 0 }, { 20262, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    },
    { { 21416, 32768, 0 }, { 20855, 32768, 0 }, { 23410, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    } },
  { { { 20238, 32768, 0 }, { 21057, 32768, 0 }, { 19159, 32768, 0 },
      { 22337, 32768, 0 }, { 20159, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    },
    { { 20125, 32768, 0 }, { 20559, 32768, 0 }, { 21707, 32768, 0 },
      { 22296, 32768, 0 }, { 17333, 32768, 0 }, { 16384, 32768, 0 },
      { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    } },
  { { { 19941, 32768, 0 }, { 20527, 32768, 0 }, { 21470, 32768, 0 },
      { 22487, 32768, 0 }, { 19558, 32768, 0 }, { 22354, 32768, 0 },
      { 20331, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    },
    { { 22752, 32768, 0 }, { 25006, 32768, 0 }, { 22075, 32768, 0 },
      { 21576, 32768, 0 }, { 17740, 32768, 0 }, { 21690, 32768, 0 },
      { 19211, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
    } },
  { { { 21442, 32768, 0 }, { 22358, 32768, 0 }, { 18503, 32768, 0 },
      { 20291, 32768, 0 }, { 19945, 32768, 0 }, { 21294, 32768, 0 },
      { 21178, 32768, 0 }, { 19400, 32768, 0 }, { 10556, 32768, 0 }
    },
    { { 24648, 32768, 0 }, { 24949, 32768, 0 }, { 20708, 32768, 0 },

```

```

    { 23905, 32768, 0 }, { 20501, 32768, 0 }, { 9558, 32768, 0 },
    { 9423, 32768, 0 }, { 30365, 32768, 0 }, { 19253, 32768, 0 }
  } },
{ { { 26064, 32768, 0 }, { 22098, 32768, 0 }, { 19613, 32768, 0 },
    { 20525, 32768, 0 }, { 17595, 32768, 0 }, { 16618, 32768, 0 },
    { 20497, 32768, 0 }, { 18989, 32768, 0 }, { 15513, 32768, 0 }
  },
  { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 }
  } } }
}

```

```

DefaultDc_Sign_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ DC_SIGN_CONTEXTS ][ 3 ] = {
  { {
    { 128 * 125, 32768, 0 },
    { 128 * 102, 32768, 0 },
    { 128 * 147, 32768, 0 },
  },
    {
    { 128 * 119, 32768, 0 },
    { 128 * 101, 32768, 0 },
    { 128 * 135, 32768, 0 },
  } },
  { {
    { 128 * 125, 32768, 0 },
    { 128 * 102, 32768, 0 },
    { 128 * 147, 32768, 0 },
  },
    {
    { 128 * 119, 32768, 0 },
    { 128 * 101, 32768, 0 },
    { 128 * 135, 32768, 0 },
  } },
  { {
    { 128 * 125, 32768, 0 },
    { 128 * 102, 32768, 0 },
    { 128 * 147, 32768, 0 },
  },
    {
    { 128 * 119, 32768, 0 },
    { 128 * 101, 32768, 0 },
    { 128 * 135, 32768, 0 },
  } },
  { {
    { 128 * 125, 32768, 0 },
    { 128 * 102, 32768, 0 },
    { 128 * 147, 32768, 0 },
  },
    {
    { 128 * 119, 32768, 0 },
    { 128 * 101, 32768, 0 },
    { 128 * 135, 32768, 0 },
  } }
}

```

```

Default_Coeff_Base_Eob_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ SIG_COEF_CONTEXTS_EOB ][ 4
] = {
  { { { { 17837, 29055, 32768, 0 },
        { 29600, 31446, 32768, 0 },
        { 30844, 31878, 32768, 0 },
        { 24926, 28948, 32768, 0 } },
    { { 21365, 30026, 32768, 0 },
      { 30512, 32423, 32768, 0 },
      { 31658, 32621, 32768, 0 },
      { 29630, 31881, 32768, 0 } } },
  { { { 5717, 26477, 32768, 0 },
      { 30491, 31703, 32768, 0 },
      { 31550, 32158, 32768, 0 },
      { 29648, 31491, 32768, 0 } },
    { { 12608, 27820, 32768, 0 },
      { 30680, 32225, 32768, 0 },
      { 30809, 32335, 32768, 0 },
      { 31299, 32423, 32768, 0 } } },
  { { { 1786, 12612, 32768, 0 },
      { 30663, 31625, 32768, 0 },
      { 32339, 32468, 32768, 0 },
      { 31148, 31833, 32768, 0 } },
    { { 18857, 23865, 32768, 0 },
      { 31428, 32428, 32768, 0 },
      { 31744, 32373, 32768, 0 },
      { 31775, 32526, 32768, 0 } } },
  { { { 1787, 2532, 32768, 0 },
      { 30832, 31662, 32768, 0 },
      { 31824, 32682, 32768, 0 },
      { 32133, 32569, 32768, 0 } },
    { { 13751, 22235, 32768, 0 },
      { 32089, 32409, 32768, 0 },
      { 27084, 27920, 32768, 0 },
      { 29291, 32594, 32768, 0 } } },
  { { { 1725, 3449, 32768, 0 },
      { 31102, 31935, 32768, 0 },
      { 32457, 32613, 32768, 0 },
      { 32412, 32649, 32768, 0 } },
    { { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 } } } },
  { { { { 17560, 29888, 32768, 0 },
        { 29671, 31549, 32768, 0 },
        { 31007, 32056, 32768, 0 },
        { 27286, 30006, 32768, 0 } },
    { { 26594, 31212, 32768, 0 },
      { 31208, 32582, 32768, 0 },
      { 31835, 32637, 32768, 0 },
      { 30595, 32206, 32768, 0 } } },

```

```

{ { { 15239, 29932, 32768, 0 },
    { 31315, 32095, 32768, 0 },
    { 32130, 32434, 32768, 0 },
    { 30864, 31996, 32768, 0 } },
  { { 26279, 30968, 32768, 0 },
    { 31142, 32495, 32768, 0 },
    { 31713, 32540, 32768, 0 },
    { 31929, 32594, 32768, 0 } } },
{ { { 2644, 25198, 32768, 0 },
    { 32038, 32451, 32768, 0 },
    { 32639, 32695, 32768, 0 },
    { 32166, 32518, 32768, 0 } },
  { { 17187, 27668, 32768, 0 },
    { 31714, 32550, 32768, 0 },
    { 32283, 32678, 32768, 0 },
    { 31930, 32563, 32768, 0 } } },
{ { { 1044, 2257, 32768, 0 },
    { 30755, 31923, 32768, 0 },
    { 32208, 32693, 32768, 0 },
    { 32244, 32615, 32768, 0 } },
  { { 21317, 26207, 32768, 0 },
    { 29133, 30868, 32768, 0 },
    { 29311, 31231, 32768, 0 },
    { 29657, 31087, 32768, 0 } } },
{ { { 478, 1834, 32768, 0 },
    { 31005, 31987, 32768, 0 },
    { 32317, 32724, 32768, 0 },
    { 30865, 32648, 32768, 0 } },
  { { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 } } } },
{ { { { 20092, 30774, 32768, 0 },
    { 30695, 32020, 32768, 0 },
    { 31131, 32103, 32768, 0 },
    { 28666, 30870, 32768, 0 } },
  { { 27258, 31095, 32768, 0 },
    { 31804, 32623, 32768, 0 },
    { 31763, 32528, 32768, 0 },
    { 31438, 32506, 32768, 0 } } } },
{ { { 18049, 30489, 32768, 0 },
    { 31706, 32286, 32768, 0 },
    { 32163, 32473, 32768, 0 },
    { 31550, 32184, 32768, 0 } },
  { { 27116, 30842, 32768, 0 },
    { 31971, 32598, 32768, 0 },
    { 32088, 32576, 32768, 0 },
    { 32067, 32664, 32768, 0 } } } },
{ { { 12854, 29093, 32768, 0 },
    { 32272, 32558, 32768, 0 },

```



```

    { 32667, 32729, 32768, 0 },
    { 32306, 32585, 32768, 0 } },
  { { 25476, 30366, 32768, 0 },
    { 32169, 32687, 32768, 0 },
    { 32479, 32689, 32768, 0 },
    { 31673, 32634, 32768, 0 } } },
  { { { 2809, 19301, 32768, 0 },
    { 32205, 32622, 32768, 0 },
    { 32338, 32730, 32768, 0 },
    { 31786, 32616, 32768, 0 } },
    { { 22737, 29105, 32768, 0 },
    { 30810, 32362, 32768, 0 },
    { 30014, 32627, 32768, 0 },
    { 30528, 32574, 32768, 0 } } },
  { { { 935, 3382, 32768, 0 },
    { 30789, 31909, 32768, 0 },
    { 32466, 32756, 32768, 0 },
    { 30860, 32513, 32768, 0 } },
    { { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 } } } },
{ { { { 22497, 31198, 32768, 0 },
    { 31715, 32495, 32768, 0 },
    { 31606, 32337, 32768, 0 },
    { 30388, 31990, 32768, 0 } },
    { { 27877, 31584, 32768, 0 },
    { 32170, 32728, 32768, 0 },
    { 32155, 32688, 32768, 0 },
    { 32219, 32702, 32768, 0 } } },
  { { { 21457, 31043, 32768, 0 },
    { 31951, 32483, 32768, 0 },
    { 32153, 32562, 32768, 0 },
    { 31473, 32215, 32768, 0 } },
    { { 27558, 31151, 32768, 0 },
    { 32020, 32640, 32768, 0 },
    { 32097, 32575, 32768, 0 },
    { 32242, 32719, 32768, 0 } } },
  { { { 19980, 30591, 32768, 0 },
    { 32219, 32597, 32768, 0 },
    { 32581, 32706, 32768, 0 },
    { 31803, 32287, 32768, 0 } },
    { { 26473, 30507, 32768, 0 },
    { 32431, 32723, 32768, 0 },
    { 32196, 32611, 32768, 0 },
    { 31588, 32528, 32768, 0 } } },
  { { { 24647, 30463, 32768, 0 },
    { 32412, 32695, 32768, 0 },
    { 32468, 32720, 32768, 0 },
    { 31269, 32523, 32768, 0 } },

```

```

    { { 28482, 31505, 32768, 0 },
      { 32152, 32701, 32768, 0 },
      { 31732, 32598, 32768, 0 },
      { 31767, 32712, 32768, 0 } } },
  { { { 12358, 24977, 32768, 0 },
      { 31331, 32385, 32768, 0 },
      { 32634, 32756, 32768, 0 },
      { 30411, 32548, 32768, 0 } } },
    { { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 } } } }
}

```

```

Default_Coeff_Base_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ SIG_COEF_CONTEXTS ][ 5 ] = {
  { { { { 4034, 8930, 12727, 32768, 0 },
        { 18082, 29741, 31877, 32768, 0 },
        { 12596, 26124, 30493, 32768, 0 },
        { 9446, 21118, 27005, 32768, 0 },
        { 6308, 15141, 21279, 32768, 0 },
        { 2463, 6357, 9783, 32768, 0 },
        { 20667, 30546, 31929, 32768, 0 },
        { 13043, 26123, 30134, 32768, 0 },
        { 8151, 18757, 24778, 32768, 0 },
        { 5255, 12839, 18632, 32768, 0 },
        { 2820, 7206, 11161, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 15736, 27553, 30604, 32768, 0 },
        { 11210, 23794, 28787, 32768, 0 },
        { 5947, 13874, 19701, 32768, 0 },
        { 4215, 9323, 13891, 32768, 0 },
        { 2833, 6462, 10059, 32768, 0 },
        { 19605, 30393, 31582, 32768, 0 },
        { 13523, 26252, 30248, 32768, 0 },
        { 8446, 18622, 24512, 32768, 0 },
        { 3818, 10343, 15974, 32768, 0 },
        { 1481, 4117, 6796, 32768, 0 },
        { 22649, 31302, 32190, 32768, 0 },
        { 14829, 27127, 30449, 32768, 0 },
        { 8313, 17702, 23304, 32768, 0 },
        { 3022, 8301, 12786, 32768, 0 },
        { 1536, 4412, 7184, 32768, 0 },
        { 22354, 29774, 31372, 32768, 0 },
        { 14723, 25472, 29214, 32768, 0 },
        { 6673, 13745, 18662, 32768, 0 },
        { 2068, 5766, 9322, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 } },
    { { { 6302, 16444, 21761, 32768, 0 },
        { 23040, 31538, 32475, 32768, 0 },
        { 15196, 28452, 31496, 32768, 0 },
        { 10020, 22946, 28514, 32768, 0 },
        { 6533, 16862, 23501, 32768, 0 },
        { 3538, 9816, 15076, 32768, 0 },
        { 24444, 31875, 32525, 32768, 0 },

```

```

{ 15881, 28924, 31635, 32768, 0 },
{ 9922, 22873, 28466, 32768, 0 },
{ 6527, 16966, 23691, 32768, 0 },
{ 4114, 11303, 17220, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 20201, 30770, 32209, 32768, 0 },
{ 14754, 28071, 31258, 32768, 0 },
{ 8378, 20186, 26517, 32768, 0 },
{ 5916, 15299, 21978, 32768, 0 },
{ 4268, 11583, 17901, 32768, 0 },
{ 24361, 32025, 32581, 32768, 0 },
{ 18673, 30105, 31943, 32768, 0 },
{ 10196, 22244, 27576, 32768, 0 },
{ 5495, 14349, 20417, 32768, 0 },
{ 2676, 7415, 11498, 32768, 0 },
{ 24678, 31958, 32585, 32768, 0 },
{ 18629, 29906, 31831, 32768, 0 },
{ 9364, 20724, 26315, 32768, 0 },
{ 4641, 12318, 18094, 32768, 0 },
{ 2758, 7387, 11579, 32768, 0 },
{ 25433, 31842, 32469, 32768, 0 },
{ 18795, 29289, 31411, 32768, 0 },
{ 7644, 17584, 23592, 32768, 0 },
{ 3408, 9014, 15047, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 4536, 10072, 14001, 32768, 0 },
{ 25459, 31416, 32206, 32768, 0 },
{ 16605, 28048, 30818, 32768, 0 },
{ 11008, 22857, 27719, 32768, 0 },
{ 6915, 16268, 22315, 32768, 0 },
{ 2625, 6812, 10537, 32768, 0 },
{ 24257, 31788, 32499, 32768, 0 },
{ 16880, 29454, 31879, 32768, 0 },
{ 11958, 25054, 29778, 32768, 0 },
{ 7916, 18718, 25084, 32768, 0 },
{ 3383, 8777, 13446, 32768, 0 },
{ 22720, 31603, 32393, 32768, 0 },
{ 14960, 28125, 31335, 32768, 0 },
{ 9731, 22210, 27928, 32768, 0 },
{ 6304, 15832, 22277, 32768, 0 },

```

```

{ 2910, 7818, 12166, 32768, 0 },
{ 20375, 30627, 32131, 32768, 0 },
{ 13904, 27284, 30887, 32768, 0 },
{ 9368, 21558, 27144, 32768, 0 },
{ 5937, 14966, 21119, 32768, 0 },
{ 2667, 7225, 11319, 32768, 0 },
{ 23970, 31470, 32378, 32768, 0 },
{ 17173, 29734, 32018, 32768, 0 },
{ 12795, 25441, 29965, 32768, 0 },
{ 8981, 19680, 25893, 32768, 0 },
{ 4728, 11372, 16902, 32768, 0 },
{ 24287, 31797, 32439, 32768, 0 },
{ 16703, 29145, 31696, 32768, 0 },
{ 10833, 23554, 28725, 32768, 0 },
{ 6468, 16566, 23057, 32768, 0 },
{ 2415, 6562, 10278, 32768, 0 },
{ 26610, 32395, 32659, 32768, 0 },
{ 18590, 30498, 32117, 32768, 0 },
{ 12420, 25756, 29950, 32768, 0 },
{ 7639, 18746, 24710, 32768, 0 },
{ 3001, 8086, 12347, 32768, 0 },
{ 25076, 32064, 32580, 32768, 0 },
{ 17946, 30128, 32028, 32768, 0 },
{ 12024, 24985, 29378, 32768, 0 },
{ 7517, 18390, 24304, 32768, 0 },
{ 3243, 8781, 13331, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 6037, 16771, 21957, 32768, 0 },
{ 24774, 31704, 32426, 32768, 0 },
{ 16830, 28589, 31056, 32768, 0 },
{ 10602, 22828, 27760, 32768, 0 },
{ 6733, 16829, 23071, 32768, 0 },
{ 3250, 8914, 13556, 32768, 0 },
{ 25582, 32220, 32668, 32768, 0 },
{ 18659, 30342, 32223, 32768, 0 },
{ 12546, 26149, 30515, 32768, 0 },
{ 8420, 20451, 26801, 32768, 0 },
{ 4636, 12420, 18344, 32768, 0 },
{ 27581, 32362, 32639, 32768, 0 },
{ 18987, 30083, 31978, 32768, 0 },
{ 11327, 24248, 29084, 32768, 0 },
{ 7264, 17719, 24120, 32768, 0 },
{ 3995, 10768, 16169, 32768, 0 },
{ 25893, 31831, 32487, 32768, 0 },
{ 16577, 28587, 31379, 32768, 0 },
{ 10189, 22748, 28182, 32768, 0 },
{ 6832, 17094, 23556, 32768, 0 },
{ 3708, 10110, 15334, 32768, 0 },
{ 25904, 32282, 32656, 32768, 0 },
{ 19721, 30792, 32276, 32768, 0 },

```

```

{ 12819, 26243, 30411, 32768, 0 },
{ 8572, 20614, 26891, 32768, 0 },
{ 5364, 14059, 20467, 32768, 0 },
{ 26580, 32438, 32677, 32768, 0 },
{ 20852, 31225, 32340, 32768, 0 },
{ 12435, 25700, 29967, 32768, 0 },
{ 8691, 20825, 26976, 32768, 0 },
{ 4446, 12209, 17269, 32768, 0 },
{ 27350, 32429, 32696, 32768, 0 },
{ 21372, 30977, 32272, 32768, 0 },
{ 12673, 25270, 29853, 32768, 0 },
{ 9208, 20925, 26640, 32768, 0 },
{ 5018, 13351, 18732, 32768, 0 },
{ 27351, 32479, 32713, 32768, 0 },
{ 21398, 31209, 32387, 32768, 0 },
{ 12162, 25047, 29842, 32768, 0 },
{ 7896, 18691, 25319, 32768, 0 },
{ 4670, 12882, 18881, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 5487, 10460, 13708, 32768, 0 },
{ 21597, 28303, 30674, 32768, 0 },
{ 11037, 21953, 26476, 32768, 0 },
{ 8147, 17962, 22952, 32768, 0 },
{ 5242, 13061, 18532, 32768, 0 },
{ 1889, 5208, 8182, 32768, 0 },
{ 26774, 32133, 32590, 32768, 0 },
{ 17844, 29564, 31767, 32768, 0 },
{ 11690, 24438, 29171, 32768, 0 },
{ 7542, 18215, 24459, 32768, 0 },
{ 2993, 8050, 12319, 32768, 0 },
{ 28023, 32328, 32591, 32768, 0 },
{ 18651, 30126, 31954, 32768, 0 },
{ 12164, 25146, 29589, 32768, 0 },
{ 7762, 18530, 24771, 32768, 0 },
{ 3492, 9183, 13920, 32768, 0 },
{ 27591, 32008, 32491, 32768, 0 },
{ 17149, 28853, 31510, 32768, 0 },
{ 11485, 24003, 28860, 32768, 0 },
{ 7697, 18086, 24210, 32768, 0 },
{ 3075, 7999, 12218, 32768, 0 },
{ 28268, 32482, 32654, 32768, 0 },
{ 19631, 31051, 32404, 32768, 0 },
{ 13860, 27260, 31020, 32768, 0 },
{ 9605, 21613, 27594, 32768, 0 },
{ 4876, 12162, 17908, 32768, 0 },
{ 27248, 32316, 32576, 32768, 0 },
{ 18955, 30457, 32075, 32768, 0 },
{ 11824, 23997, 28795, 32768, 0 },
{ 7346, 18196, 24647, 32768, 0 },
{ 3403, 9247, 14111, 32768, 0 },

```

```

{ 29711, 32655, 32735, 32768, 0 },
{ 21169, 31394, 32417, 32768, 0 },
{ 13487, 27198, 30957, 32768, 0 },
{ 8828, 21683, 27614, 32768, 0 },
{ 4270, 11451, 17038, 32768, 0 },
{ 28708, 32578, 32731, 32768, 0 },
{ 20120, 31241, 32482, 32768, 0 },
{ 13692, 27550, 31321, 32768, 0 },
{ 9418, 22514, 28439, 32768, 0 },
{ 4999, 13283, 19462, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 5673, 14302, 19711, 32768, 0 },
{ 26251, 30701, 31834, 32768, 0 },
{ 12782, 23783, 27803, 32768, 0 },
{ 9127, 20657, 25808, 32768, 0 },
{ 6368, 16208, 21462, 32768, 0 },
{ 2465, 7177, 10822, 32768, 0 },
{ 29961, 32563, 32719, 32768, 0 },
{ 18318, 29891, 31949, 32768, 0 },
{ 11361, 24514, 29357, 32768, 0 },
{ 7900, 19603, 25607, 32768, 0 },
{ 4002, 10590, 15546, 32768, 0 },
{ 29637, 32310, 32595, 32768, 0 },
{ 18296, 29913, 31809, 32768, 0 },
{ 10144, 21515, 26871, 32768, 0 },
{ 5358, 14322, 20394, 32768, 0 },
{ 3067, 8362, 13346, 32768, 0 },
{ 28652, 32470, 32676, 32768, 0 },
{ 17538, 30771, 32209, 32768, 0 },
{ 13924, 26882, 30494, 32768, 0 },
{ 10496, 22837, 27869, 32768, 0 },
{ 7236, 16396, 21621, 32768, 0 },
{ 30743, 32687, 32746, 32768, 0 },
{ 23006, 31676, 32489, 32768, 0 },
{ 14494, 27828, 31120, 32768, 0 },
{ 10174, 22801, 28352, 32768, 0 },
{ 6242, 15281, 21043, 32768, 0 },
{ 25817, 32243, 32720, 32768, 0 },
{ 18618, 31367, 32325, 32768, 0 },
{ 13997, 28318, 31878, 32768, 0 },
{ 12255, 26534, 31383, 32768, 0 },
{ 9561, 21588, 28450, 32768, 0 },
{ 28188, 32635, 32724, 32768, 0 },
{ 22060, 32365, 32728, 32768, 0 },
{ 18102, 30690, 32528, 32768, 0 },
{ 14196, 28864, 31999, 32768, 0 },
{ 12262, 25792, 30865, 32768, 0 },
{ 24176, 32109, 32628, 32768, 0 },
{ 18280, 29681, 31963, 32768, 0 },
{ 10205, 23703, 29664, 32768, 0 },

```

[illegible]



{ { { 677, 1500, 2736, 32768, 0 },  
{ 23987, 30702, 32117, 32768, 0 },  
{ 13554, 24571, 29263, 32768, 0 },  
{ 6211, 14556, 21155, 32768, 0 },  
{ 3135, 10972, 15625, 32768, 0 },  
{ 2435, 7127, 11427, 32768, 0 },  
{ 31300, 32532, 32550, 32768, 0 },  
{ 14757, 30365, 31954, 32768, 0 },  
{ 4405, 11612, 18553, 32768, 0 },  
{ 580, 4132, 7322, 32768, 0 },  
{ 1695, 10169, 14124, 32768, 0 },  
{ 30008, 32282, 32591, 32768, 0 },  
{ 19244, 30108, 31748, 32768, 0 },  
{ 11180, 24158, 29555, 32768, 0 },  
{ 5650, 14972, 19209, 32768, 0 },  
{ 2114, 5109, 8456, 32768, 0 },  
{ 31856, 32716, 32748, 32768, 0 },  
{ 23012, 31664, 32572, 32768, 0 },  
{ 13694, 26656, 30636, 32768, 0 },  
{ 8142, 19508, 26093, 32768, 0 },  
{ 4253, 10955, 16724, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 },  
{ 8192, 16384, 24576, 32768, 0 } } } },  
{ { { 601, 983, 1311, 32768, 0 },  
{ 18725, 23406, 28087, 32768, 0 },  
{ 5461, 8192, 10923, 32768, 0 },  
{ 3781, 15124, 21425, 32768, 0 },  
{ 2587, 7761, 12072, 32768, 0 },  
{ 106, 458, 810, 32768, 0 },  
{ 22282, 29710, 31894, 32768, 0 },  
{ 8508, 20926, 25984, 32768, 0 },  
{ 3726, 12713, 18083, 32768, 0 },  
{ 1620, 7112, 10893, 32768, 0 },  
{ 729, 2236, 3495, 32768, 0 },  
{ 30163, 32474, 32684, 32768, 0 },  
{ 18304, 30464, 32000, 32768, 0 },

[illegible]

```

{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } } },
{ { { { 6041, 11854, 15927, 32768, 0 },
{ 20326, 30905, 32251, 32768, 0 },
{ 14164, 26831, 30725, 32768, 0 },
{ 9760, 20647, 26585, 32768, 0 },
{ 6416, 14953, 21219, 32768, 0 },
{ 2966, 7151, 10891, 32768, 0 },
{ 23567, 31374, 32254, 32768, 0 },
{ 14978, 27416, 30946, 32768, 0 },
{ 9434, 20225, 26254, 32768, 0 },
{ 6658, 14558, 20535, 32768, 0 },
{ 3916, 8677, 12989, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 18088, 29545, 31587, 32768, 0 },
{ 13062, 25843, 30073, 32768, 0 },
{ 8940, 16827, 22251, 32768, 0 },
{ 7654, 13220, 17973, 32768, 0 },
{ 5733, 10316, 14456, 32768, 0 },
{ 22879, 31388, 32114, 32768, 0 },
{ 15215, 27993, 30955, 32768, 0 },
{ 9397, 19445, 24978, 32768, 0 },

```

```

{ 3442, 9813, 15344, 32768, 0 },
{ 1368, 3936, 6532, 32768, 0 },
{ 25494, 32033, 32406, 32768, 0 },
{ 16772, 27963, 30718, 32768, 0 },
{ 9419, 18165, 23260, 32768, 0 },
{ 2677, 7501, 11797, 32768, 0 },
{ 1516, 4344, 7170, 32768, 0 },
{ 26556, 31454, 32101, 32768, 0 },
{ 17128, 27035, 30108, 32768, 0 },
{ 8324, 15344, 20249, 32768, 0 },
{ 1903, 5696, 9469, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 8455, 19003, 24368, 32768, 0 },
  { 23563, 32021, 32604, 32768, 0 },
  { 16237, 29446, 31935, 32768, 0 },
  { 10724, 23999, 29358, 32768, 0 },
  { 6725, 17528, 24416, 32768, 0 },
  { 3927, 10927, 16825, 32768, 0 },
  { 26313, 32288, 32634, 32768, 0 },
  { 17430, 30095, 32095, 32768, 0 },
  { 11116, 24606, 29679, 32768, 0 },
  { 7195, 18384, 25269, 32768, 0 },
  { 4726, 12852, 19315, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 8192, 16384, 24576, 32768, 0 },
  { 22822, 31648, 32483, 32768, 0 },
  { 16724, 29633, 31929, 32768, 0 },
  { 10261, 23033, 28725, 32768, 0 },
  { 7029, 17840, 24528, 32768, 0 },
  { 4867, 13886, 21502, 32768, 0 },
  { 25298, 31892, 32491, 32768, 0 },
  { 17809, 29330, 31512, 32768, 0 },
  { 9668, 21329, 26579, 32768, 0 },
  { 4774, 12956, 18976, 32768, 0 },
  { 2322, 7030, 11540, 32768, 0 },
  { 25472, 31920, 32543, 32768, 0 },
  { 17957, 29387, 31632, 32768, 0 },
  { 9196, 20593, 26400, 32768, 0 },
  { 4680, 12705, 19202, 32768, 0 },
  { 2917, 8456, 13436, 32768, 0 },
  { 26471, 32059, 32574, 32768, 0 },

```

```

    { 18458, 29783, 31909, 32768, 0 },
    { 8400, 19464, 25956, 32768, 0 },
    { 3812, 10973, 17206, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } },
{ { { 6779, 13743, 17678, 32768, 0 },
    { 24806, 31797, 32457, 32768, 0 },
    { 17616, 29047, 31372, 32768, 0 },
    { 11063, 23175, 28003, 32768, 0 },
    { 6521, 16110, 22324, 32768, 0 },
    { 2764, 7504, 11654, 32768, 0 },
    { 25266, 32367, 32637, 32768, 0 },
    { 19054, 30553, 32175, 32768, 0 },
    { 12139, 25212, 29807, 32768, 0 },
    { 7311, 18162, 24704, 32768, 0 },
    { 3397, 9164, 14074, 32768, 0 },
    { 25988, 32208, 32522, 32768, 0 },
    { 16253, 28912, 31526, 32768, 0 },
    { 9151, 21387, 27372, 32768, 0 },
    { 5688, 14915, 21496, 32768, 0 },
    { 2717, 7627, 12004, 32768, 0 },
    { 23144, 31855, 32443, 32768, 0 },
    { 16070, 28491, 31325, 32768, 0 },
    { 8702, 20467, 26517, 32768, 0 },
    { 5243, 13956, 20367, 32768, 0 },
    { 2621, 7335, 11567, 32768, 0 },
    { 26636, 32340, 32630, 32768, 0 },
    { 19990, 31050, 32341, 32768, 0 },
    { 13243, 26105, 30315, 32768, 0 },
    { 8588, 19521, 25918, 32768, 0 },
    { 4717, 11585, 17304, 32768, 0 },
    { 25844, 32292, 32582, 32768, 0 },
    { 19090, 30635, 32097, 32768, 0 },
    { 11963, 24546, 28939, 32768, 0 },
    { 6218, 16087, 22354, 32768, 0 },
    { 2340, 6608, 10426, 32768, 0 },
    { 28046, 32576, 32694, 32768, 0 },
    { 21178, 31313, 32296, 32768, 0 },
    { 13486, 26184, 29870, 32768, 0 },
    { 7149, 17871, 23723, 32768, 0 },
    { 2833, 7958, 12259, 32768, 0 },
    { 27710, 32528, 32686, 32768, 0 },
    { 20674, 31076, 32268, 32768, 0 },
    { 12413, 24955, 29243, 32768, 0 },
    { 6676, 16927, 23097, 32768, 0 },
    { 2966, 8333, 12919, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } },
{ { { 8639, 19339, 24429, 32768, 0 },
    { 24404, 31837, 32525, 32768, 0 },
    { 16997, 29425, 31784, 32768, 0 },

```

```

{ 11253, 24234, 29149, 32768, 0 },
{ 6751, 17394, 24028, 32768, 0 },
{ 3490, 9830, 15191, 32768, 0 },
{ 26283, 32471, 32714, 32768, 0 },
{ 19599, 31168, 32442, 32768, 0 },
{ 13146, 26954, 30893, 32768, 0 },
{ 8214, 20588, 26890, 32768, 0 },
{ 4699, 13081, 19300, 32768, 0 },
{ 28212, 32458, 32669, 32768, 0 },
{ 18594, 30316, 32100, 32768, 0 },
{ 11219, 24408, 29234, 32768, 0 },
{ 6865, 17656, 24149, 32768, 0 },
{ 3678, 10362, 16006, 32768, 0 },
{ 25825, 32136, 32616, 32768, 0 },
{ 17313, 29853, 32021, 32768, 0 },
{ 11197, 24471, 29472, 32768, 0 },
{ 6947, 17781, 24405, 32768, 0 },
{ 3768, 10660, 16261, 32768, 0 },
{ 27352, 32500, 32706, 32768, 0 },
{ 20850, 31468, 32469, 32768, 0 },
{ 14021, 27707, 31133, 32768, 0 },
{ 8964, 21748, 27838, 32768, 0 },
{ 5437, 14665, 21187, 32768, 0 },
{ 26304, 32492, 32698, 32768, 0 },
{ 20409, 31380, 32385, 32768, 0 },
{ 13682, 27222, 30632, 32768, 0 },
{ 8974, 21236, 26685, 32768, 0 },
{ 4234, 11665, 16934, 32768, 0 },
{ 26273, 32357, 32711, 32768, 0 },
{ 20672, 31242, 32441, 32768, 0 },
{ 14172, 27254, 30902, 32768, 0 },
{ 9870, 21898, 27275, 32768, 0 },
{ 5164, 13506, 19270, 32768, 0 },
{ 26725, 32459, 32728, 32768, 0 },
{ 20991, 31442, 32527, 32768, 0 },
{ 13071, 26434, 30811, 32768, 0 },
{ 8184, 20090, 26742, 32768, 0 },
{ 4803, 13255, 19895, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 7555, 14942, 18501, 32768, 0 },
{ 24410, 31178, 32287, 32768, 0 },
{ 14394, 26738, 30253, 32768, 0 },
{ 8413, 19554, 25195, 32768, 0 },
{ 4766, 12924, 18785, 32768, 0 },
{ 2029, 5806, 9207, 32768, 0 },
{ 26776, 32364, 32663, 32768, 0 },
{ 18732, 29967, 31931, 32768, 0 },
{ 11005, 23786, 28852, 32768, 0 },
{ 6466, 16909, 23510, 32768, 0 },
{ 3044, 8638, 13419, 32768, 0 },

```

```

{ 29208, 32582, 32704, 32768, 0 },
{ 20068, 30857, 32208, 32768, 0 },
{ 12003, 25085, 29595, 32768, 0 },
{ 6947, 17750, 24189, 32768, 0 },
{ 3245, 9103, 14007, 32768, 0 },
{ 27359, 32465, 32669, 32768, 0 },
{ 19421, 30614, 32174, 32768, 0 },
{ 11915, 25010, 29579, 32768, 0 },
{ 6950, 17676, 24074, 32768, 0 },
{ 3007, 8473, 13096, 32768, 0 },
{ 29002, 32676, 32735, 32768, 0 },
{ 22102, 31849, 32576, 32768, 0 },
{ 14408, 28009, 31405, 32768, 0 },
{ 9027, 21679, 27931, 32768, 0 },
{ 4694, 12678, 18748, 32768, 0 },
{ 28216, 32528, 32682, 32768, 0 },
{ 20849, 31264, 32318, 32768, 0 },
{ 12756, 25815, 29751, 32768, 0 },
{ 7565, 18801, 24923, 32768, 0 },
{ 3509, 9533, 14477, 32768, 0 },
{ 30133, 32687, 32739, 32768, 0 },
{ 23063, 31910, 32515, 32768, 0 },
{ 14588, 28051, 31132, 32768, 0 },
{ 9085, 21649, 27457, 32768, 0 },
{ 4261, 11654, 17264, 32768, 0 },
{ 29518, 32691, 32748, 32768, 0 },
{ 22451, 31959, 32613, 32768, 0 },
{ 14864, 28722, 31700, 32768, 0 },
{ 9695, 22964, 28716, 32768, 0 },
{ 4932, 13358, 19502, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 6465, 16958, 21688, 32768, 0 },
{ 25199, 31514, 32360, 32768, 0 },
{ 14774, 27149, 30607, 32768, 0 },
{ 9257, 21438, 26972, 32768, 0 },
{ 5723, 15183, 21882, 32768, 0 },
{ 3150, 8879, 13731, 32768, 0 },
{ 26989, 32262, 32682, 32768, 0 },
{ 17396, 29937, 32085, 32768, 0 },
{ 11387, 24901, 29784, 32768, 0 },
{ 7289, 18821, 25548, 32768, 0 },
{ 3734, 10577, 16086, 32768, 0 },
{ 29728, 32501, 32695, 32768, 0 },
{ 17431, 29701, 31903, 32768, 0 },
{ 9921, 22826, 28300, 32768, 0 },
{ 5896, 15434, 22068, 32768, 0 },
{ 3430, 9646, 14757, 32768, 0 },
{ 28614, 32511, 32705, 32768, 0 },
{ 19364, 30638, 32263, 32768, 0 },
{ 13129, 26254, 30402, 32768, 0 },

```

```

{ 8754, 20484, 26440, 32768, 0 },
{ 4378, 11607, 17110, 32768, 0 },
{ 30292, 32671, 32744, 32768, 0 },
{ 21780, 31603, 32501, 32768, 0 },
{ 14314, 27829, 31291, 32768, 0 },
{ 9611, 22327, 28263, 32768, 0 },
{ 4890, 13087, 19065, 32768, 0 },
{ 25862, 32567, 32733, 32768, 0 },
{ 20794, 32050, 32567, 32768, 0 },
{ 17243, 30625, 32254, 32768, 0 },
{ 13283, 27628, 31474, 32768, 0 },
{ 9669, 22532, 28918, 32768, 0 },
{ 27435, 32697, 32748, 32768, 0 },
{ 24922, 32390, 32714, 32768, 0 },
{ 21449, 31504, 32536, 32768, 0 },
{ 16392, 29729, 31832, 32768, 0 },
{ 11692, 24884, 29076, 32768, 0 },
{ 24193, 32290, 32735, 32768, 0 },
{ 18909, 31104, 32563, 32768, 0 },
{ 12236, 26841, 31403, 32768, 0 },
{ 8171, 21840, 29082, 32768, 0 },
{ 7224, 17280, 25275, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 3078, 6839, 9890, 32768, 0 },
{ 13837, 20450, 24479, 32768, 0 },
{ 5914, 14222, 19328, 32768, 0 },
{ 3866, 10267, 14762, 32768, 0 },
{ 2612, 7208, 11042, 32768, 0 },
{ 1067, 2991, 4776, 32768, 0 },
{ 25817, 31646, 32529, 32768, 0 },
{ 13708, 26338, 30385, 32768, 0 },
{ 7328, 18585, 24870, 32768, 0 },
{ 4691, 13080, 19276, 32768, 0 },
{ 1825, 5253, 8352, 32768, 0 },
{ 29386, 32315, 32624, 32768, 0 },
{ 17160, 29001, 31360, 32768, 0 },
{ 9602, 21862, 27396, 32768, 0 },
{ 5915, 15772, 22148, 32768, 0 },
{ 2786, 7779, 12047, 32768, 0 },
{ 29246, 32450, 32663, 32768, 0 },
{ 18696, 29929, 31818, 32768, 0 },
{ 10510, 23369, 28560, 32768, 0 },
{ 6229, 16499, 23125, 32768, 0 },
{ 2608, 7448, 11705, 32768, 0 },
{ 30753, 32710, 32748, 32768, 0 },
{ 21638, 31487, 32503, 32768, 0 },
{ 12937, 26854, 30870, 32768, 0 },
{ 8182, 20596, 26970, 32768, 0 },
{ 3637, 10269, 15497, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },

```



[illegible]

[illegible]

[illegible]

```

{ 7881, 15930, 22096, 32768, 0 },
{ 5388, 10960, 15918, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 20745, 30773, 32093, 32768, 0 },
{ 15200, 27221, 30861, 32768, 0 },
{ 13032, 20873, 25667, 32768, 0 },
{ 12285, 18663, 23494, 32768, 0 },
{ 11563, 17481, 21489, 32768, 0 },
{ 26260, 31982, 32320, 32768, 0 },
{ 15397, 28083, 31100, 32768, 0 },
{ 9742, 19217, 24824, 32768, 0 },
{ 3261, 9629, 15362, 32768, 0 },
{ 1480, 4322, 7499, 32768, 0 },
{ 27599, 32256, 32460, 32768, 0 },
{ 16857, 27659, 30774, 32768, 0 },
{ 9551, 18290, 23748, 32768, 0 },
{ 3052, 8933, 14103, 32768, 0 },
{ 2021, 5910, 9787, 32768, 0 },
{ 29005, 32015, 32392, 32768, 0 },
{ 17677, 27694, 30863, 32768, 0 },
{ 9204, 17356, 23219, 32768, 0 },
{ 2403, 7516, 12814, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 10808, 22056, 26896, 32768, 0 },
{ 25739, 32313, 32676, 32768, 0 },
{ 17288, 30203, 32221, 32768, 0 },
{ 11359, 24878, 29896, 32768, 0 },
{ 6949, 17767, 24893, 32768, 0 },
{ 4287, 11796, 18071, 32768, 0 },
{ 27880, 32521, 32705, 32768, 0 },
{ 19038, 31004, 32414, 32768, 0 },
{ 12564, 26345, 30768, 32768, 0 },
{ 8269, 19947, 26779, 32768, 0 },
{ 5674, 14657, 21674, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },

```

```

{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 25742, 32319, 32671, 32768, 0 },
{ 19557, 31164, 32454, 32768, 0 },
{ 13381, 26381, 30755, 32768, 0 },
{ 10101, 21466, 26722, 32768, 0 },
{ 9209, 19650, 26825, 32768, 0 },
{ 27107, 31917, 32432, 32768, 0 },
{ 18056, 28893, 31203, 32768, 0 },
{ 10200, 21434, 26764, 32768, 0 },
{ 4660, 12913, 19502, 32768, 0 },
{ 2368, 6930, 12504, 32768, 0 },
{ 26960, 32158, 32613, 32768, 0 },
{ 18628, 30005, 32031, 32768, 0 },
{ 10233, 22442, 28232, 32768, 0 },
{ 5471, 14630, 21516, 32768, 0 },
{ 3235, 10767, 17109, 32768, 0 },
{ 27696, 32440, 32692, 32768, 0 },
{ 20032, 31167, 32438, 32768, 0 },
{ 8700, 21341, 28442, 32768, 0 },
{ 5662, 14831, 21795, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9704, 17294, 21132, 32768, 0 },
{ 26762, 32278, 32633, 32768, 0 },
{ 18382, 29620, 31819, 32768, 0 },
{ 10891, 23475, 28723, 32768, 0 },
{ 6358, 16583, 23309, 32768, 0 },
{ 3248, 9118, 14141, 32768, 0 },
{ 27204, 32573, 32699, 32768, 0 },
{ 19818, 30824, 32329, 32768, 0 },
{ 11772, 25120, 30041, 32768, 0 },
{ 6995, 18033, 25039, 32768, 0 },
{ 3752, 10442, 16098, 32768, 0 },
{ 27222, 32256, 32559, 32768, 0 },
{ 15356, 28399, 31475, 32768, 0 },
{ 8821, 20635, 27057, 32768, 0 },
{ 5511, 14404, 21239, 32768, 0 },
{ 2935, 8222, 13051, 32768, 0 },
{ 24875, 32120, 32529, 32768, 0 },
{ 15233, 28265, 31445, 32768, 0 },
{ 8605, 20570, 26932, 32768, 0 },
{ 5431, 14413, 21196, 32768, 0 },
{ 2994, 8341, 13223, 32768, 0 },
{ 28201, 32604, 32700, 32768, 0 },
{ 21041, 31446, 32456, 32768, 0 },
{ 13221, 26213, 30475, 32768, 0 },
{ 8255, 19385, 26037, 32768, 0 },

```

```

{ 4930, 12585, 18830, 32768, 0 },
{ 28768, 32448, 32627, 32768, 0 },
{ 19705, 30561, 32021, 32768, 0 },
{ 11572, 23589, 28220, 32768, 0 },
{ 5532, 15034, 21446, 32768, 0 },
{ 2460, 7150, 11456, 32768, 0 },
{ 29874, 32619, 32699, 32768, 0 },
{ 21621, 31071, 32201, 32768, 0 },
{ 12511, 24747, 28992, 32768, 0 },
{ 6281, 16395, 22748, 32768, 0 },
{ 3246, 9278, 14497, 32768, 0 },
{ 29715, 32625, 32712, 32768, 0 },
{ 20958, 31011, 32283, 32768, 0 },
{ 11233, 23671, 28806, 32768, 0 },
{ 6012, 16128, 22868, 32768, 0 },
{ 3427, 9851, 15414, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 11016, 22111, 26794, 32768, 0 },
  { 25946, 32357, 32677, 32768, 0 },
  { 17890, 30452, 32252, 32768, 0 },
  { 11678, 25142, 29816, 32768, 0 },
  { 6720, 17534, 24584, 32768, 0 },
  { 4230, 11665, 17820, 32768, 0 },
  { 28400, 32623, 32747, 32768, 0 },
  { 21164, 31668, 32575, 32768, 0 },
  { 13572, 27388, 31182, 32768, 0 },
  { 8234, 20750, 27358, 32768, 0 },
  { 5065, 14055, 20897, 32768, 0 },
  { 28981, 32547, 32705, 32768, 0 },
  { 18681, 30543, 32239, 32768, 0 },
  { 10919, 24075, 29286, 32768, 0 },
  { 6431, 17199, 24077, 32768, 0 },
  { 3819, 10464, 16618, 32768, 0 },
  { 26870, 32467, 32693, 32768, 0 },
  { 19041, 30831, 32347, 32768, 0 },
  { 11794, 25211, 30016, 32768, 0 },
  { 6888, 18019, 24970, 32768, 0 },
  { 4370, 12363, 18992, 32768, 0 },
  { 29578, 32670, 32744, 32768, 0 },
  { 23159, 32007, 32613, 32768, 0 },
  { 15315, 28669, 31676, 32768, 0 },
  { 9298, 22607, 28782, 32768, 0 },
  { 6144, 15913, 22968, 32768, 0 },
  { 28110, 32499, 32669, 32768, 0 },
  { 21574, 30937, 32015, 32768, 0 },
  { 12759, 24818, 28727, 32768, 0 },
  { 6545, 16761, 23042, 32768, 0 },
  { 3649, 10597, 16833, 32768, 0 },
  { 28163, 32552, 32728, 32768, 0 },
  { 22101, 31469, 32464, 32768, 0 },

```

```

{ 13160, 25472, 30143, 32768, 0 },
{ 7303, 18684, 25468, 32768, 0 },
{ 5241, 13975, 20955, 32768, 0 },
{ 28400, 32631, 32744, 32768, 0 },
{ 22104, 31793, 32603, 32768, 0 },
{ 13557, 26571, 30846, 32768, 0 },
{ 7749, 19861, 26675, 32768, 0 },
{ 4873, 14030, 21234, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9800, 17635, 21073, 32768, 0 },
{ 26153, 31885, 32527, 32768, 0 },
{ 15038, 27852, 31006, 32768, 0 },
{ 8718, 20564, 26486, 32768, 0 },
{ 5128, 14076, 20514, 32768, 0 },
{ 2636, 7566, 11925, 32768, 0 },
{ 27551, 32504, 32701, 32768, 0 },
{ 18310, 30054, 32100, 32768, 0 },
{ 10211, 23420, 29082, 32768, 0 },
{ 6222, 16876, 23916, 32768, 0 },
{ 3462, 9954, 15498, 32768, 0 },
{ 29991, 32633, 32721, 32768, 0 },
{ 19883, 30751, 32201, 32768, 0 },
{ 11141, 24184, 29285, 32768, 0 },
{ 6420, 16940, 23774, 32768, 0 },
{ 3392, 9753, 15118, 32768, 0 },
{ 28465, 32616, 32712, 32768, 0 },
{ 19850, 30702, 32244, 32768, 0 },
{ 10983, 24024, 29223, 32768, 0 },
{ 6294, 16770, 23582, 32768, 0 },
{ 3244, 9283, 14509, 32768, 0 },
{ 30023, 32717, 32748, 32768, 0 },
{ 22940, 32032, 32626, 32768, 0 },
{ 14282, 27928, 31473, 32768, 0 },
{ 8562, 21327, 27914, 32768, 0 },
{ 4846, 13393, 19919, 32768, 0 },
{ 29981, 32590, 32695, 32768, 0 },
{ 20465, 30963, 32166, 32768, 0 },
{ 11479, 23579, 28195, 32768, 0 },
{ 5916, 15648, 22073, 32768, 0 },
{ 3031, 8605, 13398, 32768, 0 },
{ 31146, 32691, 32739, 32768, 0 },
{ 23106, 31724, 32444, 32768, 0 },
{ 13783, 26738, 30439, 32768, 0 },
{ 7852, 19468, 25807, 32768, 0 },
{ 3860, 11124, 16853, 32768, 0 },
{ 31014, 32724, 32748, 32768, 0 },
{ 23629, 32109, 32628, 32768, 0 },
{ 14747, 28115, 31403, 32768, 0 },
{ 8545, 21242, 27478, 32768, 0 },
{ 4574, 12781, 19067, 32768, 0 },

```

```

    { 8192, 16384, 24576, 32768, 0 } },
  { { 9185, 19694, 24688, 32768, 0 },
    { 26081, 31985, 32621, 32768, 0 },
    { 16015, 29000, 31787, 32768, 0 },
    { 10542, 23690, 29206, 32768, 0 },
    { 6732, 17945, 24677, 32768, 0 },
    { 3916, 11039, 16722, 32768, 0 },
    { 28224, 32566, 32744, 32768, 0 },
    { 19100, 31138, 32485, 32768, 0 },
    { 12528, 26620, 30879, 32768, 0 },
    { 7741, 20277, 26885, 32768, 0 },
    { 4566, 12845, 18990, 32768, 0 },
    { 29933, 32593, 32718, 32768, 0 },
    { 17670, 30333, 32155, 32768, 0 },
    { 10385, 23600, 28909, 32768, 0 },
    { 6243, 16236, 22407, 32768, 0 },
    { 3976, 10389, 16017, 32768, 0 },
    { 28377, 32561, 32738, 32768, 0 },
    { 19366, 31175, 32482, 32768, 0 },
    { 13327, 27175, 31094, 32768, 0 },
    { 8258, 20769, 27143, 32768, 0 },
    { 4703, 13198, 19527, 32768, 0 },
    { 31086, 32706, 32748, 32768, 0 },
    { 22853, 31902, 32583, 32768, 0 },
    { 14759, 28186, 31419, 32768, 0 },
    { 9284, 22382, 28348, 32768, 0 },
    { 5585, 15192, 21868, 32768, 0 },
    { 28291, 32652, 32746, 32768, 0 },
    { 19849, 32107, 32571, 32768, 0 },
    { 14834, 26818, 29214, 32768, 0 },
    { 10306, 22594, 28672, 32768, 0 },
    { 6615, 17384, 23384, 32768, 0 },
    { 28947, 32604, 32745, 32768, 0 },
    { 25625, 32289, 32646, 32768, 0 },
    { 18758, 28672, 31403, 32768, 0 },
    { 10017, 23430, 28523, 32768, 0 },
    { 6862, 15269, 22131, 32768, 0 },
    { 23933, 32509, 32739, 32768, 0 },
    { 19927, 31495, 32631, 32768, 0 },
    { 11903, 26023, 30621, 32768, 0 },
    { 7026, 20094, 27252, 32768, 0 },
    { 5998, 18106, 24437, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } },
  { { { 4456, 11274, 15533, 32768, 0 },
    { 21219, 29079, 31616, 32768, 0 },
    { 11173, 23774, 28567, 32768, 0 },
    { 7282, 18293, 24263, 32768, 0 },
    { 4890, 13286, 19115, 32768, 0 },
    { 1890, 5508, 8659, 32768, 0 },
    { 26651, 32136, 32647, 32768, 0 },

```



[illegible]

[illegible]

[illegible]

```

    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } } },
{ { { { 7062, 16472, 22319, 32768, 0 },
        { 24538, 32261, 32674, 32768, 0 },
        { 13675, 28041, 31779, 32768, 0 },
        { 8590, 20674, 27631, 32768, 0 },
        { 5685, 14675, 22013, 32768, 0 },
        { 3655, 9898, 15731, 32768, 0 },
        { 26493, 32418, 32658, 32768, 0 },
        { 16376, 29342, 32090, 32768, 0 },
        { 10594, 22649, 28970, 32768, 0 },
        { 8176, 17170, 24303, 32768, 0 },
        { 5605, 12694, 19139, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 23888, 31902, 32542, 32768, 0 },
        { 18612, 29687, 31987, 32768, 0 },
        { 16245, 24852, 29249, 32768, 0 },
        { 15765, 22608, 27559, 32768, 0 },
        { 19895, 24699, 27510, 32768, 0 },
        { 28401, 32212, 32457, 32768, 0 },
        { 15274, 27825, 30980, 32768, 0 },
        { 9364, 18128, 24332, 32768, 0 },
        { 2283, 8193, 15082, 32768, 0 },
        { 1228, 3972, 7881, 32768, 0 },
        { 29455, 32469, 32620, 32768, 0 },
        { 17981, 28245, 31388, 32768, 0 },
        { 10921, 20098, 26240, 32768, 0 },
        { 3743, 11829, 18657, 32768, 0 },
        { 2374, 9593, 15715, 32768, 0 },
        { 31068, 32466, 32635, 32768, 0 },
        { 20321, 29572, 31971, 32768, 0 },
        { 10771, 20255, 27119, 32768, 0 },

```

```

    { 2795, 10410, 17361, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } },
  { { 9320, 22102, 27840, 32768, 0 },
    { 27057, 32464, 32724, 32768, 0 },
    { 16331, 30268, 32309, 32768, 0 },
    { 10319, 23935, 29720, 32768, 0 },
    { 6189, 16448, 24106, 32768, 0 },
    { 3589, 10884, 18808, 32768, 0 },
    { 29026, 32624, 32748, 32768, 0 },
    { 19226, 31507, 32587, 32768, 0 },
    { 12692, 26921, 31203, 32768, 0 },
    { 7049, 19532, 27635, 32768, 0 },
    { 7727, 15669, 23252, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 28056, 32625, 32748, 32768, 0 },
    { 22383, 32075, 32669, 32768, 0 },
    { 15417, 27098, 31749, 32768, 0 },
    { 18127, 26493, 27190, 32768, 0 },
    { 5461, 16384, 21845, 32768, 0 },
    { 27982, 32091, 32584, 32768, 0 },
    { 19045, 29868, 31972, 32768, 0 },
    { 10397, 22266, 27932, 32768, 0 },
    { 5990, 13697, 21500, 32768, 0 },
    { 1792, 6912, 15104, 32768, 0 },
    { 28198, 32501, 32718, 32768, 0 },
    { 21534, 31521, 32569, 32768, 0 },
    { 11109, 25217, 30017, 32768, 0 },
    { 5671, 15124, 26151, 32768, 0 },
    { 4681, 14043, 18725, 32768, 0 },
    { 28688, 32580, 32741, 32768, 0 },
    { 22576, 32079, 32661, 32768, 0 },
    { 10627, 22141, 28340, 32768, 0 },
    { 9362, 14043, 28087, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } },
  { { { 7754, 16948, 22142, 32768, 0 },
    { 25670, 32330, 32691, 32768, 0 },
    { 15663, 29225, 31994, 32768, 0 },
    { 9878, 23288, 29158, 32768, 0 },
    { 6419, 17088, 24336, 32768, 0 },

```

```

{ 3859, 11003, 17039, 32768, 0 },
{ 27562, 32595, 32725, 32768, 0 },
{ 17575, 30588, 32399, 32768, 0 },
{ 10819, 24838, 30309, 32768, 0 },
{ 7124, 18686, 25916, 32768, 0 },
{ 4479, 12688, 19340, 32768, 0 },
{ 28385, 32476, 32673, 32768, 0 },
{ 15306, 29005, 31938, 32768, 0 },
{ 8937, 21615, 28322, 32768, 0 },
{ 5982, 15603, 22786, 32768, 0 },
{ 3620, 10267, 16136, 32768, 0 },
{ 27280, 32464, 32667, 32768, 0 },
{ 15607, 29160, 32004, 32768, 0 },
{ 9091, 22135, 28740, 32768, 0 },
{ 6232, 16632, 24020, 32768, 0 },
{ 4047, 11377, 17672, 32768, 0 },
{ 29220, 32630, 32718, 32768, 0 },
{ 19650, 31220, 32462, 32768, 0 },
{ 13050, 26312, 30827, 32768, 0 },
{ 9228, 20870, 27468, 32768, 0 },
{ 6146, 15149, 21971, 32768, 0 },
{ 30169, 32481, 32623, 32768, 0 },
{ 17212, 29311, 31554, 32768, 0 },
{ 9911, 21311, 26882, 32768, 0 },
{ 4487, 13314, 20372, 32768, 0 },
{ 2570, 7772, 12889, 32768, 0 },
{ 30924, 32613, 32708, 32768, 0 },
{ 19490, 30206, 32107, 32768, 0 },
{ 11232, 23998, 29276, 32768, 0 },
{ 6769, 17955, 25035, 32768, 0 },
{ 4398, 12623, 19214, 32768, 0 },
{ 30609, 32627, 32722, 32768, 0 },
{ 19370, 30582, 32287, 32768, 0 },
{ 10457, 23619, 29409, 32768, 0 },
{ 6443, 17637, 24834, 32768, 0 },
{ 4645, 13236, 20106, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 8626, 20271, 26216, 32768, 0 },
{ 26707, 32406, 32711, 32768, 0 },
{ 16999, 30329, 32286, 32768, 0 },
{ 11445, 25123, 30286, 32768, 0 },
{ 6411, 18828, 25601, 32768, 0 },
{ 6801, 12458, 20248, 32768, 0 },
{ 29918, 32682, 32748, 32768, 0 },
{ 20649, 31739, 32618, 32768, 0 },
{ 12879, 27773, 31581, 32768, 0 },
{ 7896, 21751, 28244, 32768, 0 },
{ 5260, 14870, 23698, 32768, 0 },
{ 29252, 32593, 32731, 32768, 0 },
{ 17072, 30460, 32294, 32768, 0 },

```

```

{ 10653, 24143, 29365, 32768, 0 },
{ 6536, 17490, 23983, 32768, 0 },
{ 4929, 13170, 20085, 32768, 0 },
{ 28137, 32518, 32715, 32768, 0 },
{ 18171, 30784, 32407, 32768, 0 },
{ 11437, 25436, 30459, 32768, 0 },
{ 7252, 18534, 26176, 32768, 0 },
{ 4126, 13353, 20978, 32768, 0 },
{ 31162, 32726, 32748, 32768, 0 },
{ 23017, 32222, 32701, 32768, 0 },
{ 15629, 29233, 32046, 32768, 0 },
{ 9387, 22621, 29480, 32768, 0 },
{ 6922, 17616, 25010, 32768, 0 },
{ 28838, 32265, 32614, 32768, 0 },
{ 19701, 30206, 31920, 32768, 0 },
{ 11214, 22410, 27933, 32768, 0 },
{ 5320, 14177, 23034, 32768, 0 },
{ 5049, 12881, 17827, 32768, 0 },
{ 27484, 32471, 32734, 32768, 0 },
{ 21076, 31526, 32561, 32768, 0 },
{ 12707, 26303, 31211, 32768, 0 },
{ 8169, 21722, 28219, 32768, 0 },
{ 6045, 19406, 27042, 32768, 0 },
{ 27753, 32572, 32745, 32768, 0 },
{ 20832, 31878, 32653, 32768, 0 },
{ 13250, 27356, 31674, 32768, 0 },
{ 7718, 21508, 29858, 32768, 0 },
{ 7209, 18350, 25559, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 7876, 16901, 21741, 32768, 0 },
{ 24001, 31898, 32625, 32768, 0 },
{ 14529, 27959, 31451, 32768, 0 },
{ 8273, 20818, 27258, 32768, 0 },
{ 5278, 14673, 21510, 32768, 0 },
{ 2983, 8843, 14039, 32768, 0 },
{ 28016, 32574, 32732, 32768, 0 },
{ 17471, 30306, 32301, 32768, 0 },
{ 10224, 24063, 29728, 32768, 0 },
{ 6602, 17954, 25052, 32768, 0 },
{ 4002, 11585, 17759, 32768, 0 },
{ 30190, 32634, 32739, 32768, 0 },
{ 17497, 30282, 32270, 32768, 0 },
{ 10229, 23729, 29538, 32768, 0 },
{ 6344, 17211, 24440, 32768, 0 },
{ 3849, 11189, 17108, 32768, 0 },
{ 28570, 32583, 32726, 32768, 0 },
{ 17521, 30161, 32238, 32768, 0 },
{ 10153, 23565, 29378, 32768, 0 },
{ 6455, 17341, 24443, 32768, 0 },
{ 3907, 11042, 17024, 32768, 0 },

```

```

{ 30689, 32715, 32748, 32768, 0 },
{ 21546, 31840, 32610, 32768, 0 },
{ 13547, 27581, 31459, 32768, 0 },
{ 8912, 21757, 28309, 32768, 0 },
{ 5548, 15080, 22046, 32768, 0 },
{ 30783, 32540, 32685, 32768, 0 },
{ 17540, 29528, 31668, 32768, 0 },
{ 10160, 21468, 26783, 32768, 0 },
{ 4724, 13393, 20054, 32768, 0 },
{ 2702, 8174, 13102, 32768, 0 },
{ 31648, 32686, 32742, 32768, 0 },
{ 20954, 31094, 32337, 32768, 0 },
{ 12420, 25698, 30179, 32768, 0 },
{ 7304, 19320, 26248, 32768, 0 },
{ 4366, 12261, 18864, 32768, 0 },
{ 31581, 32723, 32748, 32768, 0 },
{ 21373, 31586, 32525, 32768, 0 },
{ 12744, 26625, 30885, 32768, 0 },
{ 7431, 20322, 26950, 32768, 0 },
{ 4692, 13323, 20111, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 7833, 18369, 24095, 32768, 0 },
{ 26650, 32273, 32702, 32768, 0 },
{ 16371, 29961, 32191, 32768, 0 },
{ 11055, 24082, 29629, 32768, 0 },
{ 6892, 18644, 25400, 32768, 0 },
{ 5006, 13057, 19240, 32768, 0 },
{ 29834, 32666, 32748, 32768, 0 },
{ 19577, 31335, 32570, 32768, 0 },
{ 12253, 26509, 31122, 32768, 0 },
{ 7991, 20772, 27711, 32768, 0 },
{ 5677, 15910, 23059, 32768, 0 },
{ 30109, 32532, 32720, 32768, 0 },
{ 16747, 30166, 32252, 32768, 0 },
{ 10134, 23542, 29184, 32768, 0 },
{ 5791, 16176, 23556, 32768, 0 },
{ 4362, 10414, 17284, 32768, 0 },
{ 29492, 32626, 32748, 32768, 0 },
{ 19894, 31402, 32525, 32768, 0 },
{ 12942, 27071, 30869, 32768, 0 },
{ 8346, 21216, 27405, 32768, 0 },
{ 6572, 17087, 23859, 32768, 0 },
{ 32035, 32735, 32748, 32768, 0 },
{ 22957, 31838, 32618, 32768, 0 },
{ 14724, 28572, 31772, 32768, 0 },
{ 10364, 23999, 29553, 32768, 0 },
{ 7004, 18433, 25655, 32768, 0 },
{ 27528, 32277, 32681, 32768, 0 },
{ 16959, 31171, 32096, 32768, 0 },
{ 10486, 23593, 27962, 32768, 0 },

```



[illegible]

[illegible]

[illegible]

[illegible]

```

Default_Coeff_Br_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ LEVEL_CONTEXTS ][ BR_CDF_SIZE +
1 ] = {
  { { { { 14298, 20718, 24174, 32768, 0 },
        { 12536, 19601, 23789, 32768, 0 },
        { 8712, 15051, 19503, 32768, 0 },
        { 6170, 11327, 15434, 32768, 0 },
        { 4742, 8926, 12538, 32768, 0 },
        { 3803, 7317, 10546, 32768, 0 },
        { 1696, 3317, 4871, 32768, 0 },
        { 14392, 19951, 22756, 32768, 0 },
        { 15978, 23218, 26818, 32768, 0 },
        { 12187, 19474, 23889, 32768, 0 },
        { 9176, 15640, 20259, 32768, 0 },
        { 7068, 12655, 17028, 32768, 0 },
        { 5656, 10442, 14472, 32768, 0 },
        { 2580, 4992, 7244, 32768, 0 },
        { 12136, 18049, 21426, 32768, 0 },
        { 13784, 20721, 24481, 32768, 0 },
        { 10836, 17621, 21900, 32768, 0 },
        { 8372, 14444, 18847, 32768, 0 },
        { 6523, 11779, 16000, 32768, 0 },
        { 5337, 9898, 13760, 32768, 0 },
        { 3034, 5860, 8462, 32768, 0 } },
    { { { 15967, 22905, 26286, 32768, 0 },
        { 13534, 20654, 24579, 32768, 0 },
        { 9504, 16092, 20535, 32768, 0 },
        { 6975, 12568, 16903, 32768, 0 },
        { 5364, 10091, 14020, 32768, 0 },
        { 4357, 8370, 11857, 32768, 0 },
        { 2506, 4934, 7218, 32768, 0 },
        { 23032, 28815, 30936, 32768, 0 },
        { 19540, 26704, 29719, 32768, 0 },
        { 15158, 22969, 27097, 32768, 0 },
        { 11408, 18865, 23650, 32768, 0 },
        { 8885, 15448, 20250, 32768, 0 },
        { 7108, 12853, 17416, 32768, 0 },
        { 4231, 8041, 11480, 32768, 0 },
        { 19823, 26490, 29156, 32768, 0 },
        { 18890, 25929, 28932, 32768, 0 },
        { 15660, 23491, 27433, 32768, 0 },
        { 12147, 19776, 24488, 32768, 0 },
        { 9728, 16774, 21649, 32768, 0 },
        { 7919, 14277, 19066, 32768, 0 },
        { 5440, 10170, 14185, 32768, 0 } } },
    { { { 14406, 20862, 24414, 32768, 0 },
        { 11824, 18907, 23109, 32768, 0 },
        { 8257, 14393, 18803, 32768, 0 },
        { 5860, 10747, 14778, 32768, 0 },
        { 4475, 8486, 11984, 32768, 0 },
        { 3606, 6954, 10043, 32768, 0 },

```

```

{ 1736, 3410, 5048, 32768, 0 },
{ 14430, 20046, 22882, 32768, 0 },
{ 15593, 22899, 26709, 32768, 0 },
{ 12102, 19368, 23811, 32768, 0 },
{ 9059, 15584, 20262, 32768, 0 },
{ 6999, 12603, 17048, 32768, 0 },
{ 5684, 10497, 14553, 32768, 0 },
{ 2822, 5438, 7862, 32768, 0 },
{ 15785, 21585, 24359, 32768, 0 },
{ 18347, 25229, 28266, 32768, 0 },
{ 14974, 22487, 26389, 32768, 0 },
{ 11423, 18681, 23271, 32768, 0 },
{ 8863, 15350, 20008, 32768, 0 },
{ 7153, 12852, 17278, 32768, 0 },
{ 3707, 7036, 9982, 32768, 0 } },
{ { 15460, 21696, 25469, 32768, 0 },
{ 12170, 19249, 23191, 32768, 0 },
{ 8723, 15027, 19332, 32768, 0 },
{ 6428, 11704, 15874, 32768, 0 },
{ 4922, 9292, 13052, 32768, 0 },
{ 4139, 7695, 11010, 32768, 0 },
{ 2291, 4508, 6598, 32768, 0 },
{ 19856, 26920, 29828, 32768, 0 },
{ 17923, 25289, 28792, 32768, 0 },
{ 14278, 21968, 26297, 32768, 0 },
{ 10910, 18136, 22950, 32768, 0 },
{ 8423, 14815, 19627, 32768, 0 },
{ 6771, 12283, 16774, 32768, 0 },
{ 4074, 7750, 11081, 32768, 0 },
{ 19852, 26074, 28672, 32768, 0 },
{ 19371, 26110, 28989, 32768, 0 },
{ 16265, 23873, 27663, 32768, 0 },
{ 12758, 20378, 24952, 32768, 0 },
{ 10095, 17098, 21961, 32768, 0 },
{ 8250, 14628, 19451, 32768, 0 },
{ 5205, 9745, 13622, 32768, 0 } } },
{ { { 10563, 16233, 19763, 32768, 0 },
{ 9794, 16022, 19804, 32768, 0 },
{ 6750, 11945, 15759, 32768, 0 },
{ 4963, 9186, 12752, 32768, 0 },
{ 3845, 7435, 10627, 32768, 0 },
{ 3051, 6085, 8834, 32768, 0 },
{ 1311, 2596, 3830, 32768, 0 },
{ 11246, 16404, 19689, 32768, 0 },
{ 12315, 18911, 22731, 32768, 0 },
{ 10557, 17095, 21289, 32768, 0 },
{ 8136, 14006, 18249, 32768, 0 },
{ 6348, 11474, 15565, 32768, 0 },
{ 5196, 9655, 13400, 32768, 0 },
{ 2349, 4526, 6587, 32768, 0 },

```

```

    { 13337, 18730, 21569, 32768, 0 },
    { 19306, 26071, 28882, 32768, 0 },
    { 15952, 23540, 27254, 32768, 0 },
    { 12409, 19934, 24430, 32768, 0 },
    { 9760, 16706, 21389, 32768, 0 },
    { 8004, 14220, 18818, 32768, 0 },
    { 4138, 7794, 10961, 32768, 0 } },
  { { 10870, 16684, 20949, 32768, 0 },
    { 9664, 15230, 18680, 32768, 0 },
    { 6886, 12109, 15408, 32768, 0 },
    { 4825, 8900, 12305, 32768, 0 },
    { 3630, 7162, 10314, 32768, 0 },
    { 3036, 6429, 9387, 32768, 0 },
    { 1671, 3296, 4940, 32768, 0 },
    { 13819, 19159, 23026, 32768, 0 },
    { 11984, 19108, 23120, 32768, 0 },
    { 10690, 17210, 21663, 32768, 0 },
    { 7984, 14154, 18333, 32768, 0 },
    { 6868, 12294, 16124, 32768, 0 },
    { 5274, 8994, 12868, 32768, 0 },
    { 2988, 5771, 8424, 32768, 0 },
    { 19736, 26647, 29141, 32768, 0 },
    { 18933, 26070, 28984, 32768, 0 },
    { 15779, 23048, 27200, 32768, 0 },
    { 12638, 20061, 24532, 32768, 0 },
    { 10692, 17545, 22220, 32768, 0 },
    { 9217, 15251, 20054, 32768, 0 },
    { 5078, 9284, 12594, 32768, 0 } } },
  { { { 2331, 3662, 5244, 32768, 0 },
    { 2891, 4771, 6145, 32768, 0 },
    { 4598, 7623, 9729, 32768, 0 },
    { 3520, 6845, 9199, 32768, 0 },
    { 3417, 6119, 9324, 32768, 0 },
    { 2601, 5412, 7385, 32768, 0 },
    { 600, 1173, 1744, 32768, 0 },
    { 7672, 13286, 17469, 32768, 0 },
    { 4232, 7792, 10793, 32768, 0 },
    { 2915, 5317, 7397, 32768, 0 },
    { 2318, 4356, 6152, 32768, 0 },
    { 2127, 4000, 5554, 32768, 0 },
    { 1850, 3478, 5275, 32768, 0 },
    { 977, 1933, 2843, 32768, 0 },
    { 18280, 24387, 27989, 32768, 0 },
    { 15852, 22671, 26185, 32768, 0 },
    { 13845, 20951, 24789, 32768, 0 },
    { 11055, 17966, 22129, 32768, 0 },
    { 9138, 15422, 19801, 32768, 0 },
    { 7454, 13145, 17456, 32768, 0 },
    { 3370, 6393, 9013, 32768, 0 } },
    { { 5842, 9229, 10838, 32768, 0 },

```

[illegible]



```

    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } } },
{ { { { 14995, 21341, 24749, 32768, 0 },
        { 13158, 20289, 24601, 32768, 0 },
        { 8941, 15326, 19876, 32768, 0 },
        { 6297, 11541, 15807, 32768, 0 },
        { 4817, 9029, 12776, 32768, 0 },
        { 3731, 7273, 10627, 32768, 0 },
        { 1847, 3617, 5354, 32768, 0 },
        { 14472, 19659, 22343, 32768, 0 },
        { 16806, 24162, 27533, 32768, 0 },
        { 12900, 20404, 24713, 32768, 0 },
        { 9411, 16112, 20797, 32768, 0 },
        { 7056, 12697, 17148, 32768, 0 },
        { 5544, 10339, 14460, 32768, 0 },
        { 2954, 5704, 8319, 32768, 0 },
        { 12464, 18071, 21354, 32768, 0 },
        { 15482, 22528, 26034, 32768, 0 },
        { 12070, 19269, 23624, 32768, 0 },
        { 8953, 15406, 20106, 32768, 0 },
        { 7027, 12730, 17220, 32768, 0 },
        { 5887, 10913, 15140, 32768, 0 },
        { 3793, 7278, 10447, 32768, 0 } } },
{ { { 15571, 22232, 25749, 32768, 0 },
        { 14506, 21575, 25374, 32768, 0 },
        { 10189, 17089, 21569, 32768, 0 },
        { 7316, 13301, 17915, 32768, 0 },
        { 5783, 10912, 15190, 32768, 0 },
        { 4760, 9155, 13088, 32768, 0 },
        { 2993, 5966, 8774, 32768, 0 },
        { 23424, 28903, 30778, 32768, 0 },
        { 20775, 27666, 30290, 32768, 0 },
        { 16474, 24410, 28299, 32768, 0 },
        { 12471, 20180, 24987, 32768, 0 },
        { 9410, 16487, 21439, 32768, 0 },
        { 7536, 13614, 18529, 32768, 0 },
        { 5048, 9586, 13549, 32768, 0 },
        { 21090, 27290, 29756, 32768, 0 },
        { 20796, 27402, 30026, 32768, 0 },
        { 17819, 25485, 28969, 32768, 0 },

```

```

    { 13860, 21909, 26462, 32768, 0 },
    { 11002, 18494, 23529, 32768, 0 },
    { 8953, 15929, 20897, 32768, 0 },
    { 6448, 11918, 16454, 32768, 0 } } },
{ { { 15999, 22208, 25449, 32768, 0 },
    { 13050, 19988, 24122, 32768, 0 },
    { 8594, 14864, 19378, 32768, 0 },
    { 6033, 11079, 15238, 32768, 0 },
    { 4554, 8683, 12347, 32768, 0 },
    { 3672, 7139, 10337, 32768, 0 },
    { 1900, 3771, 5576, 32768, 0 },
    { 15788, 21340, 23949, 32768, 0 },
    { 16825, 24235, 27758, 32768, 0 },
    { 12873, 20402, 24810, 32768, 0 },
    { 9590, 16363, 21094, 32768, 0 },
    { 7352, 13209, 17733, 32768, 0 },
    { 5960, 10989, 15184, 32768, 0 },
    { 3232, 6234, 9007, 32768, 0 },
    { 15761, 20716, 23224, 32768, 0 },
    { 19318, 25989, 28759, 32768, 0 },
    { 15529, 23094, 26929, 32768, 0 },
    { 11662, 18989, 23641, 32768, 0 },
    { 8955, 15568, 20366, 32768, 0 },
    { 7281, 13106, 17708, 32768, 0 },
    { 4248, 8059, 11440, 32768, 0 } } },
{ { { 14899, 21217, 24503, 32768, 0 },
    { 13519, 20283, 24047, 32768, 0 },
    { 9429, 15966, 20365, 32768, 0 },
    { 6700, 12355, 16652, 32768, 0 },
    { 5088, 9704, 13716, 32768, 0 },
    { 4243, 8154, 11731, 32768, 0 },
    { 2702, 5364, 7861, 32768, 0 },
    { 22745, 28388, 30454, 32768, 0 },
    { 20235, 27146, 29922, 32768, 0 },
    { 15896, 23715, 27637, 32768, 0 },
    { 11840, 19350, 24131, 32768, 0 },
    { 9122, 15932, 20880, 32768, 0 },
    { 7488, 13581, 18362, 32768, 0 },
    { 5114, 9568, 13370, 32768, 0 },
    { 20845, 26553, 28932, 32768, 0 },
    { 20981, 27372, 29884, 32768, 0 },
    { 17781, 25335, 28785, 32768, 0 },
    { 13760, 21708, 26297, 32768, 0 },
    { 10975, 18415, 23365, 32768, 0 },
    { 9045, 15789, 20686, 32768, 0 },
    { 6130, 11199, 15423, 32768, 0 } } } },
{ { { 13549, 19724, 23158, 32768, 0 },
    { 11844, 18382, 22246, 32768, 0 },
    { 7919, 13619, 17773, 32768, 0 },
    { 5486, 10143, 13946, 32768, 0 },

```

```

{ 4166, 7983, 11324, 32768, 0 },
{ 3364, 6506, 9427, 32768, 0 },
{ 1598, 3160, 4674, 32768, 0 },
{ 15281, 20979, 23781, 32768, 0 },
{ 14939, 22119, 25952, 32768, 0 },
{ 11363, 18407, 22812, 32768, 0 },
{ 8609, 14857, 19370, 32768, 0 },
{ 6737, 12184, 16480, 32768, 0 },
{ 5506, 10263, 14262, 32768, 0 },
{ 2990, 5786, 8380, 32768, 0 },
{ 20249, 25253, 27417, 32768, 0 },
{ 21070, 27518, 30001, 32768, 0 },
{ 16854, 24469, 28074, 32768, 0 },
{ 12864, 20486, 25000, 32768, 0 },
{ 9962, 16978, 21778, 32768, 0 },
{ 8074, 14338, 19048, 32768, 0 },
{ 4494, 8479, 11906, 32768, 0 } },
{ { 13960, 19617, 22829, 32768, 0 },
  { 11150, 17341, 21228, 32768, 0 },
  { 7150, 12964, 17190, 32768, 0 },
  { 5331, 10002, 13867, 32768, 0 },
  { 4167, 7744, 11057, 32768, 0 },
  { 3480, 6629, 9646, 32768, 0 },
  { 1883, 3784, 5686, 32768, 0 },
  { 18752, 25660, 28912, 32768, 0 },
  { 16968, 24586, 28030, 32768, 0 },
  { 13520, 21055, 25313, 32768, 0 },
  { 10453, 17626, 22280, 32768, 0 },
  { 8386, 14505, 19116, 32768, 0 },
  { 6742, 12595, 17008, 32768, 0 },
  { 4273, 8140, 11499, 32768, 0 },
  { 22120, 27827, 30233, 32768, 0 },
  { 20563, 27358, 29895, 32768, 0 },
  { 17076, 24644, 28153, 32768, 0 },
  { 13362, 20942, 25309, 32768, 0 },
  { 10794, 17965, 22695, 32768, 0 },
  { 9014, 15652, 20319, 32768, 0 },
  { 5708, 10512, 14497, 32768, 0 } } },
{ { { 5705, 10930, 15725, 32768, 0 },
    { 7946, 12765, 16115, 32768, 0 },
    { 6801, 12123, 16226, 32768, 0 },
    { 5462, 10135, 14200, 32768, 0 },
    { 4189, 8011, 11507, 32768, 0 },
    { 3191, 6229, 9408, 32768, 0 },
    { 1057, 2137, 3212, 32768, 0 },
    { 10018, 17067, 21491, 32768, 0 },
    { 7380, 12582, 16453, 32768, 0 },
    { 6068, 10845, 14339, 32768, 0 },
    { 5098, 9198, 12555, 32768, 0 },
    { 4312, 8010, 11119, 32768, 0 },

```

[illegible]

```

    { 8192, 16384, 24576, 32768, 0 } },
  { { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } } },
  { { { { 16138, 22223, 25509, 32768, 0 },
    { 15347, 22430, 26332, 32768, 0 },
    { 9614, 16736, 21332, 32768, 0 },
    { 6600, 12275, 16907, 32768, 0 },
    { 4811, 9424, 13547, 32768, 0 },
    { 3748, 7809, 11420, 32768, 0 },
    { 2254, 4587, 6890, 32768, 0 },
    { 15196, 20284, 23177, 32768, 0 },
    { 18317, 25469, 28451, 32768, 0 },
    { 13918, 21651, 25842, 32768, 0 },
    { 10052, 17150, 21995, 32768, 0 },
    { 7499, 13630, 18587, 32768, 0 },
    { 6158, 11417, 16003, 32768, 0 },
    { 4014, 7785, 11252, 32768, 0 },
    { 15048, 21067, 24384, 32768, 0 },
    { 18202, 25346, 28553, 32768, 0 },
    { 14302, 22019, 26356, 32768, 0 },
    { 10839, 18139, 23166, 32768, 0 },
    { 8715, 15744, 20806, 32768, 0 },
    { 7536, 13576, 18544, 32768, 0 },
    { 5413, 10335, 14498, 32768, 0 } } },
    { { 17394, 24501, 27895, 32768, 0 },
      { 15889, 23420, 27185, 32768, 0 },
      { 11561, 19133, 23870, 32768, 0 },
      { 8285, 14812, 19844, 32768, 0 },
      { 6496, 12043, 16550, 32768, 0 },
      { 4771, 9574, 13677, 32768, 0 },
      { 3603, 6830, 10144, 32768, 0 },

```

```

{ 21656, 27704, 30200, 32768, 0 },
{ 21324, 27915, 30511, 32768, 0 },
{ 17327, 25336, 28997, 32768, 0 },
{ 13417, 21381, 26033, 32768, 0 },
{ 10132, 17425, 22338, 32768, 0 },
{ 8580, 15016, 19633, 32768, 0 },
{ 5694, 11477, 16411, 32768, 0 },
{ 24116, 29780, 31450, 32768, 0 },
{ 23853, 29695, 31591, 32768, 0 },
{ 20085, 27614, 30428, 32768, 0 },
{ 15326, 24335, 28575, 32768, 0 },
{ 11814, 19472, 24810, 32768, 0 },
{ 10221, 18611, 24767, 32768, 0 },
{ 7689, 14558, 20321, 32768, 0 } } },
{ { { 16214, 22380, 25770, 32768, 0 },
{ 14213, 21304, 25295, 32768, 0 },
{ 9213, 15823, 20455, 32768, 0 },
{ 6395, 11758, 16139, 32768, 0 },
{ 4779, 9187, 13066, 32768, 0 },
{ 3821, 7501, 10953, 32768, 0 },
{ 2293, 4567, 6795, 32768, 0 },
{ 15859, 21283, 23820, 32768, 0 },
{ 18404, 25602, 28726, 32768, 0 },
{ 14325, 21980, 26206, 32768, 0 },
{ 10669, 17937, 22720, 32768, 0 },
{ 8297, 14642, 19447, 32768, 0 },
{ 6746, 12389, 16893, 32768, 0 },
{ 4324, 8251, 11770, 32768, 0 },
{ 16532, 21631, 24475, 32768, 0 },
{ 20667, 27150, 29668, 32768, 0 },
{ 16728, 24510, 28175, 32768, 0 },
{ 12861, 20645, 25332, 32768, 0 },
{ 10076, 17361, 22417, 32768, 0 },
{ 8395, 14940, 19963, 32768, 0 },
{ 5731, 10683, 14912, 32768, 0 } },
{ { 14433, 21155, 24938, 32768, 0 },
{ 14658, 21716, 25545, 32768, 0 },
{ 9923, 16824, 21557, 32768, 0 },
{ 6982, 13052, 17721, 32768, 0 },
{ 5419, 10503, 15050, 32768, 0 },
{ 4852, 9162, 13014, 32768, 0 },
{ 3271, 6395, 9630, 32768, 0 },
{ 22210, 27833, 30109, 32768, 0 },
{ 20750, 27368, 29821, 32768, 0 },
{ 16894, 24828, 28573, 32768, 0 },
{ 13247, 21276, 25757, 32768, 0 },
{ 10038, 17265, 22563, 32768, 0 },
{ 8587, 14947, 20327, 32768, 0 },
{ 5645, 11371, 15252, 32768, 0 },
{ 22027, 27526, 29714, 32768, 0 },

```

```

    { 23098, 29146, 31221, 32768, 0 },
    { 19886, 27341, 30272, 32768, 0 },
    { 15609, 23747, 28046, 32768, 0 },
    { 11993, 20065, 24939, 32768, 0 },
    { 9637, 18267, 23671, 32768, 0 },
    { 7625, 13801, 19144, 32768, 0 } } },
{ { { 14438, 20798, 24089, 32768, 0 },
    { 12621, 19203, 23097, 32768, 0 },
    { 8177, 14125, 18402, 32768, 0 },
    { 5674, 10501, 14456, 32768, 0 },
    { 4236, 8239, 11733, 32768, 0 },
    { 3447, 6750, 9806, 32768, 0 },
    { 1986, 3950, 5864, 32768, 0 },
    { 16208, 22099, 24930, 32768, 0 },
    { 16537, 24025, 27585, 32768, 0 },
    { 12780, 20381, 24867, 32768, 0 },
    { 9767, 16612, 21416, 32768, 0 },
    { 7686, 13738, 18398, 32768, 0 },
    { 6333, 11614, 15964, 32768, 0 },
    { 3941, 7571, 10836, 32768, 0 },
    { 22819, 27422, 29202, 32768, 0 },
    { 22224, 28514, 30721, 32768, 0 },
    { 17660, 25433, 28913, 32768, 0 },
    { 13574, 21482, 26002, 32768, 0 },
    { 10629, 17977, 22938, 32768, 0 },
    { 8612, 15298, 20265, 32768, 0 },
    { 5607, 10491, 14596, 32768, 0 } } },
{ { { 13569, 19800, 23206, 32768, 0 },
    { 13128, 19924, 23869, 32768, 0 },
    { 8329, 14841, 19403, 32768, 0 },
    { 6130, 10976, 15057, 32768, 0 },
    { 4682, 8839, 12518, 32768, 0 },
    { 3656, 7409, 10588, 32768, 0 },
    { 2577, 5099, 7412, 32768, 0 },
    { 22427, 28684, 30585, 32768, 0 },
    { 20913, 27750, 30139, 32768, 0 },
    { 15840, 24109, 27834, 32768, 0 },
    { 12308, 20029, 24569, 32768, 0 },
    { 10216, 16785, 21458, 32768, 0 },
    { 8309, 14203, 19113, 32768, 0 },
    { 6043, 11168, 15307, 32768, 0 },
    { 23166, 28901, 30998, 32768, 0 },
    { 21899, 28405, 30751, 32768, 0 },
    { 18413, 26091, 29443, 32768, 0 },
    { 15233, 23114, 27352, 32768, 0 },
    { 12683, 20472, 25288, 32768, 0 },
    { 10702, 18259, 23409, 32768, 0 },
    { 8125, 14464, 19226, 32768, 0 } } } },
{ { { 9040, 14786, 18360, 32768, 0 },
    { 9979, 15718, 19415, 32768, 0 },

```

[illegible]



```

{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } } },
{ { { { 18315, 24289, 27551, 32768, 0 },
{ 16854, 24068, 27835, 32768, 0 },
{ 10140, 17927, 23173, 32768, 0 },
{ 6722, 12982, 18267, 32768, 0 },
{ 4661, 9826, 14706, 32768, 0 },
{ 3832, 8165, 12294, 32768, 0 },
{ 2795, 6098, 9245, 32768, 0 },
{ 17145, 23326, 26672, 32768, 0 },
{ 20733, 27680, 30308, 32768, 0 },
{ 16032, 24461, 28546, 32768, 0 },
{ 11653, 20093, 25081, 32768, 0 },
{ 9290, 16429, 22086, 32768, 0 },
{ 7796, 14598, 19982, 32768, 0 },
{ 6502, 12378, 17441, 32768, 0 },
{ 21681, 27732, 30320, 32768, 0 },
{ 22389, 29044, 31261, 32768, 0 },
{ 19027, 26731, 30087, 32768, 0 },
{ 14739, 23755, 28624, 32768, 0 },

```

```

    { 11358, 20778, 25511, 32768, 0 },
    { 10995, 18073, 24190, 32768, 0 },
    { 9162, 14990, 20617, 32768, 0 } },
  { { 21425, 27952, 30388, 32768, 0 },
    { 18062, 25838, 29034, 32768, 0 },
    { 11956, 19881, 24808, 32768, 0 },
    { 7718, 15000, 20980, 32768, 0 },
    { 5702, 11254, 16143, 32768, 0 },
    { 4898, 9088, 16864, 32768, 0 },
    { 3679, 6776, 11907, 32768, 0 },
    { 23294, 30160, 31663, 32768, 0 },
    { 24397, 29896, 31836, 32768, 0 },
    { 19245, 27128, 30593, 32768, 0 },
    { 13202, 19825, 26404, 32768, 0 },
    { 11578, 19297, 23957, 32768, 0 },
    { 8073, 13297, 21370, 32768, 0 },
    { 5461, 10923, 19745, 32768, 0 },
    { 27367, 30521, 31934, 32768, 0 },
    { 24904, 30671, 31940, 32768, 0 },
    { 23075, 28460, 31299, 32768, 0 },
    { 14400, 23658, 30417, 32768, 0 },
    { 13885, 23882, 28325, 32768, 0 },
    { 14746, 22938, 27853, 32768, 0 },
    { 5461, 16384, 27307, 32768, 0 } } },
  { { { 18274, 24813, 27890, 32768, 0 },
    { 15537, 23149, 27003, 32768, 0 },
    { 9449, 16740, 21827, 32768, 0 },
    { 6700, 12498, 17261, 32768, 0 },
    { 4988, 9866, 14198, 32768, 0 },
    { 4236, 8147, 11902, 32768, 0 },
    { 2867, 5860, 8654, 32768, 0 },
    { 17124, 23171, 26101, 32768, 0 },
    { 20396, 27477, 30148, 32768, 0 },
    { 16573, 24629, 28492, 32768, 0 },
    { 12749, 20846, 25674, 32768, 0 },
    { 10233, 17878, 22818, 32768, 0 },
    { 8525, 15332, 20363, 32768, 0 },
    { 6283, 11632, 16255, 32768, 0 },
    { 20466, 26511, 29286, 32768, 0 },
    { 23059, 29174, 31191, 32768, 0 },
    { 19481, 27263, 30241, 32768, 0 },
    { 15458, 23631, 28137, 32768, 0 },
    { 12416, 20608, 25693, 32768, 0 },
    { 10261, 18011, 23261, 32768, 0 },
    { 8016, 14655, 19666, 32768, 0 } } },
  { { { 17616, 24586, 28112, 32768, 0 },
    { 15809, 23299, 27155, 32768, 0 },
    { 10767, 18890, 23793, 32768, 0 },
    { 7727, 14255, 18865, 32768, 0 },
    { 6129, 11926, 16882, 32768, 0 },

```

```

{ 4482, 9704, 14861, 32768, 0 },
{ 3277, 7452, 11522, 32768, 0 },
{ 22956, 28551, 30730, 32768, 0 },
{ 22724, 28937, 30961, 32768, 0 },
{ 18467, 26324, 29580, 32768, 0 },
{ 13234, 20713, 25649, 32768, 0 },
{ 11181, 17592, 22481, 32768, 0 },
{ 8291, 18358, 24576, 32768, 0 },
{ 7568, 11881, 14984, 32768, 0 },
{ 24948, 29001, 31147, 32768, 0 },
{ 25674, 30619, 32151, 32768, 0 },
{ 20841, 26793, 29603, 32768, 0 },
{ 14669, 24356, 28666, 32768, 0 },
{ 11334, 23593, 28219, 32768, 0 },
{ 8922, 14762, 22873, 32768, 0 },
{ 8301, 13544, 20535, 32768, 0 } } },
{ { { 17113, 23733, 27081, 32768, 0 },
{ 14139, 21406, 25452, 32768, 0 },
{ 8552, 15002, 19776, 32768, 0 },
{ 5871, 11120, 15378, 32768, 0 },
{ 4455, 8616, 12253, 32768, 0 },
{ 3469, 6910, 10386, 32768, 0 },
{ 2255, 4553, 6782, 32768, 0 },
{ 18224, 24376, 27053, 32768, 0 },
{ 19290, 26710, 29614, 32768, 0 },
{ 14936, 22991, 27184, 32768, 0 },
{ 11238, 18951, 23762, 32768, 0 },
{ 8786, 15617, 20588, 32768, 0 },
{ 7317, 13228, 18003, 32768, 0 },
{ 5101, 9512, 13493, 32768, 0 },
{ 22639, 28222, 30210, 32768, 0 },
{ 23216, 29331, 31307, 32768, 0 },
{ 19075, 26762, 29895, 32768, 0 },
{ 15014, 23113, 27457, 32768, 0 },
{ 11938, 19857, 24752, 32768, 0 },
{ 9942, 17280, 22282, 32768, 0 },
{ 7167, 13144, 17752, 32768, 0 } } },
{ { { 15820, 22738, 26488, 32768, 0 },
{ 13530, 20885, 25216, 32768, 0 },
{ 8395, 15530, 20452, 32768, 0 },
{ 6574, 12321, 16380, 32768, 0 },
{ 5353, 10419, 14568, 32768, 0 },
{ 4613, 8446, 12381, 32768, 0 },
{ 3440, 7158, 9903, 32768, 0 },
{ 24247, 29051, 31224, 32768, 0 },
{ 22118, 28058, 30369, 32768, 0 },
{ 16498, 24768, 28389, 32768, 0 },
{ 12920, 21175, 26137, 32768, 0 },
{ 10730, 18619, 25352, 32768, 0 },
{ 10187, 16279, 22791, 32768, 0 },

```

```

{ 9310, 14631, 22127, 32768, 0 },
{ 24970, 30558, 32057, 32768, 0 },
{ 24801, 29942, 31698, 32768, 0 },
{ 22432, 28453, 30855, 32768, 0 },
{ 19054, 25680, 29580, 32768, 0 },
{ 14392, 23036, 28109, 32768, 0 },
{ 12495, 20947, 26650, 32768, 0 },
{ 12442, 20326, 26214, 32768, 0 } } },
{ { { 12162, 18785, 22648, 32768, 0 },
{ 12749, 19697, 23806, 32768, 0 },
{ 8580, 15297, 20346, 32768, 0 },
{ 6169, 11749, 16543, 32768, 0 },
{ 4836, 9391, 13448, 32768, 0 },
{ 3821, 7711, 11613, 32768, 0 },
{ 2228, 4601, 7070, 32768, 0 },
{ 16319, 24725, 28280, 32768, 0 },
{ 15698, 23277, 27168, 32768, 0 },
{ 12726, 20368, 25047, 32768, 0 },
{ 9912, 17015, 21976, 32768, 0 },
{ 7888, 14220, 19179, 32768, 0 },
{ 6777, 12284, 17018, 32768, 0 },
{ 4492, 8590, 12252, 32768, 0 },
{ 23249, 28904, 30947, 32768, 0 },
{ 21050, 27908, 30512, 32768, 0 },
{ 17440, 25340, 28949, 32768, 0 },
{ 14059, 22018, 26541, 32768, 0 },
{ 11288, 18903, 23898, 32768, 0 },
{ 9411, 16342, 21428, 32768, 0 },
{ 6278, 11588, 15944, 32768, 0 } } },
{ { { 13981, 20067, 23226, 32768, 0 },
{ 16922, 23580, 26783, 32768, 0 },
{ 11005, 19039, 24487, 32768, 0 },
{ 7389, 14218, 19798, 32768, 0 },
{ 5598, 11505, 17206, 32768, 0 },
{ 6090, 11213, 15659, 32768, 0 },
{ 3820, 7371, 10119, 32768, 0 },
{ 21082, 26925, 29675, 32768, 0 },
{ 21262, 28627, 31128, 32768, 0 },
{ 18392, 26454, 30437, 32768, 0 },
{ 14870, 22910, 27096, 32768, 0 },
{ 12620, 19484, 24908, 32768, 0 },
{ 9290, 16553, 22802, 32768, 0 },
{ 6668, 14288, 20004, 32768, 0 },
{ 27704, 31055, 31949, 32768, 0 },
{ 24709, 29978, 31788, 32768, 0 },
{ 21668, 29264, 31657, 32768, 0 },
{ 18295, 26968, 30074, 32768, 0 },
{ 16399, 24422, 29313, 32768, 0 },
{ 14347, 23026, 28104, 32768, 0 },
{ 12370, 19806, 24477, 32768, 0 } } } },

```

[illegible]

## 9.5. Quantizer matrix tables

### 9.5.1. General

There are three fundamental quantizer matrix sizes: 32x32, 32x16 and 16x32. One set of these three sizes of matrix is defined for each quantizer level and for each plane type (luma or chroma). All other quantizer matrix sizes are subsampled from these. All matrix sizes, including derived ones, are defined in the table `Quantizer_Matrix` in [section 9.5.3](#). For informative purposes the subsampling process is defined in [section 9.5.2](#).

### 9.5.2. Derivation process (Informative)

**Note:** This subsection is provided for information only regarding the derivation of `Quantizer_Matrix`, and is not required to correctly decode AV1 bitstreams (and therefore not invoked by this specification). All required tables are defined in [section 9.5.3](#).

The input to this process is a transform size `txSz`.

The output is an array `derivedMatrix` of size `Tx_Width[ txSz ] * Tx_Height[ txSz ]`, containing the derived matrix.

- The array `fundamentalMatrix` is derived as follows.
  - If `Tx_Width[ txSz ]` is equal to `Tx_Height[ txSz ]`, `fundamentalMatrix` is set equal to the 32x32 fundamental matrix for the quantizer level and plane type in question.
  - If `Tx_Width[ txSz ]` is greater than `Tx_Height[ txSz ]`, `fundamentalMatrix` is set equal to the 32x16 fundamental matrix for the quantizer level and plane type in question.
  - Otherwise (`Tx_Width[ txSz ]` is less than `Tx_Height[ txSz ]`), `fundamentalMatrix` is set equal to the 16x32 fundamental matrix for the quantizer level and plane type in question.
- The variable `fW` is set equal to the width of `fundamentalMatrix`.
- The variable `fH` is set equal to the height of `fundamentalMatrix`.
- The variable `ratioW` is set equal to  $fW / Tx\_Width[ txSz ]$ .
- The variable `ratioH` is set equal to  $fH / Tx\_Height[ txSz ]$ .
- The variable `phaseW` is set equal to  $(ratioW + 1) / 2 - 1$ .
- The variable `phaseH` is set equal to  $(ratioH + 1) / 2 - 1$ .
- The array element `derivedMatrix[ i * Tx_Width[ txSz ] + j ]` is set equal to `fundamentalMatrix[ (ratioH * i + phaseH) * fW + ratioW * j + phaseW ]` for  $i=0..Tx\_Height[ txSz ] - 1$ , for  $j=0..Tx\_Width[ txSz ] - 1$ .

## 9.5.3. Tables

```
Qm_Offset[ TX_SIZES_ALL ] = { 0, 16, 80, 336, 336, 1360, 1392, 1424, 1552, 1680, 2192,  
                               336, 336, 2704, 2768, 2832, 3088, 1680, 2192 }
```

```

Quantizer_Matrix[ 16 ][ 2 ][ QM_TOTAL_SIZE ] = {
{
  /* Luma */
  /* Size 4x4 */
  32, 43, 73, 97, 43, 67, 94, 110, 73, 94, 137, 150, 97, 110, 150, 200,
  /* Size 8x8 */
  32, 32, 38, 51, 68, 84, 95, 109, 32, 35, 40, 49, 63, 76, 89, 102, 38,
  40, 54, 65, 78, 91, 98, 106, 51, 49, 65, 82, 97, 111, 113, 121, 68, 63,
  78, 97, 117, 134, 138, 142, 84, 76, 91, 111, 134, 152, 159, 168, 95, 89,
  98, 113, 138, 159, 183, 199, 109, 102, 106, 121, 142, 168, 199, 220,
  /* Size 16x16 */
  32, 31, 31, 34, 36, 44, 48, 59, 65, 80, 83, 91, 97, 104, 111, 119, 31,
  32, 32, 33, 34, 41, 44, 54, 59, 72, 75, 83, 90, 97, 104, 112, 31, 32,
  33, 35, 36, 42, 45, 54, 59, 71, 74, 81, 86, 93, 100, 107, 34, 33, 35,
  39, 42, 47, 51, 58, 63, 74, 76, 81, 84, 90, 97, 105, 36, 34, 36, 42, 48,
  54, 57, 64, 68, 79, 81, 88, 91, 96, 102, 105, 44, 41, 42, 47, 54, 63,
  67, 75, 79, 90, 92, 95, 100, 102, 109, 112, 48, 44, 45, 51, 57, 67, 71,
  80, 85, 96, 99, 107, 108, 111, 117, 120, 59, 54, 54, 58, 64, 75, 80, 92,
  98, 110, 113, 115, 116, 122, 125, 130, 65, 59, 59, 63, 68, 79, 85, 98,
  105, 118, 121, 127, 130, 134, 135, 140, 80, 72, 71, 74, 79, 90, 96, 110,
  118, 134, 137, 140, 143, 144, 146, 152, 83, 75, 74, 76, 81, 92, 99, 113,
  121, 137, 140, 151, 152, 155, 158, 165, 91, 83, 81, 81, 88, 95, 107,
  115, 127, 140, 151, 159, 166, 169, 173, 179, 97, 90, 86, 84, 91, 100,
  108, 116, 130, 143, 152, 166, 174, 182, 189, 193, 104, 97, 93, 90, 96,
  102, 111, 122, 134, 144, 155, 169, 182, 191, 200, 210, 111, 104, 100,
  97, 102, 109, 117, 125, 135, 146, 158, 173, 189, 200, 210, 220, 119,
  112, 107, 105, 105, 112, 120, 130, 140, 152, 165, 179, 193, 210, 220,
  231,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 32, 34, 35, 36, 39, 44, 46, 48, 54, 59, 62, 65, 71,
  80, 81, 83, 88, 91, 94, 97, 101, 104, 107, 111, 115, 119, 123, 31, 32,
  32, 32, 32, 32, 34, 34, 35, 38, 42, 44, 46, 51, 56, 59, 62, 68, 76, 77,
  78, 84, 86, 89, 92, 95, 99, 102, 105, 109, 113, 116, 31, 32, 32, 32, 32,
  32, 33, 34, 34, 37, 41, 42, 44, 49, 54, 56, 59, 65, 72, 73, 75, 80, 83,
  86, 90, 93, 97, 101, 104, 108, 112, 116, 31, 32, 32, 32, 33, 33, 34, 35,
  35, 38, 41, 43, 45, 49, 54, 56, 59, 64, 72, 73, 74, 79, 82, 85, 88, 91,
  94, 97, 101, 104, 107, 111, 31, 32, 32, 33, 33, 34, 35, 36, 36, 39, 42,
  44, 45, 50, 54, 56, 59, 64, 71, 72, 74, 78, 81, 84, 86, 89, 93, 96, 100,
  104, 107, 111, 32, 32, 32, 33, 34, 35, 37, 37, 38, 40, 42, 44, 46, 49,
  53, 55, 58, 63, 69, 70, 72, 76, 79, 82, 85, 89, 93, 96, 99, 102, 106,
  109, 34, 34, 33, 34, 35, 37, 39, 41, 42, 45, 47, 49, 51, 54, 58, 60, 63,
  68, 74, 75, 76, 80, 81, 82, 84, 87, 90, 93, 97, 101, 105, 110, 35, 34,
  34, 35, 36, 37, 41, 43, 45, 47, 50, 52, 53, 57, 61, 63, 65, 70, 76, 77,
  79, 82, 84, 86, 89, 91, 92, 93, 96, 100, 103, 107, 36, 35, 34, 35, 36,
  38, 42, 45, 48, 50, 54, 55, 57, 60, 64, 66, 68, 73, 79, 80, 81, 85, 88,
  90, 91, 93, 96, 99, 102, 103, 105, 107, 39, 38, 37, 38, 39, 40, 45, 47,
  50, 54, 58, 59, 61, 65, 69, 71, 73, 78, 84, 85, 86, 91, 92, 92, 95, 98,
  100, 101, 103, 106, 110, 114, 44, 42, 41, 41, 42, 42, 47, 50, 54, 58,
  63, 65, 67, 71, 75, 77, 79, 84, 90, 91, 92, 95, 95, 97, 100, 101, 102,
  105, 109, 111, 112, 114, 46, 44, 42, 43, 44, 44, 49, 52, 55, 59, 65, 67,

```



```

69, 74, 78, 80, 82, 87, 93, 94, 95, 98, 100, 103, 102, 105, 108, 110,
111, 113, 117, 121, 48, 46, 44, 45, 45, 46, 51, 53, 57, 61, 67, 69, 71,
76, 80, 83, 85, 90, 96, 97, 99, 103, 107, 105, 108, 111, 111, 113, 117,
119, 120, 122, 54, 51, 49, 49, 50, 49, 54, 57, 60, 65, 71, 74, 76, 82,
87, 89, 92, 97, 104, 105, 106, 111, 110, 111, 114, 113, 116, 120, 120,
121, 125, 130, 59, 56, 54, 54, 54, 53, 58, 61, 64, 69, 75, 78, 80, 87,
92, 95, 98, 103, 110, 111, 113, 115, 115, 119, 116, 120, 122, 122, 125,
129, 130, 130, 62, 59, 56, 56, 56, 55, 60, 63, 66, 71, 77, 80, 83, 89,
95, 98, 101, 107, 114, 115, 117, 119, 123, 121, 125, 126, 125, 129, 131,
131, 135, 140, 65, 62, 59, 59, 59, 58, 63, 65, 68, 73, 79, 82, 85, 92,
98, 101, 105, 111, 118, 119, 121, 126, 127, 128, 130, 130, 134, 133,
135, 140, 140, 140, 71, 68, 65, 64, 64, 63, 68, 70, 73, 78, 84, 87, 90,
97, 103, 107, 111, 117, 125, 126, 128, 134, 132, 136, 133, 138, 137,
140, 143, 142, 145, 150, 80, 76, 72, 72, 71, 69, 74, 76, 79, 84, 90, 93,
96, 104, 110, 114, 118, 125, 134, 135, 137, 139, 140, 139, 143, 142,
144, 146, 146, 151, 152, 151, 81, 77, 73, 73, 72, 70, 75, 77, 80, 85,
91, 94, 97, 105, 111, 115, 119, 126, 135, 137, 138, 144, 147, 146, 148,
149, 151, 150, 156, 155, 157, 163, 83, 78, 75, 74, 74, 72, 76, 79, 81,
86, 92, 95, 99, 106, 113, 117, 121, 128, 137, 138, 140, 147, 151, 156,
152, 157, 155, 161, 158, 162, 165, 164, 88, 84, 80, 79, 78, 76, 80, 82,
85, 91, 95, 98, 103, 111, 115, 119, 126, 134, 139, 144, 147, 152, 154,
158, 163, 159, 165, 163, 168, 168, 169, 176, 91, 86, 83, 82, 81, 79, 81,
84, 88, 92, 95, 100, 107, 110, 115, 123, 127, 132, 140, 147, 151, 154,
159, 161, 166, 171, 169, 173, 173, 176, 179, 177, 94, 89, 86, 85, 84,
82, 82, 86, 90, 92, 97, 103, 105, 111, 119, 121, 128, 136, 139, 146,
156, 158, 161, 166, 168, 174, 179, 178, 180, 183, 183, 190, 97, 92, 90,
88, 86, 85, 84, 89, 91, 95, 100, 102, 108, 114, 116, 125, 130, 133, 143,
148, 152, 163, 166, 168, 174, 176, 182, 187, 189, 188, 193, 191, 101,
95, 93, 91, 89, 89, 87, 91, 93, 98, 101, 105, 111, 113, 120, 126, 130,
138, 142, 149, 157, 159, 171, 174, 176, 183, 184, 191, 195, 199, 197,
204, 104, 99, 97, 94, 93, 93, 90, 92, 96, 100, 102, 108, 111, 116, 122,
125, 134, 137, 144, 151, 155, 165, 169, 179, 182, 184, 191, 193, 200,
204, 210, 206, 107, 102, 101, 97, 96, 96, 93, 93, 99, 101, 105, 110,
113, 120, 122, 129, 133, 140, 146, 150, 161, 163, 173, 178, 187, 191,
193, 200, 202, 210, 214, 222, 111, 105, 104, 101, 100, 99, 97, 96, 102,
103, 109, 111, 117, 120, 125, 131, 135, 143, 146, 156, 158, 168, 173,
180, 189, 195, 200, 202, 210, 212, 220, 224, 115, 109, 108, 104, 104,
102, 101, 100, 103, 106, 111, 113, 119, 121, 129, 131, 140, 142, 151,
155, 162, 168, 176, 183, 188, 199, 204, 210, 212, 220, 222, 230, 119,
113, 112, 107, 107, 106, 105, 103, 105, 110, 112, 117, 120, 125, 130,
135, 140, 145, 152, 157, 165, 169, 179, 183, 193, 197, 210, 214, 220,
222, 231, 232, 123, 116, 116, 111, 111, 109, 110, 107, 107, 114, 114,
121, 122, 130, 130, 140, 140, 150, 151, 163, 164, 176, 177, 190, 191,
204, 206, 222, 224, 230, 232, 242,
/* Size 4x8 */
32, 42, 75, 91, 33, 42, 69, 86, 37, 58, 84, 91, 49, 71, 103, 110, 65,
84, 125, 128, 80, 97, 142, 152, 91, 100, 145, 178, 104, 112, 146, 190,
/* Size 8x4 */
32, 33, 37, 49, 65, 80, 91, 104, 42, 42, 58, 71, 84, 97, 100, 112, 75,
69, 84, 103, 125, 142, 145, 146, 91, 86, 91, 110, 128, 152, 178, 190,

```

```
/* Size 8x16 */
```

```
32, 32, 36, 53, 65, 87, 93, 99, 31, 33, 34, 49, 59, 78, 86, 93, 32, 34,
36, 50, 59, 77, 82, 89, 34, 37, 42, 54, 63, 79, 80, 88, 36, 38, 48, 60,
68, 84, 86, 90, 44, 43, 53, 71, 79, 95, 94, 97, 48, 46, 56, 76, 85, 102,
105, 105, 58, 54, 63, 87, 98, 116, 112, 115, 65, 58, 68, 92, 105, 124,
122, 124, 79, 70, 79, 104, 118, 141, 135, 135, 82, 72, 81, 106, 121,
144, 149, 146, 91, 80, 88, 106, 130, 148, 162, 159, 97, 86, 94, 107,
128, 157, 167, 171, 103, 93, 98, 114, 131, 150, 174, 186, 110, 100, 101,
117, 138, 161, 183, 193, 118, 107, 105, 118, 136, 157, 182, 203,
```

```
/* Size 16x8 */
```

```
32, 31, 32, 34, 36, 44, 48, 58, 65, 79, 82, 91, 97, 103, 110, 118, 32,
33, 34, 37, 38, 43, 46, 54, 58, 70, 72, 80, 86, 93, 100, 107, 36, 34,
36, 42, 48, 53, 56, 63, 68, 79, 81, 88, 94, 98, 101, 105, 53, 49, 50,
54, 60, 71, 76, 87, 92, 104, 106, 106, 107, 114, 117, 118, 65, 59, 59,
63, 68, 79, 85, 98, 105, 118, 121, 130, 128, 131, 138, 136, 87, 78, 77,
79, 84, 95, 102, 116, 124, 141, 144, 148, 157, 150, 161, 157, 93, 86,
82, 80, 86, 94, 105, 112, 122, 135, 149, 162, 167, 174, 183, 182, 99,
93, 89, 88, 90, 97, 105, 115, 124, 135, 146, 159, 171, 186, 193, 203,
```

```
/* Size 16x32 */
```

```
32, 31, 32, 34, 36, 44, 53, 59, 65, 79, 87, 90, 93, 96, 99, 102, 31, 32,
32, 34, 35, 42, 51, 56, 62, 75, 82, 85, 88, 91, 94, 97, 31, 32, 33, 33,
34, 41, 49, 54, 59, 72, 78, 82, 86, 90, 93, 97, 31, 32, 33, 34, 35, 41,
49, 54, 59, 71, 78, 81, 84, 87, 90, 93, 32, 32, 34, 35, 36, 42, 50, 54,
59, 71, 77, 80, 82, 86, 89, 93, 32, 33, 35, 37, 38, 42, 49, 53, 58, 69,
75, 78, 82, 86, 89, 92, 34, 34, 37, 39, 42, 48, 54, 58, 63, 73, 79, 78,
80, 83, 88, 92, 35, 34, 37, 41, 45, 50, 57, 61, 65, 76, 82, 83, 84, 84,
87, 90, 36, 34, 38, 43, 48, 54, 60, 64, 68, 78, 84, 87, 86, 89, 90, 90,
39, 37, 40, 45, 50, 58, 65, 69, 73, 84, 89, 89, 91, 91, 93, 96, 44, 41,
43, 48, 53, 63, 71, 75, 79, 90, 95, 93, 94, 95, 97, 97, 46, 43, 44, 49,
55, 65, 73, 78, 82, 93, 98, 100, 98, 100, 99, 103, 48, 45, 46, 51, 56,
67, 76, 80, 85, 96, 102, 102, 105, 102, 105, 104, 53, 49, 50, 54, 60,
71, 82, 87, 92, 103, 109, 107, 107, 110, 107, 111, 58, 54, 54, 58, 63,
75, 87, 92, 98, 110, 116, 115, 112, 111, 115, 112, 61, 57, 56, 60, 66,
77, 89, 95, 101, 114, 120, 118, 119, 118, 116, 120, 65, 60, 58, 63, 68,
79, 92, 98, 105, 118, 124, 123, 122, 123, 124, 121, 71, 65, 63, 68, 73,
84, 97, 103, 111, 125, 132, 132, 130, 128, 127, 130, 79, 72, 70, 74, 79,
90, 104, 110, 118, 133, 141, 136, 135, 135, 135, 131, 81, 74, 71, 75,
80, 91, 105, 112, 119, 135, 142, 140, 140, 138, 139, 142, 82, 75, 72,
76, 81, 92, 106, 113, 121, 136, 144, 151, 149, 149, 146, 143, 88, 80,
77, 80, 85, 97, 108, 115, 126, 142, 149, 153, 153, 152, 152, 154, 91,
83, 80, 81, 88, 100, 106, 114, 130, 142, 148, 155, 162, 160, 159, 155,
94, 85, 83, 82, 91, 100, 105, 118, 131, 137, 153, 160, 165, 167, 166,
168, 97, 88, 86, 85, 94, 100, 107, 123, 128, 140, 157, 161, 167, 173,
171, 169, 100, 91, 89, 87, 97, 100, 111, 121, 127, 145, 152, 164, 173,
178, 182, 181, 103, 94, 93, 90, 98, 101, 114, 120, 131, 144, 150, 170,
174, 180, 186, 183, 107, 97, 96, 93, 100, 104, 117, 119, 136, 142, 155,
168, 177, 187, 191, 198, 110, 101, 100, 97, 101, 108, 117, 123, 138,
141, 161, 165, 183, 188, 193, 200, 114, 104, 104, 100, 103, 112, 117,
127, 137, 146, 159, 167, 185, 190, 201, 206, 118, 108, 107, 103, 105,
115, 118, 131, 136, 151, 157, 172, 182, 197, 203, 208, 122, 111, 111,
```

```

107, 107, 119, 119, 136, 136, 156, 156, 178, 179, 203, 204, 217,
/* Size 32x16 */
32, 31, 31, 31, 32, 32, 34, 35, 36, 39, 44, 46, 48, 53, 58, 61, 65, 71,
79, 81, 82, 88, 91, 94, 97, 100, 103, 107, 110, 114, 118, 122, 31, 32,
32, 32, 32, 33, 34, 34, 34, 37, 41, 43, 45, 49, 54, 57, 60, 65, 72, 74,
75, 80, 83, 85, 88, 91, 94, 97, 101, 104, 108, 111, 32, 32, 33, 33, 34,
35, 37, 37, 38, 40, 43, 44, 46, 50, 54, 56, 58, 63, 70, 71, 72, 77, 80,
83, 86, 89, 93, 96, 100, 104, 107, 111, 34, 34, 33, 34, 35, 37, 39, 41,
43, 45, 48, 49, 51, 54, 58, 60, 63, 68, 74, 75, 76, 80, 81, 82, 85, 87,
90, 93, 97, 100, 103, 107, 36, 35, 34, 35, 36, 38, 42, 45, 48, 50, 53,
55, 56, 60, 63, 66, 68, 73, 79, 80, 81, 85, 88, 91, 94, 97, 98, 100,
101, 103, 105, 107, 44, 42, 41, 41, 42, 42, 48, 50, 54, 58, 63, 65, 67,
71, 75, 77, 79, 84, 90, 91, 92, 97, 100, 100, 100, 100, 101, 104, 108,
112, 115, 119, 53, 51, 49, 49, 50, 49, 54, 57, 60, 65, 71, 73, 76, 82,
87, 89, 92, 97, 104, 105, 106, 108, 106, 105, 107, 111, 114, 117, 117,
117, 118, 119, 59, 56, 54, 54, 54, 53, 58, 61, 64, 69, 75, 78, 80, 87,
92, 95, 98, 103, 110, 112, 113, 115, 114, 118, 123, 121, 120, 119, 123,
127, 131, 136, 65, 62, 59, 59, 59, 58, 63, 65, 68, 73, 79, 82, 85, 92,
98, 101, 105, 111, 118, 119, 121, 126, 130, 131, 128, 127, 131, 136,
138, 137, 136, 136, 79, 75, 72, 71, 71, 69, 73, 76, 78, 84, 90, 93, 96,
103, 110, 114, 118, 125, 133, 135, 136, 142, 142, 137, 140, 145, 144,
142, 141, 146, 151, 156, 87, 82, 78, 78, 77, 75, 79, 82, 84, 89, 95, 98,
102, 109, 116, 120, 124, 132, 141, 142, 144, 149, 148, 153, 157, 152,
150, 155, 161, 159, 157, 156, 90, 85, 82, 81, 80, 78, 78, 83, 87, 89,
93, 100, 102, 107, 115, 118, 123, 132, 136, 140, 151, 153, 155, 160,
161, 164, 170, 168, 165, 167, 172, 178, 93, 88, 86, 84, 82, 82, 80, 84,
86, 91, 94, 98, 105, 107, 112, 119, 122, 130, 135, 140, 149, 153, 162,
165, 167, 173, 174, 177, 183, 185, 182, 179, 96, 91, 90, 87, 86, 86, 83,
84, 89, 91, 95, 100, 102, 110, 111, 118, 123, 128, 135, 138, 149, 152,
160, 167, 173, 178, 180, 187, 188, 190, 197, 203, 99, 94, 93, 90, 89,
89, 88, 87, 90, 93, 97, 99, 105, 107, 115, 116, 124, 127, 135, 139, 146,
152, 159, 166, 171, 182, 186, 191, 193, 201, 203, 204, 102, 97, 97, 93,
93, 92, 92, 90, 90, 96, 97, 103, 104, 111, 112, 120, 121, 130, 131, 142,
143, 154, 155, 168, 169, 181, 183, 198, 200, 206, 208, 217,
/* Size 4x16 */
31, 44, 79, 96, 32, 41, 72, 90, 32, 42, 71, 86, 34, 48, 73, 83, 34, 54,
78, 89, 41, 63, 90, 95, 45, 67, 96, 102, 54, 75, 110, 111, 60, 79, 118,
123, 72, 90, 133, 135, 75, 92, 136, 149, 83, 100, 142, 160, 88, 100,
140, 173, 94, 101, 144, 180, 101, 108, 141, 188, 108, 115, 151, 197,
/* Size 16x4 */
31, 32, 32, 34, 34, 41, 45, 54, 60, 72, 75, 83, 88, 94, 101, 108, 44,
41, 42, 48, 54, 63, 67, 75, 79, 90, 92, 100, 100, 101, 108, 115, 79, 72,
71, 73, 78, 90, 96, 110, 118, 133, 136, 142, 140, 144, 141, 151, 96, 90,
86, 83, 89, 95, 102, 111, 123, 135, 149, 160, 173, 180, 188, 197,
/* Size 8x32 */
32, 32, 36, 53, 65, 87, 93, 99, 31, 32, 35, 51, 62, 82, 88, 94, 31, 33,
34, 49, 59, 78, 86, 93, 31, 33, 35, 49, 59, 78, 84, 90, 32, 34, 36, 50,
59, 77, 82, 89, 32, 35, 38, 49, 58, 75, 82, 89, 34, 37, 42, 54, 63, 79,
80, 88, 35, 37, 45, 57, 65, 82, 84, 87, 36, 38, 48, 60, 68, 84, 86, 90,
39, 40, 50, 65, 73, 89, 91, 93, 44, 43, 53, 71, 79, 95, 94, 97, 46, 44,

```

```

55, 73, 82, 98, 98, 99, 48, 46, 56, 76, 85, 102, 105, 105, 53, 50, 60,
82, 92, 109, 107, 107, 58, 54, 63, 87, 98, 116, 112, 115, 61, 56, 66,
89, 101, 120, 119, 116, 65, 58, 68, 92, 105, 124, 122, 124, 71, 63, 73,
97, 111, 132, 130, 127, 79, 70, 79, 104, 118, 141, 135, 135, 81, 71, 80,
105, 119, 142, 140, 139, 82, 72, 81, 106, 121, 144, 149, 146, 88, 77,
85, 108, 126, 149, 153, 152, 91, 80, 88, 106, 130, 148, 162, 159, 94,
83, 91, 105, 131, 153, 165, 166, 97, 86, 94, 107, 128, 157, 167, 171,
100, 89, 97, 111, 127, 152, 173, 182, 103, 93, 98, 114, 131, 150, 174,
186, 107, 96, 100, 117, 136, 155, 177, 191, 110, 100, 101, 117, 138,
161, 183, 193, 114, 104, 103, 117, 137, 159, 185, 201, 118, 107, 105,
118, 136, 157, 182, 203, 122, 111, 107, 119, 136, 156, 179, 204,
/* Size 32x8 */
32, 31, 31, 31, 32, 32, 34, 35, 36, 39, 44, 46, 48, 53, 58, 61, 65, 71,
79, 81, 82, 88, 91, 94, 97, 100, 103, 107, 110, 114, 118, 122, 32, 32,
33, 33, 34, 35, 37, 37, 38, 40, 43, 44, 46, 50, 54, 56, 58, 63, 70, 71,
72, 77, 80, 83, 86, 89, 93, 96, 100, 104, 107, 111, 36, 35, 34, 35, 36,
38, 42, 45, 48, 50, 53, 55, 56, 60, 63, 66, 68, 73, 79, 80, 81, 85, 88,
91, 94, 97, 98, 100, 101, 103, 105, 107, 53, 51, 49, 49, 50, 49, 54, 57,
60, 65, 71, 73, 76, 82, 87, 89, 92, 97, 104, 105, 106, 108, 106, 105,
107, 111, 114, 117, 117, 117, 118, 119, 65, 62, 59, 59, 59, 58, 63, 65,
68, 73, 79, 82, 85, 92, 98, 101, 105, 111, 118, 119, 121, 126, 130, 131,
128, 127, 131, 136, 138, 137, 136, 136, 87, 82, 78, 78, 77, 75, 79, 82,
84, 89, 95, 98, 102, 109, 116, 120, 124, 132, 141, 142, 144, 149, 148,
153, 157, 152, 150, 155, 161, 159, 157, 156, 93, 88, 86, 84, 82, 82, 80,
84, 86, 91, 94, 98, 105, 107, 112, 119, 122, 130, 135, 140, 149, 153,
162, 165, 167, 173, 174, 177, 183, 185, 182, 179, 99, 94, 93, 90, 89,
89, 88, 87, 90, 93, 97, 99, 105, 107, 115, 116, 124, 127, 135, 139, 146,
152, 159, 166, 171, 182, 186, 191, 193, 201, 203, 204 },
{ /* Chroma */
/* Size 4x4 */
35, 46, 57, 66, 46, 60, 69, 71, 57, 69, 90, 90, 66, 71, 90, 109,
/* Size 8x8 */
31, 38, 47, 50, 57, 63, 67, 71, 38, 47, 46, 47, 52, 57, 62, 67, 47, 46,
54, 57, 61, 66, 67, 68, 50, 47, 57, 66, 72, 77, 75, 75, 57, 52, 61, 72,
82, 88, 86, 84, 63, 57, 66, 77, 88, 96, 95, 95, 67, 62, 67, 75, 86, 95,
104, 107, 71, 67, 68, 75, 84, 95, 107, 113,
/* Size 16x16 */
32, 30, 33, 41, 49, 49, 50, 54, 57, 63, 65, 68, 70, 72, 74, 76, 30, 32,
35, 42, 46, 45, 46, 49, 52, 57, 58, 62, 64, 67, 70, 72, 33, 35, 39, 45,
47, 45, 46, 49, 51, 56, 57, 60, 62, 64, 66, 69, 41, 42, 45, 48, 50, 49,
50, 52, 53, 57, 58, 59, 60, 61, 64, 67, 49, 46, 47, 50, 53, 53, 54, 55,
56, 60, 61, 64, 64, 65, 66, 66, 49, 45, 45, 49, 53, 58, 60, 62, 63, 67,
68, 67, 69, 68, 70, 70, 50, 46, 46, 50, 54, 60, 61, 65, 67, 71, 71, 74,
73, 73, 74, 74, 54, 49, 49, 52, 55, 62, 65, 71, 73, 78, 79, 78, 77, 78,
78, 78, 57, 52, 51, 53, 56, 63, 67, 73, 76, 82, 83, 84, 84, 84, 82, 83,
63, 57, 56, 57, 60, 67, 71, 78, 82, 89, 90, 90, 89, 88, 87, 88, 65, 58,
57, 58, 61, 68, 71, 79, 83, 90, 91, 94, 93, 93, 92, 93, 68, 62, 60, 59,
64, 67, 74, 78, 84, 90, 94, 98, 99, 98, 98, 98, 70, 64, 62, 60, 64, 69,
73, 77, 84, 89, 93, 99, 102, 103, 104, 104, 72, 67, 64, 61, 65, 68, 73,
78, 84, 88, 93, 98, 103, 106, 108, 109, 74, 70, 66, 64, 66, 70, 74, 78,

```

```

82, 87, 92, 98, 104, 108, 111, 112, 76, 72, 69, 67, 66, 70, 74, 78, 83,
88, 93, 98, 104, 109, 112, 116,
/* Size 32x32 */
32, 31, 30, 32, 33, 36, 41, 45, 49, 48, 49, 50, 50, 52, 54, 56, 57, 60,
63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 78, 31, 31, 31, 33,
34, 38, 42, 45, 47, 47, 47, 47, 48, 50, 52, 53, 54, 57, 60, 61, 61, 63,
64, 65, 66, 67, 68, 69, 70, 71, 72, 74, 30, 31, 32, 33, 35, 40, 42, 44,
46, 45, 45, 45, 46, 47, 49, 51, 52, 54, 57, 58, 58, 61, 62, 63, 64, 66,
67, 68, 70, 71, 72, 74, 32, 33, 33, 35, 37, 41, 43, 45, 47, 46, 45, 46,
46, 47, 49, 50, 51, 54, 57, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68,
69, 70, 33, 34, 35, 37, 39, 43, 45, 46, 47, 46, 45, 46, 46, 47, 49, 50,
51, 53, 56, 57, 57, 59, 60, 61, 62, 63, 64, 65, 66, 68, 69, 70, 36, 38,
40, 41, 43, 47, 47, 47, 48, 46, 45, 46, 46, 47, 48, 49, 50, 52, 54, 55,
55, 57, 58, 59, 61, 62, 64, 65, 66, 67, 68, 69, 41, 42, 42, 43, 45, 47,
48, 49, 50, 49, 49, 49, 50, 50, 52, 52, 53, 55, 57, 58, 58, 60, 59, 59,
60, 61, 61, 63, 64, 66, 67, 69, 45, 45, 44, 45, 46, 47, 49, 50, 51, 51,
51, 51, 52, 52, 53, 54, 55, 57, 59, 59, 60, 61, 61, 62, 63, 63, 63, 63,
63, 64, 65, 66, 49, 47, 46, 47, 47, 48, 50, 51, 53, 53, 53, 54, 54, 54,
55, 56, 56, 58, 60, 61, 61, 63, 64, 64, 64, 64, 65, 66, 66, 66, 66, 66,
48, 47, 45, 46, 46, 46, 49, 51, 53, 54, 55, 56, 56, 57, 58, 59, 60, 61,
63, 64, 64, 66, 66, 65, 66, 67, 67, 67, 67, 68, 69, 70, 49, 47, 45, 45,
45, 45, 49, 51, 53, 55, 58, 59, 60, 61, 62, 63, 63, 65, 67, 67, 68, 69,
67, 68, 69, 68, 68, 69, 70, 70, 70, 70, 50, 47, 45, 46, 46, 46, 49, 51,
54, 56, 59, 60, 60, 62, 64, 64, 65, 67, 69, 69, 70, 70, 71, 71, 70, 70,
71, 71, 71, 71, 72, 74, 50, 48, 46, 46, 46, 46, 50, 52, 54, 56, 60, 60,
61, 63, 65, 66, 67, 68, 71, 71, 71, 73, 74, 72, 73, 74, 73, 73, 74, 74,
74, 74, 52, 50, 47, 47, 47, 47, 50, 52, 54, 57, 61, 62, 63, 66, 68, 69,
70, 72, 75, 75, 75, 77, 75, 75, 76, 75, 75, 76, 75, 75, 76, 77, 54, 52,
49, 49, 49, 48, 52, 53, 55, 58, 62, 64, 65, 68, 71, 72, 73, 75, 78, 78,
79, 79, 78, 79, 77, 78, 78, 77, 78, 79, 78, 78, 56, 53, 51, 50, 50, 49,
52, 54, 56, 59, 63, 64, 66, 69, 72, 73, 75, 77, 80, 80, 81, 81, 82, 80,
81, 81, 79, 81, 80, 79, 81, 82, 57, 54, 52, 51, 51, 50, 53, 55, 56, 60,
63, 65, 67, 70, 73, 75, 76, 79, 82, 82, 83, 85, 84, 83, 84, 83, 84, 82,
82, 84, 83, 82, 60, 57, 54, 54, 53, 52, 55, 57, 58, 61, 65, 67, 68, 72,
75, 77, 79, 82, 85, 85, 86, 88, 86, 87, 85, 86, 85, 85, 86, 84, 85, 86,
63, 60, 57, 57, 56, 54, 57, 59, 60, 63, 67, 69, 71, 75, 78, 80, 82, 85,
89, 89, 90, 90, 90, 89, 89, 88, 88, 88, 87, 88, 88, 87, 64, 61, 58, 57,
57, 55, 58, 59, 61, 64, 67, 69, 71, 75, 78, 80, 82, 85, 89, 90, 91, 92,
93, 92, 92, 91, 91, 90, 91, 90, 90, 92, 65, 61, 58, 58, 57, 55, 58, 60,
61, 64, 68, 70, 71, 75, 79, 81, 83, 86, 90, 91, 91, 94, 94, 96, 93, 94,
93, 94, 92, 93, 93, 92, 67, 63, 61, 60, 59, 57, 60, 61, 63, 66, 69, 70,
73, 77, 79, 81, 85, 88, 90, 92, 94, 96, 96, 97, 98, 95, 97, 95, 96, 95,
95, 96, 68, 64, 62, 61, 60, 58, 59, 61, 64, 66, 67, 71, 74, 75, 78, 82,
84, 86, 90, 93, 94, 96, 98, 98, 99, 100, 98, 99, 98, 98, 98, 97, 69, 65,
63, 62, 61, 59, 59, 62, 64, 65, 68, 71, 72, 75, 79, 80, 83, 87, 89, 92,
96, 97, 98, 100, 100, 101, 102, 101, 101, 101, 100, 102, 70, 66, 64, 63,
62, 61, 60, 63, 64, 66, 69, 70, 73, 76, 77, 81, 84, 85, 89, 92, 93, 98,
99, 100, 102, 102, 103, 104, 104, 103, 104, 102, 71, 67, 66, 64, 63, 62,
61, 63, 64, 67, 68, 70, 74, 75, 78, 81, 83, 86, 88, 91, 94, 95, 100,
101, 102, 104, 104, 105, 106, 107, 105, 107, 72, 68, 67, 65, 64, 64, 61,

```

```

63, 65, 67, 68, 71, 73, 75, 78, 79, 84, 85, 88, 91, 93, 97, 98, 102,
103, 104, 106, 106, 108, 108, 109, 107, 73, 69, 68, 66, 65, 65, 63, 63,
66, 67, 69, 71, 73, 76, 77, 81, 82, 85, 88, 90, 94, 95, 99, 101, 104,
105, 106, 109, 108, 110, 111, 112, 74, 70, 70, 67, 66, 66, 64, 63, 66,
67, 70, 71, 74, 75, 78, 80, 82, 86, 87, 91, 92, 96, 98, 101, 104, 106,
108, 108, 111, 111, 112, 113, 75, 71, 71, 68, 68, 67, 66, 64, 66, 68,
70, 71, 74, 75, 79, 79, 84, 84, 88, 90, 93, 95, 98, 101, 103, 107, 108,
110, 111, 113, 113, 115, 76, 72, 72, 69, 69, 68, 67, 65, 66, 69, 70, 72,
74, 76, 78, 81, 83, 85, 88, 90, 93, 95, 98, 100, 104, 105, 109, 111,
112, 113, 116, 115, 78, 74, 74, 70, 70, 69, 69, 66, 66, 70, 70, 74, 74,
77, 78, 82, 82, 86, 87, 92, 92, 96, 97, 102, 102, 107, 107, 112, 113,
115, 115, 118,
/* Size 4x8 */
31, 47, 60, 66, 40, 45, 54, 61, 46, 56, 64, 64, 48, 61, 75, 73, 54, 65,
85, 82, 61, 69, 92, 92, 64, 68, 90, 102, 68, 71, 87, 105,
/* Size 8x4 */
31, 40, 46, 48, 54, 61, 64, 68, 47, 45, 56, 61, 65, 69, 68, 71, 60, 54,
64, 75, 85, 92, 90, 87, 66, 61, 64, 73, 82, 92, 102, 105,
/* Size 8x16 */
32, 37, 48, 52, 57, 66, 68, 71, 30, 40, 46, 48, 52, 60, 63, 66, 33, 43,
47, 47, 51, 59, 60, 63, 42, 47, 50, 50, 53, 60, 59, 62, 49, 48, 53, 54,
57, 62, 62, 62, 49, 46, 53, 61, 64, 69, 66, 66, 50, 46, 54, 64, 67, 73,
72, 70, 54, 49, 55, 68, 73, 80, 76, 75, 57, 50, 56, 70, 76, 84, 80, 79,
63, 55, 60, 75, 82, 92, 87, 84, 64, 56, 61, 75, 83, 93, 93, 89, 68, 59,
64, 74, 86, 94, 98, 94, 70, 62, 66, 73, 83, 96, 99, 98, 72, 64, 66, 75,
83, 92, 101, 104, 74, 67, 66, 74, 84, 94, 103, 106, 76, 69, 67, 73, 82,
91, 101, 109,
/* Size 16x8 */
32, 30, 33, 42, 49, 49, 50, 54, 57, 63, 64, 68, 70, 72, 74, 76, 37, 40,
43, 47, 48, 46, 46, 49, 50, 55, 56, 59, 62, 64, 67, 69, 48, 46, 47, 50,
53, 53, 54, 55, 56, 60, 61, 64, 66, 66, 66, 67, 52, 48, 47, 50, 54, 61,
64, 68, 70, 75, 75, 74, 73, 75, 74, 73, 57, 52, 51, 53, 57, 64, 67, 73,
76, 82, 83, 86, 83, 83, 84, 82, 66, 60, 59, 60, 62, 69, 73, 80, 84, 92,
93, 94, 96, 92, 94, 91, 68, 63, 60, 59, 62, 66, 72, 76, 80, 87, 93, 98,
99, 101, 103, 101, 71, 66, 63, 62, 62, 66, 70, 75, 79, 84, 89, 94, 98,
104, 106, 109,
/* Size 16x32 */
32, 31, 37, 42, 48, 49, 52, 54, 57, 63, 66, 67, 68, 69, 71, 72, 31, 31,
38, 42, 47, 47, 50, 52, 54, 60, 63, 64, 65, 66, 67, 68, 30, 32, 40, 42,
46, 45, 48, 50, 52, 57, 60, 62, 63, 65, 66, 68, 32, 34, 41, 44, 46, 45,
48, 49, 51, 57, 59, 61, 62, 63, 64, 65, 33, 36, 43, 45, 47, 46, 47, 49,
51, 56, 59, 60, 60, 62, 63, 65, 37, 40, 47, 47, 47, 45, 47, 48, 50, 54,
57, 58, 60, 61, 62, 63, 42, 43, 47, 48, 50, 49, 50, 52, 53, 57, 60, 58,
59, 60, 62, 63, 45, 44, 47, 49, 51, 51, 52, 54, 55, 59, 61, 61, 61, 60,
61, 61, 49, 46, 48, 50, 53, 53, 54, 55, 57, 60, 62, 63, 62, 63, 62, 62,
48, 46, 47, 50, 53, 56, 57, 59, 60, 64, 66, 65, 65, 64, 64, 65, 49, 45,
46, 49, 53, 58, 61, 62, 64, 67, 69, 67, 66, 66, 66, 65, 49, 46, 46, 49,
53, 59, 62, 64, 65, 69, 71, 70, 68, 68, 67, 68, 50, 46, 46, 50, 54, 59,
64, 65, 67, 71, 73, 72, 72, 70, 70, 69, 52, 48, 47, 50, 54, 61, 66, 68,
71, 75, 77, 74, 73, 73, 71, 72, 54, 50, 49, 52, 55, 62, 68, 71, 73, 78,

```

```

80, 78, 76, 74, 75, 73, 55, 51, 49, 52, 56, 63, 69, 72, 75, 80, 82, 80,
79, 78, 76, 77, 57, 52, 50, 53, 56, 64, 70, 73, 76, 82, 84, 82, 80, 80,
79, 77, 60, 54, 52, 55, 58, 65, 72, 75, 79, 85, 88, 86, 84, 82, 81, 81,
63, 57, 55, 58, 60, 67, 75, 78, 82, 89, 92, 88, 87, 85, 84, 81, 64, 58,
55, 58, 61, 68, 75, 78, 82, 89, 92, 90, 89, 87, 86, 86, 64, 59, 56, 58,
61, 68, 75, 79, 83, 90, 93, 95, 93, 91, 89, 87, 67, 61, 58, 60, 63, 69,
76, 79, 85, 92, 95, 96, 94, 92, 91, 91, 68, 62, 59, 60, 64, 71, 74, 78,
86, 91, 94, 96, 98, 96, 94, 91, 69, 62, 60, 60, 65, 70, 72, 79, 85, 88,
95, 98, 99, 98, 97, 96, 70, 63, 62, 60, 66, 69, 73, 81, 83, 89, 96, 97,
99, 101, 98, 97, 71, 64, 63, 61, 67, 68, 74, 79, 82, 90, 93, 98, 102,
102, 102, 101, 72, 65, 64, 62, 66, 68, 75, 78, 83, 89, 92, 100, 101,
103, 104, 102, 73, 66, 65, 63, 66, 69, 75, 76, 84, 87, 93, 98, 102, 105,
106, 107, 74, 67, 67, 64, 66, 70, 74, 77, 84, 86, 94, 96, 103, 105, 106,
107, 75, 68, 68, 65, 66, 71, 74, 78, 83, 87, 93, 96, 103, 105, 109, 109,
76, 69, 69, 66, 67, 72, 73, 80, 82, 88, 91, 97, 101, 107, 109, 110, 77,
70, 70, 67, 67, 73, 73, 81, 81, 90, 90, 99, 99, 108, 108, 113,

```

```
/* Size 32x16 */
```

```

32, 31, 30, 32, 33, 37, 42, 45, 49, 48, 49, 49, 50, 52, 54, 55, 57, 60,
63, 64, 64, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 31, 31, 32, 34,
36, 40, 43, 44, 46, 46, 45, 46, 46, 48, 50, 51, 52, 54, 57, 58, 59, 61,
62, 62, 63, 64, 65, 66, 67, 68, 69, 70, 37, 38, 40, 41, 43, 47, 47, 47,
48, 47, 46, 46, 46, 47, 49, 49, 50, 52, 55, 55, 56, 58, 59, 60, 62, 63,
64, 65, 67, 68, 69, 70, 42, 42, 42, 44, 45, 47, 48, 49, 50, 50, 49, 49,
50, 50, 52, 52, 53, 55, 58, 58, 58, 60, 60, 60, 60, 61, 62, 63, 64, 65,
66, 67, 48, 47, 46, 46, 47, 47, 50, 51, 53, 53, 53, 53, 54, 54, 55, 56,
56, 58, 60, 61, 61, 63, 64, 65, 66, 67, 66, 66, 66, 66, 67, 67, 49, 47,
45, 45, 46, 45, 49, 51, 53, 56, 58, 59, 59, 61, 62, 63, 64, 65, 67, 68,
68, 69, 71, 70, 69, 68, 68, 69, 70, 71, 72, 73, 52, 50, 48, 48, 47, 47,
50, 52, 54, 57, 61, 62, 64, 66, 68, 69, 70, 72, 75, 75, 75, 76, 74, 72,
73, 74, 75, 75, 74, 74, 73, 73, 54, 52, 50, 49, 49, 48, 52, 54, 55, 59,
62, 64, 65, 68, 71, 72, 73, 75, 78, 78, 79, 79, 78, 79, 81, 79, 78, 76,
77, 78, 80, 81, 57, 54, 52, 51, 51, 50, 53, 55, 57, 60, 64, 65, 67, 71,
73, 75, 76, 79, 82, 82, 83, 85, 86, 85, 83, 82, 83, 84, 84, 83, 82, 81,
63, 60, 57, 57, 56, 54, 57, 59, 60, 64, 67, 69, 71, 75, 78, 80, 82, 85,
89, 89, 90, 92, 91, 88, 89, 90, 89, 87, 86, 87, 88, 90, 66, 63, 60, 59,
59, 57, 60, 61, 62, 66, 69, 71, 73, 77, 80, 82, 84, 88, 92, 92, 93, 95,
94, 95, 96, 93, 92, 93, 94, 93, 91, 90, 67, 64, 62, 61, 60, 58, 58, 61,
63, 65, 67, 70, 72, 74, 78, 80, 82, 86, 88, 90, 95, 96, 96, 98, 97, 98,
100, 98, 96, 96, 97, 99, 68, 65, 63, 62, 60, 60, 59, 61, 62, 65, 66, 68,
72, 73, 76, 79, 80, 84, 87, 89, 93, 94, 98, 99, 99, 102, 101, 102, 103,
103, 101, 99, 69, 66, 65, 63, 62, 61, 60, 60, 63, 64, 66, 68, 70, 73,
74, 78, 80, 82, 85, 87, 91, 92, 96, 98, 101, 102, 103, 105, 105, 105,
107, 108, 71, 67, 66, 64, 63, 62, 62, 61, 62, 64, 66, 67, 70, 71, 75,
76, 79, 81, 84, 86, 89, 91, 94, 97, 98, 102, 104, 106, 106, 109, 109,
108, 72, 68, 68, 65, 65, 63, 63, 61, 62, 65, 65, 68, 69, 72, 73, 77, 77,
81, 81, 86, 87, 91, 91, 96, 97, 101, 102, 107, 107, 109, 110, 113,

```

```
/* Size 4x16 */
```

```

31, 49, 63, 69, 32, 45, 57, 65, 36, 46, 56, 62, 43, 49, 57, 60, 46, 53,
60, 63, 45, 58, 67, 66, 46, 59, 71, 70, 50, 62, 78, 74, 52, 64, 82, 80,
57, 67, 89, 85, 59, 68, 90, 91, 62, 71, 91, 96, 63, 69, 89, 101, 65, 68,

```

```

89, 103, 67, 70, 86, 105, 69, 72, 88, 107,
/* Size 16x4 */
31, 32, 36, 43, 46, 45, 46, 50, 52, 57, 59, 62, 63, 65, 67, 69, 49, 45,
46, 49, 53, 58, 59, 62, 64, 67, 68, 71, 69, 68, 70, 72, 63, 57, 56, 57,
60, 67, 71, 78, 82, 89, 90, 91, 89, 89, 86, 88, 69, 65, 62, 60, 63, 66,
70, 74, 80, 85, 91, 96, 101, 103, 105, 107,
/* Size 8x32 */
32, 37, 48, 52, 57, 66, 68, 71, 31, 38, 47, 50, 54, 63, 65, 67, 30, 40,
46, 48, 52, 60, 63, 66, 32, 41, 46, 48, 51, 59, 62, 64, 33, 43, 47, 47,
51, 59, 60, 63, 37, 47, 47, 47, 50, 57, 60, 62, 42, 47, 50, 50, 53, 60,
59, 62, 45, 47, 51, 52, 55, 61, 61, 61, 49, 48, 53, 54, 57, 62, 62, 62,
48, 47, 53, 57, 60, 66, 65, 64, 49, 46, 53, 61, 64, 69, 66, 66, 49, 46,
53, 62, 65, 71, 68, 67, 50, 46, 54, 64, 67, 73, 72, 70, 52, 47, 54, 66,
71, 77, 73, 71, 54, 49, 55, 68, 73, 80, 76, 75, 55, 49, 56, 69, 75, 82,
79, 76, 57, 50, 56, 70, 76, 84, 80, 79, 60, 52, 58, 72, 79, 88, 84, 81,
63, 55, 60, 75, 82, 92, 87, 84, 64, 55, 61, 75, 82, 92, 89, 86, 64, 56,
61, 75, 83, 93, 93, 89, 67, 58, 63, 76, 85, 95, 94, 91, 68, 59, 64, 74,
86, 94, 98, 94, 69, 60, 65, 72, 85, 95, 99, 97, 70, 62, 66, 73, 83, 96,
99, 98, 71, 63, 67, 74, 82, 93, 102, 102, 72, 64, 66, 75, 83, 92, 101,
104, 73, 65, 66, 75, 84, 93, 102, 106, 74, 67, 66, 74, 84, 94, 103, 106,
75, 68, 66, 74, 83, 93, 103, 109, 76, 69, 67, 73, 82, 91, 101, 109, 77,
70, 67, 73, 81, 90, 99, 108,
/* Size 32x8 */
32, 31, 30, 32, 33, 37, 42, 45, 49, 48, 49, 49, 50, 52, 54, 55, 57, 60,
63, 64, 64, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 37, 38, 40, 41,
43, 47, 47, 47, 48, 47, 46, 46, 46, 47, 49, 49, 50, 52, 55, 55, 56, 58,
59, 60, 62, 63, 64, 65, 67, 68, 69, 70, 48, 47, 46, 46, 47, 47, 50, 51,
53, 53, 53, 53, 54, 54, 55, 56, 56, 58, 60, 61, 61, 63, 64, 65, 66, 67,
66, 66, 66, 66, 67, 67, 52, 50, 48, 48, 47, 47, 50, 52, 54, 57, 61, 62,
64, 66, 68, 69, 70, 72, 75, 75, 75, 76, 74, 72, 73, 74, 75, 75, 74, 74,
73, 73, 57, 54, 52, 51, 51, 50, 53, 55, 57, 60, 64, 65, 67, 71, 73, 75,
76, 79, 82, 82, 83, 85, 86, 85, 83, 82, 83, 84, 84, 83, 82, 81, 66, 63,
60, 59, 59, 57, 60, 61, 62, 66, 69, 71, 73, 77, 80, 82, 84, 88, 92, 92,
93, 95, 94, 95, 96, 93, 92, 93, 94, 93, 91, 90, 68, 65, 63, 62, 60, 60,
59, 61, 62, 65, 66, 68, 72, 73, 76, 79, 80, 84, 87, 89, 93, 94, 98, 99,
99, 102, 101, 102, 103, 103, 101, 99, 71, 67, 66, 64, 63, 62, 62, 61,
62, 64, 66, 67, 70, 71, 75, 76, 79, 81, 84, 86, 89, 91, 94, 97, 98, 102,
104, 106, 106, 109, 109, 108 },
},
{
{ /* Luma */
/* Size 4x4 */
32, 41, 69, 92, 41, 63, 88, 103, 69, 88, 127, 140, 92, 103, 140, 184,
/* Size 8x8 */
32, 32, 37, 47, 62, 78, 90, 102, 32, 35, 39, 46, 58, 72, 84, 96, 37, 39,
51, 60, 71, 84, 93, 100, 47, 46, 60, 73, 87, 100, 106, 113, 62, 58, 71,
87, 105, 121, 129, 132, 78, 72, 84, 100, 121, 140, 148, 155, 90, 84, 93,
106, 129, 148, 169, 183, 102, 96, 100, 113, 132, 155, 183, 201,
/* Size 16x16 */
32, 31, 31, 32, 36, 39, 47, 54, 61, 71, 80, 86, 92, 98, 104, 111, 31,

```



32, 32, 33, 34, 37, 44, 50, 56, 65, 73, 79, 85, 91, 98, 105, 31, 32, 33,  
 34, 36, 39, 45, 50, 56, 64, 71, 77, 82, 88, 94, 100, 32, 33, 34, 36, 40,  
 42, 47, 51, 57, 65, 71, 76, 80, 85, 91, 98, 36, 34, 36, 40, 48, 50, 56,  
 60, 65, 73, 79, 84, 86, 90, 95, 98, 39, 37, 39, 42, 50, 54, 60, 65, 70,  
 78, 84, 89, 95, 96, 102, 105, 47, 44, 45, 47, 56, 60, 69, 75, 81, 89,  
 95, 100, 102, 104, 109, 112, 54, 50, 50, 51, 60, 65, 75, 82, 89, 97,  
 104, 109, 110, 114, 117, 121, 61, 56, 56, 57, 65, 70, 81, 89, 97, 106,  
 113, 119, 122, 126, 125, 130, 71, 65, 64, 65, 73, 78, 89, 97, 106, 117,  
 125, 131, 134, 134, 136, 141, 80, 73, 71, 71, 79, 84, 95, 104, 113, 125,  
 134, 140, 142, 145, 146, 152, 86, 79, 77, 76, 84, 89, 100, 109, 119,  
 131, 140, 147, 154, 157, 160, 165, 92, 85, 82, 80, 86, 95, 102, 110,  
 122, 134, 142, 154, 162, 168, 174, 178, 98, 91, 88, 85, 90, 96, 104,  
 114, 126, 134, 145, 157, 168, 176, 184, 193, 104, 98, 94, 91, 95, 102,  
 109, 117, 125, 136, 146, 160, 174, 184, 193, 201, 111, 105, 100, 98, 98,  
 105, 112, 121, 130, 141, 152, 165, 178, 193, 201, 210,

/\* Size 32x32 \*/

32, 31, 31, 31, 31, 32, 32, 34, 36, 38, 39, 44, 47, 49, 54, 59, 61, 65,  
 71, 76, 80, 83, 86, 89, 92, 95, 98, 101, 104, 108, 111, 114, 31, 32, 32,  
 32, 32, 32, 33, 34, 35, 37, 38, 42, 45, 47, 51, 56, 58, 62, 68, 72, 76,  
 78, 82, 85, 88, 90, 93, 96, 99, 102, 105, 109, 31, 32, 32, 32, 32, 32,  
 33, 33, 34, 36, 37, 41, 44, 46, 50, 54, 56, 60, 65, 70, 73, 76, 79, 82,  
 85, 88, 91, 95, 98, 101, 105, 109, 31, 32, 32, 32, 32, 33, 33, 34, 35,  
 36, 38, 41, 44, 45, 49, 54, 56, 59, 65, 69, 72, 75, 78, 81, 84, 86, 89,  
 92, 95, 98, 101, 104, 31, 32, 32, 32, 33, 34, 34, 35, 36, 38, 39, 42,  
 45, 46, 50, 54, 56, 59, 64, 68, 71, 74, 77, 79, 82, 85, 88, 91, 94, 97,  
 100, 104, 32, 32, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 45, 46, 49,  
 53, 55, 58, 63, 66, 69, 72, 74, 78, 81, 84, 87, 90, 93, 96, 99, 102, 32,  
 33, 33, 33, 34, 36, 36, 38, 40, 41, 42, 44, 47, 48, 51, 55, 57, 60, 65,  
 68, 71, 73, 76, 78, 80, 82, 85, 88, 91, 95, 98, 102, 34, 34, 33, 34, 35,  
 37, 38, 39, 42, 44, 45, 47, 50, 51, 54, 58, 60, 63, 68, 71, 74, 76, 79,  
 82, 85, 86, 87, 88, 90, 93, 96, 99, 36, 35, 34, 35, 36, 38, 40, 42, 48,  
 50, 50, 54, 56, 57, 60, 64, 65, 68, 73, 76, 79, 81, 84, 86, 86, 88, 90,  
 93, 95, 97, 98, 100, 38, 37, 36, 36, 38, 39, 41, 44, 50, 51, 52, 56, 58,  
 60, 63, 67, 68, 71, 76, 79, 82, 84, 87, 87, 90, 93, 94, 95, 96, 100,  
 103, 106, 39, 38, 37, 38, 39, 40, 42, 45, 50, 52, 54, 58, 60, 62, 65,  
 69, 70, 73, 78, 81, 84, 86, 89, 92, 95, 95, 96, 99, 102, 104, 105, 106,  
 44, 42, 41, 41, 42, 42, 44, 47, 54, 56, 58, 63, 66, 68, 71, 75, 77, 79,  
 84, 88, 90, 92, 95, 97, 97, 99, 102, 103, 103, 106, 109, 113, 47, 45,  
 44, 44, 45, 45, 47, 50, 56, 58, 60, 66, 69, 71, 75, 79, 81, 84, 89, 92,  
 95, 97, 100, 100, 102, 105, 104, 106, 109, 111, 112, 113, 49, 47, 46,  
 45, 46, 46, 48, 51, 57, 60, 62, 68, 71, 73, 77, 81, 83, 87, 92, 95, 98,  
 100, 103, 105, 107, 106, 109, 112, 112, 113, 117, 120, 54, 51, 50, 49,  
 50, 49, 51, 54, 60, 63, 65, 71, 75, 77, 82, 87, 89, 92, 97, 101, 104,  
 106, 109, 112, 110, 113, 114, 114, 117, 121, 121, 121, 59, 56, 54, 54,  
 54, 53, 55, 58, 64, 67, 69, 75, 79, 81, 87, 92, 94, 98, 103, 107, 110,  
 113, 116, 114, 117, 118, 117, 121, 122, 122, 125, 129, 61, 58, 56, 56,  
 56, 55, 57, 60, 65, 68, 70, 77, 81, 83, 89, 94, 97, 101, 106, 110, 113,  
 116, 119, 120, 122, 121, 126, 124, 125, 130, 130, 130, 65, 62, 60, 59,  
 59, 58, 60, 63, 68, 71, 73, 79, 84, 87, 92, 98, 101, 105, 111, 115, 118,  
 121, 124, 128, 125, 129, 128, 131, 133, 132, 135, 139, 71, 68, 65, 65,

```

64, 63, 65, 68, 73, 76, 78, 84, 89, 92, 97, 103, 106, 111, 117, 122,
125, 128, 131, 131, 134, 132, 134, 136, 136, 140, 141, 140, 76, 72, 70,
69, 68, 66, 68, 71, 76, 79, 81, 88, 92, 95, 101, 107, 110, 115, 122,
127, 130, 133, 136, 136, 138, 139, 141, 140, 145, 143, 146, 151, 80, 76,
73, 72, 71, 69, 71, 74, 79, 82, 84, 90, 95, 98, 104, 110, 113, 118, 125,
130, 134, 137, 140, 146, 142, 146, 145, 149, 146, 150, 152, 151, 83, 78,
76, 75, 74, 72, 73, 76, 81, 84, 86, 92, 97, 100, 106, 113, 116, 121,
128, 133, 137, 140, 144, 147, 152, 148, 154, 151, 156, 155, 156, 162,
86, 82, 79, 78, 77, 74, 76, 79, 84, 87, 89, 95, 100, 103, 109, 116, 119,
124, 131, 136, 140, 144, 147, 150, 154, 159, 157, 160, 160, 162, 165,
162, 89, 85, 82, 81, 79, 78, 78, 82, 86, 87, 92, 97, 100, 105, 112, 114,
120, 128, 131, 136, 146, 147, 150, 155, 156, 161, 166, 165, 167, 169,
169, 175, 92, 88, 85, 84, 82, 81, 80, 85, 86, 90, 95, 97, 102, 107, 110,
117, 122, 125, 134, 138, 142, 152, 154, 156, 162, 163, 168, 173, 174,
174, 178, 176, 95, 90, 88, 86, 85, 84, 82, 86, 88, 93, 95, 99, 105, 106,
113, 118, 121, 129, 132, 139, 146, 148, 159, 161, 163, 169, 170, 176,
180, 183, 181, 187, 98, 93, 91, 89, 88, 87, 85, 87, 90, 94, 96, 102,
104, 109, 114, 117, 126, 128, 134, 141, 145, 154, 157, 166, 168, 170,
176, 178, 184, 188, 193, 188, 101, 96, 95, 92, 91, 90, 88, 88, 93, 95,
99, 103, 106, 112, 114, 121, 124, 131, 136, 140, 149, 151, 160, 165,
173, 176, 178, 184, 186, 192, 196, 203, 104, 99, 98, 95, 94, 93, 91, 90,
95, 96, 102, 103, 109, 112, 117, 122, 125, 133, 136, 145, 146, 156, 160,
167, 174, 180, 184, 186, 193, 194, 201, 204, 108, 102, 101, 98, 97, 96,
95, 93, 97, 100, 104, 106, 111, 113, 121, 122, 130, 132, 140, 143, 150,
155, 162, 169, 174, 183, 188, 192, 194, 201, 202, 210, 111, 105, 105,
101, 100, 99, 98, 96, 98, 103, 105, 109, 112, 117, 121, 125, 130, 135,
141, 146, 152, 156, 165, 169, 178, 181, 193, 196, 201, 202, 210, 211,
114, 109, 109, 104, 104, 102, 102, 99, 100, 106, 106, 113, 113, 120,
121, 129, 130, 139, 140, 151, 151, 162, 162, 175, 176, 187, 188, 203,
204, 210, 211, 219,
/* Size 4x8 */
32, 42, 69, 88, 33, 42, 64, 83, 36, 56, 77, 88, 46, 67, 93, 105, 60, 79,
112, 122, 75, 92, 130, 144, 86, 95, 136, 167, 98, 105, 136, 177,
/* Size 8x4 */
32, 33, 36, 46, 60, 75, 86, 98, 42, 42, 56, 67, 79, 92, 95, 105, 69, 64,
77, 93, 112, 130, 136, 136, 88, 83, 88, 105, 122, 144, 167, 177,
/* Size 8x16 */
32, 32, 36, 47, 65, 79, 90, 96, 31, 32, 35, 44, 60, 72, 84, 90, 32, 34,
36, 45, 59, 71, 80, 87, 32, 35, 40, 47, 60, 71, 78, 85, 36, 37, 48, 56,
68, 78, 83, 87, 39, 40, 50, 60, 73, 84, 91, 94, 47, 45, 56, 69, 84, 95,
101, 101, 53, 50, 60, 75, 92, 103, 108, 110, 61, 56, 65, 81, 100, 113,
116, 118, 71, 64, 73, 89, 111, 125, 129, 129, 79, 70, 79, 95, 118, 133,
142, 138, 86, 76, 84, 100, 124, 140, 153, 150, 92, 82, 89, 101, 121,
148, 157, 161, 98, 88, 93, 108, 124, 141, 163, 174, 104, 94, 95, 110,
129, 151, 171, 181, 110, 100, 98, 111, 127, 147, 169, 188,
/* Size 16x8 */
32, 31, 32, 32, 36, 39, 47, 53, 61, 71, 79, 86, 92, 98, 104, 110, 32,
32, 34, 35, 37, 40, 45, 50, 56, 64, 70, 76, 82, 88, 94, 100, 36, 35, 36,
40, 48, 50, 56, 60, 65, 73, 79, 84, 89, 93, 95, 98, 47, 44, 45, 47, 56,
60, 69, 75, 81, 89, 95, 100, 101, 108, 110, 111, 65, 60, 59, 60, 68, 73,

```

```

84, 92, 100, 111, 118, 124, 121, 124, 129, 127, 79, 72, 71, 71, 78, 84,
95, 103, 113, 125, 133, 140, 148, 141, 151, 147, 90, 84, 80, 78, 83, 91,
101, 108, 116, 129, 142, 153, 157, 163, 171, 169, 96, 90, 87, 85, 87,
94, 101, 110, 118, 129, 138, 150, 161, 174, 181, 188,
/* Size 16x32 */
32, 31, 32, 32, 36, 44, 47, 53, 65, 73, 79, 87, 90, 93, 96, 99, 31, 32,
32, 33, 35, 42, 45, 51, 62, 69, 75, 83, 86, 88, 91, 94, 31, 32, 32, 33,
35, 41, 44, 49, 60, 67, 72, 80, 84, 87, 90, 94, 31, 32, 33, 33, 35, 41,
44, 49, 59, 66, 71, 79, 82, 84, 87, 90, 32, 32, 34, 34, 36, 42, 45, 50,
59, 65, 71, 78, 80, 83, 87, 90, 32, 33, 35, 36, 38, 42, 45, 49, 58, 64,
69, 76, 80, 83, 86, 88, 32, 33, 35, 36, 40, 44, 47, 51, 60, 66, 71, 76,
78, 81, 85, 89, 34, 34, 36, 38, 42, 48, 50, 54, 63, 69, 73, 80, 82, 81,
84, 86, 36, 34, 37, 40, 48, 54, 56, 60, 68, 74, 78, 84, 83, 86, 87, 87,
38, 36, 39, 41, 49, 56, 58, 63, 71, 77, 81, 86, 88, 88, 90, 93, 39, 37,
40, 42, 50, 58, 60, 65, 73, 79, 84, 90, 91, 92, 94, 93, 44, 41, 42, 45,
53, 63, 66, 71, 79, 85, 90, 96, 94, 96, 96, 99, 47, 44, 45, 47, 56, 66,
69, 75, 84, 90, 95, 99, 101, 98, 101, 99, 49, 46, 47, 48, 57, 67, 71,
77, 86, 93, 97, 103, 103, 105, 102, 106, 53, 49, 50, 51, 60, 71, 75, 82,
92, 99, 103, 111, 108, 107, 110, 107, 58, 54, 54, 55, 63, 75, 79, 87,
98, 105, 110, 114, 114, 113, 111, 115, 61, 56, 56, 57, 65, 77, 81, 89,
100, 107, 113, 118, 116, 117, 118, 116, 65, 60, 59, 60, 68, 79, 84, 92,
105, 112, 118, 126, 124, 122, 121, 124, 71, 65, 64, 65, 73, 84, 89, 97,
111, 119, 125, 130, 129, 129, 129, 125, 76, 69, 68, 69, 76, 88, 92, 101,
115, 123, 130, 134, 134, 131, 132, 135, 79, 72, 70, 71, 79, 90, 95, 104,
118, 127, 133, 143, 142, 141, 138, 136, 82, 75, 73, 74, 81, 92, 97, 106,
121, 130, 136, 146, 145, 144, 144, 145, 86, 78, 76, 77, 84, 95, 100,
109, 124, 133, 140, 147, 153, 151, 150, 146, 89, 81, 79, 78, 87, 95, 99,
112, 124, 130, 145, 152, 156, 157, 156, 158, 92, 84, 82, 80, 89, 95,
101, 116, 121, 132, 148, 151, 157, 163, 161, 159, 95, 86, 85, 83, 92,
95, 105, 114, 120, 136, 143, 155, 163, 167, 171, 170, 98, 89, 88, 85,
93, 95, 108, 113, 124, 136, 141, 160, 163, 169, 174, 171, 101, 92, 91,
88, 94, 98, 110, 112, 128, 133, 146, 158, 166, 175, 179, 185, 104, 95,
94, 91, 95, 101, 110, 115, 129, 132, 151, 154, 171, 175, 181, 186, 107,
98, 97, 94, 96, 105, 110, 119, 128, 136, 149, 156, 173, 177, 188, 192,
110, 101, 100, 97, 98, 108, 111, 123, 127, 141, 147, 161, 169, 183, 188,
193, 114, 104, 104, 100, 100, 111, 111, 126, 127, 145, 145, 166, 166,
189, 190, 201,
/* Size 32x16 */
32, 31, 31, 31, 32, 32, 32, 34, 36, 38, 39, 44, 47, 49, 53, 58, 61, 65,
71, 76, 79, 82, 86, 89, 92, 95, 98, 101, 104, 107, 110, 114, 31, 32, 32,
32, 32, 33, 33, 34, 34, 36, 37, 41, 44, 46, 49, 54, 56, 60, 65, 69, 72,
75, 78, 81, 84, 86, 89, 92, 95, 98, 101, 104, 32, 32, 32, 33, 34, 35,
35, 36, 37, 39, 40, 42, 45, 47, 50, 54, 56, 59, 64, 68, 70, 73, 76, 79,
82, 85, 88, 91, 94, 97, 100, 104, 32, 33, 33, 33, 34, 36, 36, 38, 40,
41, 42, 45, 47, 48, 51, 55, 57, 60, 65, 69, 71, 74, 77, 78, 80, 83, 85,
88, 91, 94, 97, 100, 36, 35, 35, 35, 36, 38, 40, 42, 48, 49, 50, 53, 56,
57, 60, 63, 65, 68, 73, 76, 79, 81, 84, 87, 89, 92, 93, 94, 95, 96, 98,
100, 44, 42, 41, 41, 42, 42, 44, 48, 54, 56, 58, 63, 66, 67, 71, 75, 77,
79, 84, 88, 90, 92, 95, 95, 95, 95, 95, 98, 101, 105, 108, 111, 47, 45,
44, 44, 45, 45, 47, 50, 56, 58, 60, 66, 69, 71, 75, 79, 81, 84, 89, 92,

```

```

95, 97, 100, 99, 101, 105, 108, 110, 110, 110, 111, 111, 53, 51, 49, 49,
50, 49, 51, 54, 60, 63, 65, 71, 75, 77, 82, 87, 89, 92, 97, 101, 104,
106, 109, 112, 116, 114, 113, 112, 115, 119, 123, 126, 65, 62, 60, 59,
59, 58, 60, 63, 68, 71, 73, 79, 84, 86, 92, 98, 100, 105, 111, 115, 118,
121, 124, 124, 121, 120, 124, 128, 129, 128, 127, 127, 73, 69, 67, 66,
65, 64, 66, 69, 74, 77, 79, 85, 90, 93, 99, 105, 107, 112, 119, 123,
127, 130, 133, 130, 132, 136, 136, 133, 132, 136, 141, 145, 79, 75, 72,
71, 71, 69, 71, 73, 78, 81, 84, 90, 95, 97, 103, 110, 113, 118, 125,
130, 133, 136, 140, 145, 148, 143, 141, 146, 151, 149, 147, 145, 87, 83,
80, 79, 78, 76, 76, 80, 84, 86, 90, 96, 99, 103, 111, 114, 118, 126,
130, 134, 143, 146, 147, 152, 151, 155, 160, 158, 154, 156, 161, 166,
90, 86, 84, 82, 80, 80, 78, 82, 83, 88, 91, 94, 101, 103, 108, 114, 116,
124, 129, 134, 142, 145, 153, 156, 157, 163, 163, 166, 171, 173, 169,
166, 93, 88, 87, 84, 83, 83, 81, 81, 86, 88, 92, 96, 98, 105, 107, 113,
117, 122, 129, 131, 141, 144, 151, 157, 163, 167, 169, 175, 175, 177,
183, 189, 96, 91, 90, 87, 87, 86, 85, 84, 87, 90, 94, 96, 101, 102, 110,
111, 118, 121, 129, 132, 138, 144, 150, 156, 161, 171, 174, 179, 181,
188, 188, 190, 99, 94, 94, 90, 90, 88, 89, 86, 87, 93, 93, 99, 99, 106,
107, 115, 116, 124, 125, 135, 136, 145, 146, 158, 159, 170, 171, 185,
186, 192, 193, 201,
/* Size 4x16 */
31, 44, 73, 93, 32, 41, 67, 87, 32, 42, 65, 83, 33, 44, 66, 81, 34, 54,
74, 86, 37, 58, 79, 92, 44, 66, 90, 98, 49, 71, 99, 107, 56, 77, 107,
117, 65, 84, 119, 129, 72, 90, 127, 141, 78, 95, 133, 151, 84, 95, 132,
163, 89, 95, 136, 169, 95, 101, 132, 175, 101, 108, 141, 183,
/* Size 16x4 */
31, 32, 32, 33, 34, 37, 44, 49, 56, 65, 72, 78, 84, 89, 95, 101, 44, 41,
42, 44, 54, 58, 66, 71, 77, 84, 90, 95, 95, 95, 101, 108, 73, 67, 65,
66, 74, 79, 90, 99, 107, 119, 127, 133, 132, 136, 132, 141, 93, 87, 83,
81, 86, 92, 98, 107, 117, 129, 141, 151, 163, 169, 175, 183,
/* Size 8x32 */
32, 32, 36, 47, 65, 79, 90, 96, 31, 32, 35, 45, 62, 75, 86, 91, 31, 32,
35, 44, 60, 72, 84, 90, 31, 33, 35, 44, 59, 71, 82, 87, 32, 34, 36, 45,
59, 71, 80, 87, 32, 35, 38, 45, 58, 69, 80, 86, 32, 35, 40, 47, 60, 71,
78, 85, 34, 36, 42, 50, 63, 73, 82, 84, 36, 37, 48, 56, 68, 78, 83, 87,
38, 39, 49, 58, 71, 81, 88, 90, 39, 40, 50, 60, 73, 84, 91, 94, 44, 42,
53, 66, 79, 90, 94, 96, 47, 45, 56, 69, 84, 95, 101, 101, 49, 47, 57,
71, 86, 97, 103, 102, 53, 50, 60, 75, 92, 103, 108, 110, 58, 54, 63, 79,
98, 110, 114, 111, 61, 56, 65, 81, 100, 113, 116, 118, 65, 59, 68, 84,
105, 118, 124, 121, 71, 64, 73, 89, 111, 125, 129, 129, 76, 68, 76, 92,
115, 130, 134, 132, 79, 70, 79, 95, 118, 133, 142, 138, 82, 73, 81, 97,
121, 136, 145, 144, 86, 76, 84, 100, 124, 140, 153, 150, 89, 79, 87, 99,
124, 145, 156, 156, 92, 82, 89, 101, 121, 148, 157, 161, 95, 85, 92,
105, 120, 143, 163, 171, 98, 88, 93, 108, 124, 141, 163, 174, 101, 91,
94, 110, 128, 146, 166, 179, 104, 94, 95, 110, 129, 151, 171, 181, 107,
97, 96, 110, 128, 149, 173, 188, 110, 100, 98, 111, 127, 147, 169, 188,
114, 104, 100, 111, 127, 145, 166, 190,
/* Size 32x8 */
32, 31, 31, 31, 32, 32, 32, 34, 36, 38, 39, 44, 47, 49, 53, 58, 61, 65,
71, 76, 79, 82, 86, 89, 92, 95, 98, 101, 104, 107, 110, 114, 32, 32, 32,

```

```

33, 34, 35, 35, 36, 37, 39, 40, 42, 45, 47, 50, 54, 56, 59, 64, 68, 70,
73, 76, 79, 82, 85, 88, 91, 94, 97, 100, 104, 36, 35, 35, 35, 36, 38,
40, 42, 48, 49, 50, 53, 56, 57, 60, 63, 65, 68, 73, 76, 79, 81, 84, 87,
89, 92, 93, 94, 95, 96, 98, 100, 47, 45, 44, 44, 45, 45, 47, 50, 56, 58,
60, 66, 69, 71, 75, 79, 81, 84, 89, 92, 95, 97, 100, 99, 101, 105, 108,
110, 110, 110, 111, 111, 65, 62, 60, 59, 59, 58, 60, 63, 68, 71, 73, 79,
84, 86, 92, 98, 100, 105, 111, 115, 118, 121, 124, 124, 121, 120, 124,
128, 129, 128, 127, 127, 79, 75, 72, 71, 71, 69, 71, 73, 78, 81, 84, 90,
95, 97, 103, 110, 113, 118, 125, 130, 133, 136, 140, 145, 148, 143, 141,
146, 151, 149, 147, 145, 90, 86, 84, 82, 80, 80, 78, 82, 83, 88, 91, 94,
101, 103, 108, 114, 116, 124, 129, 134, 142, 145, 153, 156, 157, 163,
163, 166, 171, 173, 169, 166, 96, 91, 90, 87, 87, 86, 85, 84, 87, 90,
94, 96, 101, 102, 110, 111, 118, 121, 129, 132, 138, 144, 150, 156, 161,
171, 174, 179, 181, 188, 188, 190 },
{ /* Chroma */
  /* Size 4x4 */
  33, 45, 56, 64, 45, 58, 66, 69, 56, 66, 86, 87, 64, 69, 87, 105,
  /* Size 8x8 */
  31, 38, 47, 48, 54, 61, 66, 69, 38, 47, 47, 46, 50, 55, 61, 65, 47, 47,
  53, 55, 58, 63, 65, 66, 48, 46, 55, 62, 67, 72, 73, 73, 54, 50, 58, 67,
  76, 83, 84, 82, 61, 55, 63, 72, 83, 91, 92, 92, 66, 61, 65, 73, 84, 92,
  101, 103, 69, 65, 66, 73, 82, 92, 103, 109,
  /* Size 16x16 */
  32, 30, 33, 38, 49, 48, 50, 52, 55, 60, 63, 66, 68, 70, 72, 74, 30, 31,
  35, 41, 46, 46, 46, 48, 51, 55, 58, 60, 63, 65, 68, 70, 33, 35, 39, 44,
  47, 46, 46, 47, 50, 53, 56, 58, 60, 62, 65, 67, 38, 41, 44, 47, 49, 48,
  47, 48, 50, 53, 55, 58, 58, 60, 62, 65, 49, 46, 47, 49, 53, 53, 54, 54,
  56, 58, 60, 62, 62, 63, 64, 64, 48, 46, 46, 48, 53, 54, 56, 57, 59, 61,
  63, 65, 67, 66, 68, 68, 50, 46, 46, 47, 54, 56, 61, 63, 65, 68, 70, 72,
  71, 71, 72, 72, 52, 48, 47, 48, 54, 57, 63, 66, 69, 72, 75, 76, 75, 76,
  76, 76, 55, 51, 50, 50, 56, 59, 65, 69, 73, 77, 79, 81, 81, 81, 80, 80,
  60, 55, 53, 53, 58, 61, 68, 72, 77, 82, 85, 87, 87, 85, 84, 85, 63, 58,
  56, 55, 60, 63, 70, 75, 79, 85, 89, 91, 91, 90, 89, 90, 66, 60, 58, 58,
  62, 65, 72, 76, 81, 87, 91, 94, 96, 95, 95, 95, 68, 63, 60, 58, 62, 67,
  71, 75, 81, 87, 91, 96, 99, 100, 100, 100, 70, 65, 62, 60, 63, 66, 71,
  76, 81, 85, 90, 95, 100, 103, 104, 105, 72, 68, 65, 62, 64, 68, 72, 76,
  80, 84, 89, 95, 100, 104, 107, 108, 74, 70, 67, 65, 64, 68, 72, 76, 80,
  85, 90, 95, 100, 105, 108, 111,
  /* Size 32x32 */
  32, 31, 30, 31, 33, 36, 38, 41, 49, 49, 48, 49, 50, 51, 52, 54, 55, 57,
  60, 62, 63, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 31, 31, 31, 32,
  34, 38, 40, 42, 47, 47, 47, 47, 48, 48, 50, 52, 53, 54, 57, 59, 60, 61,
  63, 64, 65, 66, 67, 67, 68, 69, 70, 71, 30, 31, 31, 32, 35, 39, 41, 42,
  46, 46, 46, 45, 46, 47, 48, 50, 51, 52, 55, 57, 58, 59, 60, 62, 63, 64,
  65, 67, 68, 69, 70, 71, 31, 32, 32, 33, 36, 40, 41, 43, 46, 46, 45, 45,
  46, 46, 47, 49, 50, 51, 54, 56, 57, 58, 59, 61, 62, 63, 63, 64, 65, 66,
  67, 68, 33, 34, 35, 36, 39, 43, 44, 45, 47, 46, 46, 45, 46, 47, 47, 49,
  50, 51, 53, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68, 36, 38,
  39, 40, 43, 47, 47, 47, 48, 47, 46, 45, 46, 46, 47, 48, 49, 50, 52, 53,
  54, 55, 56, 58, 59, 61, 62, 63, 64, 65, 66, 66, 38, 40, 41, 41, 44, 47,

```

```

47, 48, 49, 48, 48, 47, 47, 47, 48, 49, 50, 51, 53, 54, 55, 56, 58, 58,
58, 59, 60, 61, 62, 64, 65, 66, 41, 42, 42, 43, 45, 47, 48, 48, 50, 50,
49, 49, 50, 50, 50, 52, 52, 53, 55, 56, 57, 58, 59, 60, 61, 61, 61, 61,
62, 63, 63, 64, 49, 47, 46, 46, 47, 48, 49, 50, 53, 53, 53, 53, 54, 54,
54, 55, 56, 56, 58, 59, 60, 61, 62, 63, 62, 62, 63, 64, 64, 64, 64, 64,
49, 47, 46, 46, 46, 47, 48, 50, 53, 53, 54, 55, 55, 55, 56, 57, 58, 58,
60, 61, 62, 63, 64, 64, 64, 65, 65, 65, 65, 66, 67, 68, 48, 47, 46, 45,
46, 46, 48, 49, 53, 54, 54, 55, 56, 56, 57, 58, 59, 60, 61, 63, 63, 64,
65, 66, 67, 66, 66, 67, 68, 68, 68, 68, 49, 47, 45, 45, 45, 45, 47, 49,
53, 55, 55, 58, 59, 60, 61, 62, 63, 63, 65, 66, 67, 68, 69, 69, 68, 68,
69, 69, 69, 69, 70, 71, 50, 48, 46, 46, 46, 46, 47, 50, 54, 55, 56, 59,
61, 61, 63, 64, 65, 66, 68, 69, 70, 71, 72, 71, 71, 72, 71, 71, 72, 72,
72, 71, 51, 48, 47, 46, 47, 46, 47, 50, 54, 55, 56, 60, 61, 62, 64, 66,
66, 67, 69, 70, 71, 72, 73, 73, 74, 73, 73, 74, 73, 73, 74, 75, 52, 50,
48, 47, 47, 47, 48, 50, 54, 56, 57, 61, 63, 64, 66, 68, 69, 70, 72, 74,
75, 75, 76, 77, 75, 76, 76, 75, 76, 77, 76, 75, 54, 52, 50, 49, 49, 48,
49, 52, 55, 57, 58, 62, 64, 66, 68, 71, 72, 73, 75, 77, 78, 79, 80, 78,
79, 78, 77, 78, 78, 77, 78, 79, 55, 53, 51, 50, 50, 49, 50, 52, 56, 58,
59, 63, 65, 66, 69, 72, 73, 74, 77, 78, 79, 80, 81, 81, 81, 80, 81, 80,
80, 81, 80, 79, 57, 54, 52, 51, 51, 50, 51, 53, 56, 58, 60, 63, 66, 67,
70, 73, 74, 76, 79, 80, 82, 83, 84, 85, 83, 84, 83, 83, 83, 82, 82, 83,
60, 57, 55, 54, 53, 52, 53, 55, 58, 60, 61, 65, 68, 69, 72, 75, 77, 79,
82, 84, 85, 86, 87, 86, 87, 85, 85, 85, 84, 86, 85, 84, 62, 59, 57, 56,
55, 53, 54, 56, 59, 61, 63, 66, 69, 70, 74, 77, 78, 80, 84, 86, 87, 88,
90, 89, 89, 88, 88, 87, 88, 87, 87, 88, 63, 60, 58, 57, 56, 54, 55, 57,
60, 62, 63, 67, 70, 71, 75, 78, 79, 82, 85, 87, 89, 90, 91, 93, 91, 91,
90, 91, 89, 90, 90, 89, 65, 61, 59, 58, 57, 55, 56, 58, 61, 63, 64, 68,
71, 72, 75, 79, 80, 83, 86, 88, 90, 91, 93, 94, 95, 92, 94, 92, 93, 92,
91, 93, 66, 63, 60, 59, 58, 56, 58, 59, 62, 64, 65, 69, 72, 73, 76, 80,
81, 84, 87, 90, 91, 93, 94, 95, 96, 97, 95, 95, 95, 95, 95, 93, 67, 64,
62, 61, 59, 58, 58, 60, 63, 64, 66, 69, 71, 73, 77, 78, 81, 85, 86, 89,
93, 94, 95, 97, 97, 98, 99, 97, 97, 97, 96, 98, 68, 65, 63, 62, 60, 59,
58, 61, 62, 64, 67, 68, 71, 74, 75, 79, 81, 83, 87, 89, 91, 95, 96, 97,
99, 98, 100, 100, 100, 99, 100, 98, 69, 66, 64, 63, 61, 61, 59, 61, 62,
65, 66, 68, 72, 73, 76, 78, 80, 84, 85, 88, 91, 92, 97, 98, 98, 101,
100, 102, 102, 103, 101, 102, 70, 67, 65, 63, 62, 62, 60, 61, 63, 65,
66, 69, 71, 73, 76, 77, 81, 83, 85, 88, 90, 94, 95, 99, 100, 100, 103,
102, 104, 104, 105, 103, 71, 67, 67, 64, 63, 63, 61, 61, 64, 65, 67, 69,
71, 74, 75, 78, 80, 83, 85, 87, 91, 92, 95, 97, 100, 102, 102, 105, 104,
106, 106, 108, 72, 68, 68, 65, 65, 64, 62, 62, 64, 65, 68, 69, 72, 73,
76, 78, 80, 83, 84, 88, 89, 93, 95, 97, 100, 102, 104, 104, 107, 106,
108, 108, 73, 69, 69, 66, 66, 65, 64, 63, 64, 66, 68, 69, 72, 73, 77,
77, 81, 82, 86, 87, 90, 92, 95, 97, 99, 103, 104, 106, 106, 109, 108,
110, 74, 70, 70, 67, 67, 66, 65, 63, 64, 67, 68, 70, 72, 74, 76, 78, 80,
82, 85, 87, 90, 91, 95, 96, 100, 101, 105, 106, 108, 108, 111, 110, 75,
71, 71, 68, 68, 66, 66, 64, 64, 68, 68, 71, 71, 75, 75, 79, 79, 83, 84,
88, 89, 93, 93, 98, 98, 102, 103, 108, 108, 110, 110, 113,
/* Size 4x8 */
31, 47, 57, 65, 40, 45, 52, 61, 46, 55, 61, 63, 47, 60, 70, 72, 52, 64,
79, 81, 59, 68, 87, 90, 63, 66, 88, 99, 66, 69, 85, 102,

```

```
/* Size 8x4 */
```

```
31, 40, 46, 47, 52, 59, 63, 66, 47, 45, 55, 60, 64, 68, 66, 69, 57, 52,  
61, 70, 79, 87, 88, 85, 65, 61, 63, 72, 81, 90, 99, 102,
```

```
/* Size 8x16 */
```

```
32, 35, 48, 50, 57, 63, 68, 70, 30, 38, 46, 46, 52, 58, 63, 65, 33, 41,  
47, 46, 51, 56, 60, 63, 39, 46, 48, 47, 51, 55, 58, 61, 49, 48, 53, 54,  
57, 60, 61, 61, 48, 46, 53, 56, 60, 64, 65, 65, 50, 46, 54, 61, 66, 70,  
71, 69, 52, 47, 54, 63, 71, 75, 75, 74, 55, 49, 56, 65, 74, 79, 79, 78,  
60, 53, 58, 68, 79, 85, 85, 82, 63, 55, 60, 70, 82, 89, 91, 87, 66, 58,  
62, 72, 84, 91, 95, 91, 68, 60, 64, 71, 81, 94, 97, 96, 70, 62, 65, 73,  
81, 89, 98, 101, 72, 65, 65, 72, 82, 92, 100, 103, 74, 67, 65, 71, 79,  
89, 98, 105,
```

```
/* Size 16x8 */
```

```
32, 30, 33, 39, 49, 48, 50, 52, 55, 60, 63, 66, 68, 70, 72, 74, 35, 38,  
41, 46, 48, 46, 46, 47, 49, 53, 55, 58, 60, 62, 65, 67, 48, 46, 47, 48,  
53, 53, 54, 54, 56, 58, 60, 62, 64, 65, 65, 65, 50, 46, 46, 47, 54, 56,  
61, 63, 65, 68, 70, 72, 71, 73, 72, 71, 57, 52, 51, 51, 57, 60, 66, 71,  
74, 79, 82, 84, 81, 81, 82, 79, 63, 58, 56, 55, 60, 64, 70, 75, 79, 85,  
89, 91, 94, 89, 92, 89, 68, 63, 60, 58, 61, 65, 71, 75, 79, 85, 91, 95,  
97, 98, 100, 98, 70, 65, 63, 61, 61, 65, 69, 74, 78, 82, 87, 91, 96,  
101, 103, 105,
```

```
/* Size 16x32 */
```

```
32, 31, 35, 38, 48, 49, 50, 52, 57, 61, 63, 67, 68, 69, 70, 71, 31, 31,  
37, 40, 47, 47, 48, 50, 54, 57, 60, 63, 64, 65, 66, 67, 30, 32, 38, 40,  
46, 45, 46, 48, 52, 55, 58, 61, 63, 64, 65, 67, 31, 33, 38, 41, 46, 45,  
46, 48, 52, 55, 57, 60, 61, 62, 63, 64, 33, 36, 41, 44, 47, 46, 46, 47,  
51, 54, 56, 59, 60, 61, 63, 64, 37, 40, 45, 47, 47, 45, 46, 47, 50, 52,  
54, 57, 59, 61, 62, 62, 39, 41, 46, 47, 48, 47, 47, 48, 51, 54, 55, 57,  
58, 59, 61, 62, 42, 43, 46, 48, 50, 49, 50, 50, 53, 56, 57, 60, 60, 59,  
60, 60, 49, 46, 48, 49, 53, 53, 54, 54, 57, 59, 60, 63, 61, 62, 61, 61,  
48, 46, 47, 48, 53, 55, 55, 56, 58, 61, 62, 64, 64, 63, 63, 64, 48, 46,  
46, 48, 53, 56, 56, 57, 60, 62, 64, 66, 65, 65, 65, 64, 49, 45, 45, 47,  
53, 58, 59, 61, 64, 66, 67, 69, 67, 67, 66, 67, 50, 46, 46, 48, 54, 59,  
61, 63, 66, 68, 70, 71, 71, 68, 69, 67, 51, 47, 47, 48, 54, 60, 61, 64,  
68, 70, 71, 73, 72, 72, 70, 71, 52, 48, 47, 48, 54, 61, 63, 66, 71, 73,  
75, 77, 75, 73, 74, 71, 54, 50, 49, 50, 55, 62, 65, 68, 73, 76, 78, 79,  
78, 76, 74, 75, 55, 51, 49, 50, 56, 63, 65, 69, 74, 77, 79, 81, 79, 78,  
78, 75, 57, 52, 50, 51, 56, 64, 66, 70, 76, 79, 82, 85, 83, 81, 79, 79,  
60, 54, 53, 53, 58, 65, 68, 72, 79, 82, 85, 87, 85, 84, 82, 80, 62, 56,  
54, 55, 60, 66, 69, 74, 81, 84, 87, 88, 87, 85, 84, 84, 63, 57, 55, 56,  
60, 67, 70, 75, 82, 86, 89, 92, 91, 89, 87, 84, 64, 59, 56, 57, 61, 68,  
71, 75, 83, 87, 90, 93, 92, 90, 89, 89, 66, 60, 58, 58, 62, 69, 72, 76,  
84, 88, 91, 94, 95, 93, 91, 89, 67, 61, 59, 58, 63, 68, 71, 78, 83, 86,  
93, 96, 96, 96, 94, 94, 68, 62, 60, 59, 64, 67, 71, 79, 81, 86, 94, 95,  
97, 98, 96, 94, 69, 63, 61, 60, 65, 66, 72, 77, 80, 88, 91, 96, 99, 99,  
100, 98, 70, 64, 62, 60, 65, 66, 73, 76, 81, 87, 89, 97, 98, 100, 101,  
99, 71, 65, 64, 61, 65, 67, 73, 74, 82, 85, 90, 95, 99, 102, 103, 104,  
72, 65, 65, 62, 65, 68, 72, 75, 82, 83, 92, 93, 100, 102, 103, 104, 73,  
66, 66, 63, 65, 69, 72, 76, 81, 85, 90, 93, 100, 102, 105, 106, 74, 67,  
67, 64, 65, 70, 71, 77, 79, 86, 89, 94, 98, 103, 105, 106, 75, 68, 68,
```

```

65, 65, 71, 71, 78, 78, 87, 87, 96, 96, 105, 105, 109,
/* Size 32x16 */
32, 31, 30, 31, 33, 37, 39, 42, 49, 48, 48, 49, 50, 51, 52, 54, 55, 57,
60, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 31, 31, 32, 33,
36, 40, 41, 43, 46, 46, 46, 45, 46, 47, 48, 50, 51, 52, 54, 56, 57, 59,
60, 61, 62, 63, 64, 65, 65, 66, 67, 68, 35, 37, 38, 38, 41, 45, 46, 46,
48, 47, 46, 45, 46, 47, 47, 49, 49, 50, 53, 54, 55, 56, 58, 59, 60, 61,
62, 64, 65, 66, 67, 68, 38, 40, 40, 41, 44, 47, 47, 48, 49, 48, 48, 47,
48, 48, 48, 50, 50, 51, 53, 55, 56, 57, 58, 58, 59, 60, 60, 61, 62, 63,
64, 65, 48, 47, 46, 46, 47, 47, 48, 50, 53, 53, 53, 53, 54, 54, 54, 55,
56, 56, 58, 60, 60, 61, 62, 63, 64, 65, 65, 65, 65, 65, 65, 65, 49, 47,
45, 45, 46, 45, 47, 49, 53, 55, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 68, 67, 66, 66, 67, 68, 69, 70, 71, 50, 48, 46, 46, 46, 46,
47, 50, 54, 55, 56, 59, 61, 61, 63, 65, 65, 66, 68, 69, 70, 71, 72, 71,
71, 72, 73, 73, 72, 72, 71, 71, 52, 50, 48, 48, 47, 47, 48, 50, 54, 56,
57, 61, 63, 64, 66, 68, 69, 70, 72, 74, 75, 75, 76, 78, 79, 77, 76, 74,
75, 76, 77, 78, 57, 54, 52, 52, 51, 50, 51, 53, 57, 58, 60, 64, 66, 68,
71, 73, 74, 76, 79, 81, 82, 83, 84, 83, 81, 80, 81, 82, 82, 81, 79, 78,
61, 57, 55, 55, 54, 52, 54, 56, 59, 61, 62, 66, 68, 70, 73, 76, 77, 79,
82, 84, 86, 87, 88, 86, 86, 88, 87, 85, 83, 85, 86, 87, 63, 60, 58, 57,
56, 54, 55, 57, 60, 62, 64, 67, 70, 71, 75, 78, 79, 82, 85, 87, 89, 90,
91, 93, 94, 91, 89, 90, 92, 90, 89, 87, 67, 63, 61, 60, 59, 57, 57, 60,
63, 64, 66, 69, 71, 73, 77, 79, 81, 85, 87, 88, 92, 93, 94, 96, 95, 96,
97, 95, 93, 93, 94, 96, 68, 64, 63, 61, 60, 59, 58, 60, 61, 64, 65, 67,
71, 72, 75, 78, 79, 83, 85, 87, 91, 92, 95, 96, 97, 99, 98, 99, 100,
100, 98, 96, 69, 65, 64, 62, 61, 61, 59, 59, 62, 63, 65, 67, 68, 72, 73,
76, 78, 81, 84, 85, 89, 90, 93, 96, 98, 99, 100, 102, 102, 102, 103,
105, 70, 66, 65, 63, 63, 62, 61, 60, 61, 63, 65, 66, 69, 70, 74, 74, 78,
79, 82, 84, 87, 89, 91, 94, 96, 100, 101, 103, 103, 105, 105, 105, 71,
67, 67, 64, 64, 62, 62, 60, 61, 64, 64, 67, 67, 71, 71, 75, 75, 79, 80,
84, 84, 89, 89, 94, 94, 98, 99, 104, 104, 106, 106, 109,
/* Size 4x16 */
31, 49, 61, 69, 32, 45, 55, 64, 36, 46, 54, 61, 41, 47, 54, 59, 46, 53,
59, 62, 46, 56, 62, 65, 46, 59, 68, 68, 48, 61, 73, 73, 51, 63, 77, 78,
54, 65, 82, 84, 57, 67, 86, 89, 60, 69, 88, 93, 62, 67, 86, 98, 64, 66,
87, 100, 65, 68, 83, 102, 67, 70, 86, 103,
/* Size 16x4 */
31, 32, 36, 41, 46, 46, 46, 48, 51, 54, 57, 60, 62, 64, 65, 67, 49, 45,
46, 47, 53, 56, 59, 61, 63, 65, 67, 69, 67, 66, 68, 70, 61, 55, 54, 54,
59, 62, 68, 73, 77, 82, 86, 88, 86, 87, 83, 86, 69, 64, 61, 59, 62, 65,
68, 73, 78, 84, 89, 93, 98, 100, 102, 103,
/* Size 8x32 */
32, 35, 48, 50, 57, 63, 68, 70, 31, 37, 47, 48, 54, 60, 64, 66, 30, 38,
46, 46, 52, 58, 63, 65, 31, 38, 46, 46, 52, 57, 61, 63, 33, 41, 47, 46,
51, 56, 60, 63, 37, 45, 47, 46, 50, 54, 59, 62, 39, 46, 48, 47, 51, 55,
58, 61, 42, 46, 50, 50, 53, 57, 60, 60, 49, 48, 53, 54, 57, 60, 61, 61,
48, 47, 53, 55, 58, 62, 64, 63, 48, 46, 53, 56, 60, 64, 65, 65, 49, 45,
53, 59, 64, 67, 67, 66, 50, 46, 54, 61, 66, 70, 71, 69, 51, 47, 54, 61,
68, 71, 72, 70, 52, 47, 54, 63, 71, 75, 75, 74, 54, 49, 55, 65, 73, 78,
78, 74, 55, 49, 56, 65, 74, 79, 79, 78, 57, 50, 56, 66, 76, 82, 83, 79,

```



```

60, 53, 58, 68, 79, 85, 85, 82, 62, 54, 60, 69, 81, 87, 87, 84, 63, 55,
60, 70, 82, 89, 91, 87, 64, 56, 61, 71, 83, 90, 92, 89, 66, 58, 62, 72,
84, 91, 95, 91, 67, 59, 63, 71, 83, 93, 96, 94, 68, 60, 64, 71, 81, 94,
97, 96, 69, 61, 65, 72, 80, 91, 99, 100, 70, 62, 65, 73, 81, 89, 98,
101, 71, 64, 65, 73, 82, 90, 99, 103, 72, 65, 65, 72, 82, 92, 100, 103,
73, 66, 65, 72, 81, 90, 100, 105, 74, 67, 65, 71, 79, 89, 98, 105, 75,
68, 65, 71, 78, 87, 96, 105,
/* Size 32x8 */
32, 31, 30, 31, 33, 37, 39, 42, 49, 48, 48, 49, 50, 51, 52, 54, 55, 57,
60, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 35, 37, 38, 38,
41, 45, 46, 46, 48, 47, 46, 45, 46, 47, 47, 49, 49, 50, 53, 54, 55, 56,
58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 48, 47, 46, 46, 47, 47, 48, 50,
53, 53, 53, 53, 54, 54, 54, 55, 56, 56, 58, 60, 60, 61, 62, 63, 64, 65,
65, 65, 65, 65, 65, 65, 50, 48, 46, 46, 46, 46, 47, 50, 54, 55, 56, 59,
61, 61, 63, 65, 65, 66, 68, 69, 70, 71, 72, 71, 71, 72, 73, 73, 72, 72,
71, 71, 57, 54, 52, 52, 51, 50, 51, 53, 57, 58, 60, 64, 66, 68, 71, 73,
74, 76, 79, 81, 82, 83, 84, 83, 81, 80, 81, 82, 82, 81, 79, 78, 63, 60,
58, 57, 56, 54, 55, 57, 60, 62, 64, 67, 70, 71, 75, 78, 79, 82, 85, 87,
89, 90, 91, 93, 94, 91, 89, 90, 92, 90, 89, 87, 68, 64, 63, 61, 60, 59,
58, 60, 61, 64, 65, 67, 71, 72, 75, 78, 79, 83, 85, 87, 91, 92, 95, 96,
97, 99, 98, 99, 100, 100, 98, 96, 70, 66, 65, 63, 63, 62, 61, 60, 61,
63, 65, 66, 69, 70, 74, 74, 78, 79, 82, 84, 87, 89, 91, 94, 96, 100,
101, 103, 103, 105, 105, 105 },
},
{
{ /* Luma */
/* Size 4x4 */
32, 38, 63, 86, 38, 56, 78, 97, 63, 78, 113, 130, 86, 97, 130, 169,
/* Size 8x8 */
32, 32, 35, 46, 57, 76, 85, 96, 32, 34, 37, 45, 54, 70, 79, 90, 35, 37,
48, 56, 64, 79, 87, 93, 46, 45, 56, 70, 80, 96, 100, 105, 57, 54, 64,
80, 93, 111, 121, 122, 76, 70, 79, 96, 111, 134, 138, 144, 85, 79, 87,
100, 121, 138, 156, 168, 96, 90, 93, 105, 122, 144, 168, 184,
/* Size 16x16 */
32, 31, 31, 32, 34, 39, 44, 49, 58, 65, 71, 81, 87, 93, 98, 104, 31, 32,
32, 32, 34, 38, 41, 46, 54, 60, 66, 75, 81, 86, 92, 98, 31, 32, 33, 34,
36, 39, 42, 46, 53, 59, 64, 73, 78, 83, 88, 94, 32, 32, 34, 35, 37, 40,
42, 46, 52, 58, 63, 71, 75, 80, 86, 92, 34, 34, 36, 37, 42, 47, 50, 53,
59, 65, 70, 77, 82, 85, 89, 92, 39, 38, 39, 40, 47, 54, 58, 62, 68, 73,
78, 85, 90, 90, 96, 98, 44, 41, 42, 42, 50, 58, 63, 68, 74, 79, 84, 91,
96, 98, 102, 104, 49, 46, 46, 46, 53, 62, 68, 73, 81, 87, 92, 99, 103,
107, 109, 112, 58, 54, 53, 52, 59, 68, 74, 81, 90, 97, 102, 110, 114,
118, 117, 121, 65, 60, 59, 58, 65, 73, 79, 87, 97, 105, 111, 120, 125,
125, 126, 130, 71, 66, 64, 63, 70, 78, 84, 92, 102, 111, 117, 127, 133,
134, 136, 141, 81, 75, 73, 71, 77, 85, 91, 99, 110, 120, 127, 137, 143,
145, 148, 152, 87, 81, 78, 75, 82, 90, 96, 103, 114, 125, 133, 143, 150,
156, 160, 163, 93, 86, 83, 80, 85, 90, 98, 107, 118, 125, 134, 145, 156,
163, 169, 177, 98, 92, 88, 86, 89, 96, 102, 109, 117, 126, 136, 148,
160, 169, 176, 184, 104, 98, 94, 92, 92, 98, 104, 112, 121, 130, 141,
152, 163, 177, 184, 191,

```

```
/* Size 32x32 */
```

```
32, 31, 31, 31, 31, 32, 32, 34, 34, 36, 39, 41, 44, 48, 49, 54, 58, 59,
65, 69, 71, 80, 81, 83, 87, 90, 93, 95, 98, 101, 104, 107, 31, 32, 32,
32, 32, 32, 32, 34, 34, 35, 38, 39, 42, 46, 47, 51, 55, 57, 62, 66, 68,
76, 77, 78, 83, 85, 88, 90, 93, 96, 99, 101, 31, 32, 32, 32, 32, 32, 32,
33, 34, 34, 38, 39, 41, 45, 46, 50, 54, 55, 60, 64, 66, 73, 75, 76, 81,
83, 86, 89, 92, 95, 98, 101, 31, 32, 32, 32, 32, 32, 32, 33, 34, 34, 37,
38, 41, 44, 45, 49, 53, 54, 59, 63, 65, 72, 74, 75, 79, 81, 84, 86, 89,
91, 94, 97, 31, 32, 32, 32, 33, 33, 34, 35, 36, 36, 39, 40, 42, 45, 46,
50, 53, 54, 59, 63, 64, 71, 73, 74, 78, 80, 83, 85, 88, 91, 94, 97, 32,
32, 32, 32, 33, 34, 34, 36, 36, 37, 40, 40, 42, 45, 46, 49, 53, 54, 58,
62, 63, 70, 72, 73, 77, 79, 82, 85, 87, 90, 92, 95, 32, 32, 32, 32, 34,
34, 35, 37, 37, 38, 40, 41, 42, 45, 46, 49, 52, 54, 58, 61, 63, 69, 71,
72, 75, 78, 80, 83, 86, 89, 92, 95, 34, 34, 33, 33, 35, 36, 37, 39, 41,
42, 45, 46, 47, 50, 51, 54, 57, 59, 63, 66, 68, 74, 75, 76, 80, 81, 82,
83, 85, 87, 90, 93, 34, 34, 34, 34, 36, 36, 37, 41, 42, 45, 47, 48, 50,
53, 53, 56, 59, 61, 65, 68, 70, 76, 77, 78, 82, 83, 85, 88, 89, 90, 92,
93, 36, 35, 34, 34, 36, 37, 38, 42, 45, 48, 50, 51, 54, 56, 57, 60, 63,
64, 68, 71, 73, 79, 80, 81, 85, 87, 89, 89, 90, 93, 96, 99, 39, 38, 38,
37, 39, 40, 40, 45, 47, 50, 54, 55, 58, 61, 62, 65, 68, 69, 73, 76, 78,
84, 85, 86, 90, 89, 90, 93, 96, 97, 98, 99, 41, 39, 39, 38, 40, 40, 41,
46, 48, 51, 55, 56, 59, 62, 63, 67, 70, 71, 75, 78, 80, 86, 87, 88, 91,
93, 96, 97, 97, 99, 102, 105, 44, 42, 41, 41, 42, 42, 42, 47, 50, 54,
58, 59, 63, 66, 68, 71, 74, 75, 79, 83, 84, 90, 91, 92, 96, 98, 98, 99,
102, 104, 104, 105, 48, 46, 45, 44, 45, 45, 45, 50, 53, 56, 61, 62, 66,
70, 71, 76, 79, 80, 85, 88, 90, 96, 97, 98, 101, 100, 102, 105, 105,
105, 109, 112, 49, 47, 46, 45, 46, 46, 46, 51, 53, 57, 62, 63, 68, 71,
73, 77, 81, 82, 87, 90, 92, 98, 99, 100, 103, 106, 107, 106, 109, 112,
112, 112, 54, 51, 50, 49, 50, 49, 49, 54, 56, 60, 65, 67, 71, 76, 77,
82, 86, 87, 92, 96, 97, 104, 105, 106, 110, 110, 109, 113, 114, 113,
116, 120, 58, 55, 54, 53, 53, 53, 52, 57, 59, 63, 68, 70, 74, 79, 81,
86, 90, 91, 97, 100, 102, 109, 110, 111, 114, 114, 118, 116, 117, 121,
121, 120, 59, 57, 55, 54, 54, 54, 54, 59, 61, 64, 69, 71, 75, 80, 82,
87, 91, 93, 99, 102, 104, 111, 112, 113, 117, 121, 120, 122, 124, 122,
125, 129, 65, 62, 60, 59, 59, 58, 58, 63, 65, 68, 73, 75, 79, 85, 87,
92, 97, 99, 105, 109, 111, 118, 120, 121, 125, 124, 125, 127, 126, 130,
130, 129, 69, 66, 64, 63, 63, 62, 61, 66, 68, 71, 76, 78, 83, 88, 90,
96, 100, 102, 109, 113, 115, 123, 125, 126, 129, 130, 131, 130, 134,
133, 135, 139, 71, 68, 66, 65, 64, 63, 63, 68, 70, 73, 78, 80, 84, 90,
92, 97, 102, 104, 111, 115, 117, 125, 127, 128, 133, 136, 134, 139, 136,
139, 141, 140, 80, 76, 73, 72, 71, 70, 69, 74, 76, 79, 84, 86, 90, 96,
98, 104, 109, 111, 118, 123, 125, 134, 136, 137, 142, 138, 143, 140,
144, 144, 144, 149, 81, 77, 75, 74, 73, 72, 71, 75, 77, 80, 85, 87, 91,
97, 99, 105, 110, 112, 120, 125, 127, 136, 137, 139, 143, 148, 145, 148,
148, 150, 152, 149, 83, 78, 76, 75, 74, 73, 72, 76, 78, 81, 86, 88, 92,
98, 100, 106, 111, 113, 121, 126, 128, 137, 139, 140, 145, 149, 153,
153, 154, 155, 155, 161, 87, 83, 81, 79, 78, 77, 75, 80, 82, 85, 90, 91,
96, 101, 103, 110, 114, 117, 125, 129, 133, 142, 143, 145, 150, 151,
156, 159, 160, 160, 163, 161, 90, 85, 83, 81, 80, 79, 78, 81, 83, 87,
89, 93, 98, 100, 106, 110, 114, 121, 124, 130, 136, 138, 148, 149, 151,
```

```

156, 157, 162, 166, 168, 166, 172, 93, 88, 86, 84, 83, 82, 80, 82, 85,
89, 90, 96, 98, 102, 107, 109, 118, 120, 125, 131, 134, 143, 145, 153,
156, 157, 163, 164, 169, 172, 177, 172, 95, 90, 89, 86, 85, 85, 83, 83,
88, 89, 93, 97, 99, 105, 106, 113, 116, 122, 127, 130, 139, 140, 148,
153, 159, 162, 164, 169, 170, 176, 179, 185, 98, 93, 92, 89, 88, 87, 86,
85, 89, 90, 96, 97, 102, 105, 109, 114, 117, 124, 126, 134, 136, 144,
148, 154, 160, 166, 169, 170, 176, 177, 184, 186, 101, 96, 95, 91, 91,
90, 89, 87, 90, 93, 97, 99, 104, 105, 112, 113, 121, 122, 130, 133, 139,
144, 150, 155, 160, 168, 172, 176, 177, 184, 185, 191, 104, 99, 98, 94,
94, 92, 92, 90, 92, 96, 98, 102, 104, 109, 112, 116, 121, 125, 130, 135,
141, 144, 152, 155, 163, 166, 177, 179, 184, 185, 191, 192, 107, 101,
101, 97, 97, 95, 95, 93, 93, 99, 99, 105, 105, 112, 112, 120, 120, 129,
129, 139, 140, 149, 149, 161, 161, 172, 172, 185, 186, 191, 192, 199,
/* Size 4x8 */
32, 38, 62, 86, 32, 40, 58, 80, 34, 51, 68, 85, 44, 61, 85, 101, 54, 69,
98, 117, 72, 84, 118, 136, 82, 89, 129, 157, 92, 98, 127, 165,
/* Size 8x4 */
32, 32, 34, 44, 54, 72, 82, 92, 38, 40, 51, 61, 69, 84, 89, 98, 62, 58,
68, 85, 98, 118, 129, 127, 86, 80, 85, 101, 117, 136, 157, 165,
/* Size 8x16 */
32, 32, 36, 44, 58, 79, 88, 93, 31, 32, 35, 41, 54, 73, 81, 88, 32, 33,
36, 42, 53, 71, 78, 84, 32, 34, 38, 42, 52, 69, 76, 82, 34, 36, 44, 50,
59, 75, 81, 84, 39, 39, 50, 58, 68, 84, 88, 90, 44, 42, 53, 63, 74, 90,
97, 97, 49, 46, 57, 67, 81, 97, 104, 105, 57, 53, 63, 74, 90, 108, 111,
113, 65, 59, 68, 79, 97, 118, 123, 122, 71, 64, 73, 84, 102, 125, 135,
131, 81, 72, 80, 91, 110, 135, 145, 141, 87, 77, 85, 96, 114, 140, 148,
151, 92, 83, 88, 102, 117, 133, 153, 163, 98, 88, 89, 103, 121, 141,
160, 169, 103, 94, 92, 103, 119, 137, 158, 175,
/* Size 16x8 */
32, 31, 32, 32, 34, 39, 44, 49, 57, 65, 71, 81, 87, 92, 98, 103, 32, 32,
33, 34, 36, 39, 42, 46, 53, 59, 64, 72, 77, 83, 88, 94, 36, 35, 36, 38,
44, 50, 53, 57, 63, 68, 73, 80, 85, 88, 89, 92, 44, 41, 42, 42, 50, 58,
63, 67, 74, 79, 84, 91, 96, 102, 103, 103, 58, 54, 53, 52, 59, 68, 74,
81, 90, 97, 102, 110, 114, 117, 121, 119, 79, 73, 71, 69, 75, 84, 90,
97, 108, 118, 125, 135, 140, 133, 141, 137, 88, 81, 78, 76, 81, 88, 97,
104, 111, 123, 135, 145, 148, 153, 160, 158, 93, 88, 84, 82, 84, 90, 97,
105, 113, 122, 131, 141, 151, 163, 169, 175,
/* Size 16x32 */
32, 31, 32, 32, 36, 39, 44, 53, 58, 65, 79, 81, 88, 90, 93, 96, 31, 32,
32, 32, 35, 38, 42, 51, 55, 62, 75, 77, 83, 86, 88, 91, 31, 32, 32, 32,
35, 38, 41, 50, 54, 60, 73, 75, 81, 84, 88, 91, 31, 32, 32, 33, 34, 37,
41, 49, 53, 59, 72, 74, 79, 82, 84, 87, 32, 32, 33, 34, 36, 39, 42, 50,
53, 59, 71, 72, 78, 81, 84, 87, 32, 32, 34, 34, 37, 40, 42, 49, 53, 58,
70, 71, 77, 80, 83, 85, 32, 33, 34, 35, 38, 40, 42, 49, 52, 58, 69, 70,
76, 78, 82, 86, 34, 34, 35, 37, 42, 45, 48, 54, 57, 63, 73, 75, 79, 79,
81, 83, 34, 34, 36, 37, 44, 47, 50, 56, 59, 65, 75, 77, 81, 83, 84, 84,
36, 34, 37, 38, 48, 51, 54, 60, 63, 68, 78, 80, 85, 85, 86, 89, 39, 37,
39, 40, 50, 54, 58, 65, 68, 73, 84, 85, 88, 89, 90, 89, 40, 38, 40, 41,
51, 55, 59, 67, 70, 75, 85, 87, 91, 92, 92, 95, 44, 41, 42, 43, 53, 58,
63, 71, 74, 79, 90, 91, 97, 94, 97, 95, 47, 44, 45, 46, 56, 61, 66, 75,

```

```

79, 85, 95, 97, 99, 101, 98, 102, 49, 46, 46, 47, 57, 62, 67, 77, 81,
86, 97, 99, 104, 102, 105, 102, 53, 49, 50, 50, 60, 65, 71, 82, 86, 92,
103, 105, 109, 108, 106, 110, 57, 53, 53, 53, 63, 68, 74, 86, 90, 97,
108, 110, 111, 112, 113, 110, 59, 54, 54, 54, 64, 69, 75, 87, 91, 98,
111, 112, 119, 117, 115, 118, 65, 60, 59, 58, 68, 73, 79, 92, 97, 105,
118, 119, 123, 123, 122, 119, 69, 63, 62, 62, 71, 76, 83, 96, 100, 109,
122, 124, 127, 125, 125, 128, 71, 65, 64, 63, 73, 78, 84, 97, 102, 111,
125, 127, 135, 134, 131, 129, 79, 72, 71, 70, 79, 84, 90, 104, 109, 118,
133, 135, 137, 136, 136, 137, 81, 74, 72, 71, 80, 85, 91, 105, 110, 120,
135, 137, 145, 143, 141, 138, 82, 75, 73, 72, 81, 86, 92, 106, 111, 121,
136, 139, 147, 148, 147, 149, 87, 79, 77, 76, 85, 90, 96, 110, 114, 125,
140, 143, 148, 154, 151, 149, 90, 82, 80, 78, 87, 89, 99, 108, 113, 129,
135, 146, 153, 157, 160, 159, 92, 84, 83, 81, 88, 90, 102, 106, 117,
128, 133, 150, 153, 158, 163, 160, 95, 87, 85, 83, 88, 92, 103, 105,
120, 125, 137, 148, 155, 164, 168, 173, 98, 89, 88, 85, 89, 95, 103,
108, 121, 124, 141, 144, 160, 164, 169, 174, 100, 92, 91, 88, 90, 98,
103, 111, 120, 127, 139, 146, 161, 165, 175, 179, 103, 94, 94, 90, 92,
101, 103, 114, 119, 131, 137, 150, 158, 170, 175, 180, 106, 97, 97, 93,
93, 104, 104, 118, 118, 135, 135, 154, 155, 175, 176, 187,
/* Size 32x16 */
32, 31, 31, 31, 32, 32, 32, 34, 34, 36, 39, 40, 44, 47, 49, 53, 57, 59,
65, 69, 71, 79, 81, 82, 87, 90, 92, 95, 98, 100, 103, 106, 31, 32, 32,
32, 32, 32, 33, 34, 34, 34, 37, 38, 41, 44, 46, 49, 53, 54, 60, 63, 65,
72, 74, 75, 79, 82, 84, 87, 89, 92, 94, 97, 32, 32, 32, 32, 33, 34, 34,
35, 36, 37, 39, 40, 42, 45, 46, 50, 53, 54, 59, 62, 64, 71, 72, 73, 77,
80, 83, 85, 88, 91, 94, 97, 32, 32, 32, 33, 34, 34, 35, 37, 37, 38, 40,
41, 43, 46, 47, 50, 53, 54, 58, 62, 63, 70, 71, 72, 76, 78, 81, 83, 85,
88, 90, 93, 36, 35, 35, 34, 36, 37, 38, 42, 44, 48, 50, 51, 53, 56, 57,
60, 63, 64, 68, 71, 73, 79, 80, 81, 85, 87, 88, 88, 89, 90, 92, 93, 39,
38, 38, 37, 39, 40, 40, 45, 47, 51, 54, 55, 58, 61, 62, 65, 68, 69, 73,
76, 78, 84, 85, 86, 90, 89, 90, 92, 95, 98, 101, 104, 44, 42, 41, 41,
42, 42, 42, 48, 50, 54, 58, 59, 63, 66, 67, 71, 74, 75, 79, 83, 84, 90,
91, 92, 96, 99, 102, 103, 103, 103, 103, 104, 53, 51, 50, 49, 50, 49,
49, 54, 56, 60, 65, 67, 71, 75, 77, 82, 86, 87, 92, 96, 97, 104, 105,
106, 110, 108, 106, 105, 108, 111, 114, 118, 58, 55, 54, 53, 53, 53, 52,
57, 59, 63, 68, 70, 74, 79, 81, 86, 90, 91, 97, 100, 102, 109, 110, 111,
114, 113, 117, 120, 121, 120, 119, 118, 65, 62, 60, 59, 59, 58, 58, 63,
65, 68, 73, 75, 79, 85, 86, 92, 97, 98, 105, 109, 111, 118, 120, 121,
125, 129, 128, 125, 124, 127, 131, 135, 79, 75, 73, 72, 71, 70, 69, 73,
75, 78, 84, 85, 90, 95, 97, 103, 108, 111, 118, 122, 125, 133, 135, 136,
140, 135, 133, 137, 141, 139, 137, 135, 81, 77, 75, 74, 72, 71, 70, 75,
77, 80, 85, 87, 91, 97, 99, 105, 110, 112, 119, 124, 127, 135, 137, 139,
143, 146, 150, 148, 144, 146, 150, 154, 88, 83, 81, 79, 78, 77, 76, 79,
81, 85, 88, 91, 97, 99, 104, 109, 111, 119, 123, 127, 135, 137, 145,
147, 148, 153, 153, 155, 160, 161, 158, 155, 90, 86, 84, 82, 81, 80, 78,
79, 83, 85, 89, 92, 94, 101, 102, 108, 112, 117, 123, 125, 134, 136,
143, 148, 154, 157, 158, 164, 164, 165, 170, 175, 93, 88, 88, 84, 84,
83, 82, 81, 84, 86, 90, 92, 97, 98, 105, 106, 113, 115, 122, 125, 131,
136, 141, 147, 151, 160, 163, 168, 169, 175, 175, 176, 96, 91, 91, 87,
87, 85, 86, 83, 84, 89, 89, 95, 95, 102, 102, 110, 110, 118, 119, 128,

```

```

129, 137, 138, 149, 149, 159, 160, 173, 174, 179, 180, 187,
/* Size 4x16 */
31, 39, 65, 90, 32, 38, 60, 84, 32, 39, 59, 81, 33, 40, 58, 78, 34, 47,
65, 83, 37, 54, 73, 89, 41, 58, 79, 94, 46, 62, 86, 102, 53, 68, 97,
112, 60, 73, 105, 123, 65, 78, 111, 134, 74, 85, 120, 143, 79, 90, 125,
154, 84, 90, 128, 158, 89, 95, 124, 164, 94, 101, 131, 170,
/* Size 16x4 */
31, 32, 32, 33, 34, 37, 41, 46, 53, 60, 65, 74, 79, 84, 89, 94, 39, 38,
39, 40, 47, 54, 58, 62, 68, 73, 78, 85, 90, 90, 95, 101, 65, 60, 59, 58,
65, 73, 79, 86, 97, 105, 111, 120, 125, 128, 124, 131, 90, 84, 81, 78,
83, 89, 94, 102, 112, 123, 134, 143, 154, 158, 164, 170,
/* Size 8x32 */
32, 32, 36, 44, 58, 79, 88, 93, 31, 32, 35, 42, 55, 75, 83, 88, 31, 32,
35, 41, 54, 73, 81, 88, 31, 32, 34, 41, 53, 72, 79, 84, 32, 33, 36, 42,
53, 71, 78, 84, 32, 34, 37, 42, 53, 70, 77, 83, 32, 34, 38, 42, 52, 69,
76, 82, 34, 35, 42, 48, 57, 73, 79, 81, 34, 36, 44, 50, 59, 75, 81, 84,
36, 37, 48, 54, 63, 78, 85, 86, 39, 39, 50, 58, 68, 84, 88, 90, 40, 40,
51, 59, 70, 85, 91, 92, 44, 42, 53, 63, 74, 90, 97, 97, 47, 45, 56, 66,
79, 95, 99, 98, 49, 46, 57, 67, 81, 97, 104, 105, 53, 50, 60, 71, 86,
103, 109, 106, 57, 53, 63, 74, 90, 108, 111, 113, 59, 54, 64, 75, 91,
111, 119, 115, 65, 59, 68, 79, 97, 118, 123, 122, 69, 62, 71, 83, 100,
122, 127, 125, 71, 64, 73, 84, 102, 125, 135, 131, 79, 71, 79, 90, 109,
133, 137, 136, 81, 72, 80, 91, 110, 135, 145, 141, 82, 73, 81, 92, 111,
136, 147, 147, 87, 77, 85, 96, 114, 140, 148, 151, 90, 80, 87, 99, 113,
135, 153, 160, 92, 83, 88, 102, 117, 133, 153, 163, 95, 85, 88, 103,
120, 137, 155, 168, 98, 88, 89, 103, 121, 141, 160, 169, 100, 91, 90,
103, 120, 139, 161, 175, 103, 94, 92, 103, 119, 137, 158, 175, 106, 97,
93, 104, 118, 135, 155, 176,
/* Size 32x8 */
32, 31, 31, 31, 32, 32, 32, 34, 34, 36, 39, 40, 44, 47, 49, 53, 57, 59,
65, 69, 71, 79, 81, 82, 87, 90, 92, 95, 98, 100, 103, 106, 32, 32, 32,
32, 33, 34, 34, 35, 36, 37, 39, 40, 42, 45, 46, 50, 53, 54, 59, 62, 64,
71, 72, 73, 77, 80, 83, 85, 88, 91, 94, 97, 36, 35, 35, 34, 36, 37, 38,
42, 44, 48, 50, 51, 53, 56, 57, 60, 63, 64, 68, 71, 73, 79, 80, 81, 85,
87, 88, 88, 89, 90, 92, 93, 44, 42, 41, 41, 42, 42, 42, 48, 50, 54, 58,
59, 63, 66, 67, 71, 74, 75, 79, 83, 84, 90, 91, 92, 96, 99, 102, 103,
103, 103, 103, 104, 58, 55, 54, 53, 53, 53, 52, 57, 59, 63, 68, 70, 74,
79, 81, 86, 90, 91, 97, 100, 102, 109, 110, 111, 114, 113, 117, 120,
121, 120, 119, 118, 79, 75, 73, 72, 71, 70, 69, 73, 75, 78, 84, 85, 90,
95, 97, 103, 108, 111, 118, 122, 125, 133, 135, 136, 140, 135, 133, 137,
141, 139, 137, 135, 88, 83, 81, 79, 78, 77, 76, 79, 81, 85, 88, 91, 97,
99, 104, 109, 111, 119, 123, 127, 135, 137, 145, 147, 148, 153, 153,
155, 160, 161, 158, 155, 93, 88, 88, 84, 84, 83, 82, 81, 84, 86, 90, 92,
97, 98, 105, 106, 113, 115, 122, 125, 131, 136, 141, 147, 151, 160, 163,
168, 169, 175, 175, 176 },
{ /* Chroma */
/* Size 4x4 */
32, 45, 53, 63, 45, 55, 62, 67, 53, 62, 80, 84, 63, 67, 84, 101,
/* Size 8x8 */
31, 36, 47, 48, 52, 60, 64, 67, 36, 43, 47, 46, 49, 55, 59, 63, 47, 47,

```

```

53, 54, 55, 60, 63, 64, 48, 46, 54, 61, 65, 70, 71, 71, 52, 49, 55, 65,
71, 78, 81, 79, 60, 55, 60, 70, 78, 89, 89, 89, 64, 59, 63, 71, 81, 89,
97, 99, 67, 63, 64, 71, 79, 89, 99, 104,
/* Size 16x16 */
32, 30, 33, 36, 44, 48, 49, 51, 54, 57, 60, 64, 67, 68, 70, 72, 30, 31,
35, 39, 44, 46, 46, 47, 50, 53, 55, 59, 61, 64, 66, 68, 33, 35, 39, 43,
46, 46, 45, 47, 49, 51, 53, 57, 59, 61, 63, 65, 36, 39, 43, 47, 47, 46,
45, 46, 48, 50, 52, 55, 57, 58, 61, 63, 44, 44, 46, 47, 50, 51, 51, 51,
53, 54, 56, 59, 61, 61, 63, 62, 48, 46, 46, 46, 51, 54, 55, 56, 58, 60,
61, 64, 65, 64, 66, 66, 49, 46, 45, 45, 51, 55, 58, 60, 62, 63, 65, 68,
69, 69, 69, 69, 51, 47, 47, 46, 51, 56, 60, 62, 65, 67, 69, 72, 73, 74,
73, 73, 54, 50, 49, 48, 53, 58, 62, 65, 70, 73, 75, 78, 79, 79, 77, 77,
57, 53, 51, 50, 54, 60, 63, 67, 73, 76, 79, 82, 84, 83, 82, 82, 60, 55,
53, 52, 56, 61, 65, 69, 75, 79, 82, 86, 88, 87, 86, 87, 64, 59, 57, 55,
59, 64, 68, 72, 78, 82, 86, 90, 93, 92, 91, 92, 67, 61, 59, 57, 61, 65,
69, 73, 79, 84, 88, 93, 95, 96, 96, 96, 68, 64, 61, 58, 61, 64, 69, 74,
79, 83, 87, 92, 96, 99, 100, 101, 70, 66, 63, 61, 63, 66, 69, 73, 77,
82, 86, 91, 96, 100, 103, 104, 72, 68, 65, 63, 62, 66, 69, 73, 77, 82,
87, 92, 96, 101, 104, 106,
/* Size 32x32 */
32, 31, 30, 30, 33, 35, 36, 41, 44, 49, 48, 48, 49, 50, 51, 52, 54, 55,
57, 59, 60, 63, 64, 65, 67, 68, 68, 69, 70, 71, 72, 73, 31, 31, 31, 31,
34, 36, 38, 42, 44, 47, 47, 47, 47, 48, 48, 50, 51, 52, 54, 56, 57, 60,
61, 61, 63, 64, 65, 66, 67, 67, 68, 69, 30, 31, 31, 31, 35, 37, 39, 42,
44, 47, 46, 46, 46, 47, 47, 48, 50, 51, 53, 54, 55, 58, 59, 60, 61, 63,
64, 65, 66, 67, 68, 69, 30, 31, 31, 32, 35, 37, 40, 42, 44, 46, 45, 45,
45, 46, 46, 47, 49, 50, 52, 53, 54, 57, 58, 58, 60, 61, 62, 63, 63, 64,
65, 66, 33, 34, 35, 35, 39, 41, 43, 45, 46, 47, 46, 46, 45, 46, 47, 47,
49, 49, 51, 53, 53, 56, 57, 57, 59, 60, 61, 62, 63, 64, 65, 66, 35, 36,
37, 37, 41, 43, 45, 46, 46, 47, 46, 46, 45, 46, 46, 47, 48, 49, 50, 52,
53, 55, 56, 56, 58, 59, 60, 61, 62, 63, 64, 64, 36, 38, 39, 40, 43, 45,
47, 47, 47, 48, 46, 46, 45, 46, 46, 47, 48, 48, 50, 51, 52, 54, 55, 55,
57, 58, 58, 59, 61, 62, 63, 64, 41, 42, 42, 42, 45, 46, 47, 48, 49, 50,
49, 49, 49, 50, 50, 50, 51, 52, 53, 54, 55, 57, 58, 58, 60, 60, 59, 59,
60, 61, 61, 62, 44, 44, 44, 44, 46, 46, 47, 49, 50, 51, 51, 51, 51, 51,
51, 52, 53, 53, 54, 56, 56, 59, 59, 59, 61, 61, 61, 62, 63, 62, 62, 62,
49, 47, 47, 46, 47, 47, 48, 50, 51, 53, 53, 53, 53, 54, 54, 54, 55, 55,
56, 58, 58, 60, 61, 61, 63, 63, 64, 63, 63, 64, 65, 66, 48, 47, 46, 45,
46, 46, 46, 49, 51, 53, 54, 54, 55, 56, 56, 57, 58, 59, 60, 61, 61, 63,
64, 64, 65, 65, 64, 65, 66, 66, 66, 66, 48, 47, 46, 45, 46, 46, 46, 49,
51, 53, 54, 55, 56, 57, 57, 58, 59, 60, 61, 62, 63, 65, 65, 65, 66, 67,
68, 67, 67, 67, 68, 69, 49, 47, 46, 45, 45, 45, 45, 49, 51, 53, 55, 56,
58, 59, 60, 61, 62, 62, 63, 65, 65, 67, 68, 68, 69, 70, 69, 69, 69, 70,
69, 69, 50, 48, 47, 46, 46, 46, 46, 50, 51, 54, 56, 57, 59, 61, 62, 63,
64, 65, 66, 68, 68, 70, 71, 71, 72, 71, 71, 72, 71, 71, 71, 72, 51, 48,
47, 46, 47, 46, 46, 50, 51, 54, 56, 57, 60, 62, 62, 64, 65, 66, 67, 69,
69, 71, 72, 72, 73, 74, 74, 72, 73, 74, 73, 73, 52, 50, 48, 47, 47, 47,
47, 50, 52, 54, 57, 58, 61, 63, 64, 66, 68, 68, 70, 72, 72, 75, 75, 75,
77, 76, 75, 76, 76, 74, 75, 76, 54, 51, 50, 49, 49, 48, 48, 51, 53, 55,
58, 59, 62, 64, 65, 68, 70, 70, 73, 74, 75, 77, 78, 78, 79, 78, 79, 78,

```

```

77, 78, 77, 77, 55, 52, 51, 50, 49, 49, 48, 52, 53, 55, 59, 60, 62, 65,
66, 68, 70, 71, 73, 75, 76, 78, 79, 79, 80, 81, 80, 80, 81, 79, 79, 81,
57, 54, 53, 52, 51, 50, 50, 53, 54, 56, 60, 61, 63, 66, 67, 70, 73, 73,
76, 78, 79, 82, 82, 83, 84, 83, 83, 83, 82, 83, 82, 81, 59, 56, 54, 53,
53, 52, 51, 54, 56, 58, 61, 62, 65, 68, 69, 72, 74, 75, 78, 80, 81, 84,
85, 85, 86, 86, 86, 84, 85, 84, 84, 85, 60, 57, 55, 54, 53, 53, 52, 55,
56, 58, 61, 63, 65, 68, 69, 72, 75, 76, 79, 81, 82, 85, 86, 86, 88, 88,
87, 88, 86, 87, 87, 85, 63, 60, 58, 57, 56, 55, 54, 57, 59, 60, 63, 65,
67, 70, 71, 75, 77, 78, 82, 84, 85, 89, 89, 90, 92, 89, 91, 89, 90, 89,
88, 89, 64, 61, 59, 58, 57, 56, 55, 58, 59, 61, 64, 65, 68, 71, 72, 75,
78, 79, 82, 85, 86, 89, 90, 91, 93, 94, 92, 92, 91, 91, 92, 90, 65, 61,
60, 58, 57, 56, 55, 58, 59, 61, 64, 65, 68, 71, 72, 75, 78, 79, 83, 85,
86, 90, 91, 91, 93, 94, 95, 94, 94, 94, 93, 94, 67, 63, 61, 60, 59, 58,
57, 60, 61, 63, 65, 66, 69, 72, 73, 77, 79, 80, 84, 86, 88, 92, 93, 93,
95, 95, 96, 97, 96, 95, 96, 94, 68, 64, 63, 61, 60, 59, 58, 60, 61, 63,
65, 67, 70, 71, 74, 76, 78, 81, 83, 86, 88, 89, 94, 94, 95, 97, 97, 98,
99, 99, 97, 99, 68, 65, 64, 62, 61, 60, 58, 59, 61, 64, 64, 68, 69, 71,
74, 75, 79, 80, 83, 86, 87, 91, 92, 95, 96, 97, 99, 99, 100, 100, 101,
99, 69, 66, 65, 63, 62, 61, 59, 59, 62, 63, 65, 67, 69, 72, 72, 76, 78,
80, 83, 84, 88, 89, 92, 94, 97, 98, 99, 101, 100, 102, 102, 104, 70, 67,
66, 63, 63, 62, 61, 60, 63, 63, 66, 67, 69, 71, 73, 76, 77, 81, 82, 85,
86, 90, 91, 94, 96, 99, 100, 100, 103, 102, 104, 104, 71, 67, 67, 64,
64, 63, 62, 61, 62, 64, 66, 67, 70, 71, 74, 74, 78, 79, 83, 84, 87, 89,
91, 94, 95, 99, 100, 102, 102, 104, 104, 106, 72, 68, 68, 65, 65, 64,
63, 61, 62, 65, 66, 68, 69, 71, 73, 75, 77, 79, 82, 84, 87, 88, 92, 93,
96, 97, 101, 102, 104, 104, 106, 106, 73, 69, 69, 66, 66, 64, 64, 62,
62, 66, 66, 69, 69, 72, 73, 76, 77, 81, 81, 85, 85, 89, 90, 94, 94, 99,
99, 104, 104, 106, 106, 108,
/* Size 4x8 */
31, 47, 54, 64, 38, 46, 50, 60, 46, 53, 57, 62, 46, 56, 66, 71, 50, 59,
74, 79, 57, 64, 82, 88, 61, 65, 85, 97, 65, 67, 82, 99,
/* Size 8x4 */
31, 38, 46, 46, 50, 57, 61, 65, 47, 46, 53, 56, 59, 64, 65, 67, 54, 50,
57, 66, 74, 82, 85, 82, 64, 60, 62, 71, 79, 88, 97, 99,
/* Size 8x16 */
32, 34, 48, 49, 54, 63, 67, 69, 31, 36, 46, 46, 50, 58, 62, 65, 33, 40,
47, 46, 49, 56, 59, 62, 37, 44, 47, 45, 48, 54, 57, 60, 44, 46, 51, 51,
53, 59, 60, 61, 48, 46, 53, 56, 58, 64, 64, 64, 49, 45, 53, 58, 62, 67,
70, 68, 51, 47, 54, 60, 65, 71, 73, 72, 54, 49, 55, 62, 70, 77, 77, 76,
57, 51, 56, 64, 73, 82, 83, 81, 60, 53, 58, 65, 75, 85, 89, 85, 64, 57,
61, 68, 78, 89, 93, 89, 66, 59, 63, 69, 79, 91, 94, 93, 68, 61, 63, 71,
79, 87, 96, 98, 70, 63, 63, 70, 80, 89, 97, 100, 72, 65, 63, 69, 77, 86,
95, 102,
/* Size 16x8 */
32, 31, 33, 37, 44, 48, 49, 51, 54, 57, 60, 64, 66, 68, 70, 72, 34, 36,
40, 44, 46, 46, 45, 47, 49, 51, 53, 57, 59, 61, 63, 65, 48, 46, 47, 47,
51, 53, 53, 54, 55, 56, 58, 61, 63, 63, 63, 63, 49, 46, 46, 45, 51, 56,
58, 60, 62, 64, 65, 68, 69, 71, 70, 69, 54, 50, 49, 48, 53, 58, 62, 65,
70, 73, 75, 78, 79, 79, 80, 77, 63, 58, 56, 54, 59, 64, 67, 71, 77, 82,
85, 89, 91, 87, 89, 86, 67, 62, 59, 57, 60, 64, 70, 73, 77, 83, 89, 93,

```

94, 96, 97, 95, 69, 65, 62, 60, 61, 64, 68, 72, 76, 81, 85, 89, 93, 98,  
100, 102,

*/\* Size 16x32 \*/*

32, 31, 34, 37, 48, 48, 49, 52, 54, 57, 63, 64, 67, 68, 69, 69, 31, 31,  
35, 38, 47, 47, 47, 50, 51, 54, 60, 61, 63, 64, 65, 66, 31, 32, 36, 39,  
46, 46, 46, 48, 50, 53, 58, 59, 62, 63, 65, 66, 30, 32, 36, 40, 46, 45,  
45, 48, 49, 52, 57, 58, 60, 61, 62, 63, 33, 36, 40, 43, 47, 46, 46, 47,  
49, 51, 56, 57, 59, 60, 62, 63, 35, 38, 42, 45, 47, 46, 45, 47, 48, 50,  
55, 56, 58, 60, 61, 61, 37, 40, 44, 47, 47, 46, 45, 47, 48, 50, 54, 55,  
57, 58, 60, 61, 42, 43, 45, 47, 50, 50, 49, 50, 51, 53, 57, 58, 59, 58,  
59, 59, 44, 44, 46, 47, 51, 51, 51, 52, 53, 54, 59, 59, 60, 61, 61, 60,  
49, 46, 47, 48, 53, 53, 53, 54, 55, 57, 60, 61, 63, 62, 62, 63, 48, 46,  
46, 47, 53, 54, 56, 57, 58, 60, 64, 64, 64, 64, 64, 63, 48, 45, 46, 46,  
53, 55, 56, 58, 59, 61, 65, 65, 66, 66, 65, 66, 49, 45, 45, 46, 53, 56,  
58, 61, 62, 64, 67, 68, 70, 67, 68, 66, 50, 46, 46, 46, 54, 56, 59, 63,  
65, 66, 70, 71, 70, 71, 68, 70, 51, 47, 47, 47, 54, 57, 60, 64, 65, 68,  
71, 72, 73, 71, 72, 70, 52, 48, 47, 47, 54, 57, 61, 66, 68, 71, 75, 75,  
76, 75, 73, 73, 54, 49, 49, 48, 55, 58, 62, 68, 70, 73, 77, 78, 77, 77,  
76, 74, 54, 50, 49, 49, 55, 59, 62, 68, 70, 74, 78, 79, 81, 79, 77, 78,  
57, 52, 51, 50, 56, 60, 64, 70, 73, 76, 82, 82, 83, 82, 81, 78, 59, 54,  
52, 52, 58, 61, 65, 72, 74, 78, 84, 85, 85, 83, 82, 82, 60, 54, 53, 52,  
58, 62, 65, 72, 75, 79, 85, 86, 89, 87, 85, 82, 63, 57, 56, 55, 60, 64,  
67, 75, 77, 82, 89, 90, 90, 88, 87, 86, 64, 58, 57, 55, 61, 64, 68, 75,  
78, 82, 89, 90, 93, 91, 89, 87, 64, 59, 57, 56, 61, 65, 68, 75, 78, 83,  
90, 91, 94, 93, 92, 91, 66, 60, 59, 57, 63, 66, 69, 77, 79, 84, 91, 93,  
94, 95, 93, 91, 67, 61, 60, 58, 63, 65, 70, 75, 78, 85, 88, 93, 96, 97,  
97, 95, 68, 62, 61, 59, 63, 64, 71, 74, 79, 84, 87, 94, 96, 97, 98, 96,  
69, 63, 62, 60, 63, 65, 71, 72, 80, 82, 88, 93, 96, 99, 100, 101, 70,  
64, 63, 60, 63, 66, 70, 73, 80, 81, 89, 90, 97, 99, 100, 101, 71, 65,  
64, 61, 63, 67, 70, 74, 78, 82, 88, 90, 97, 99, 102, 103, 72, 65, 65,  
62, 63, 68, 69, 75, 77, 83, 86, 92, 95, 100, 102, 103, 73, 66, 66, 63,  
63, 69, 69, 76, 76, 84, 84, 93, 93, 101, 101, 105,

*/\* Size 32x16 \*/*

32, 31, 31, 30, 33, 35, 37, 42, 44, 49, 48, 48, 49, 50, 51, 52, 54, 54,  
57, 59, 60, 63, 64, 64, 66, 67, 68, 69, 70, 71, 72, 73, 31, 31, 32, 32,  
36, 38, 40, 43, 44, 46, 46, 45, 45, 46, 47, 48, 49, 50, 52, 54, 54, 57,  
58, 59, 60, 61, 62, 63, 64, 65, 65, 66, 34, 35, 36, 36, 40, 42, 44, 45,  
46, 47, 46, 46, 45, 46, 47, 47, 49, 49, 51, 52, 53, 56, 57, 57, 59, 60,  
61, 62, 63, 64, 65, 66, 37, 38, 39, 40, 43, 45, 47, 47, 47, 48, 47, 46,  
46, 46, 47, 47, 48, 49, 50, 52, 52, 55, 55, 56, 57, 58, 59, 60, 60, 61,  
62, 63, 48, 47, 46, 46, 47, 47, 47, 50, 51, 53, 53, 53, 53, 54, 54, 54,  
55, 55, 56, 58, 58, 60, 61, 61, 63, 63, 63, 63, 63, 63, 63, 48, 47,  
46, 45, 46, 46, 46, 50, 51, 53, 54, 55, 56, 56, 57, 57, 58, 59, 60, 61,  
62, 64, 64, 65, 66, 65, 64, 65, 66, 67, 68, 69, 49, 47, 46, 45, 46, 45,  
45, 49, 51, 53, 56, 56, 58, 59, 60, 61, 62, 62, 64, 65, 65, 67, 68, 68,  
69, 70, 71, 71, 70, 70, 69, 69, 52, 50, 48, 48, 47, 47, 47, 50, 52, 54,  
57, 58, 61, 63, 64, 66, 68, 68, 70, 72, 72, 75, 75, 75, 77, 75, 74, 72,  
73, 74, 75, 76, 54, 51, 50, 49, 49, 48, 48, 51, 53, 55, 58, 59, 62, 65,  
65, 68, 70, 70, 73, 74, 75, 77, 78, 78, 79, 78, 79, 80, 80, 78, 77, 76,  
57, 54, 53, 52, 51, 50, 50, 53, 54, 57, 60, 61, 64, 66, 68, 71, 73, 74,



```

76, 78, 79, 82, 82, 83, 84, 85, 84, 82, 81, 82, 83, 84, 63, 60, 58, 57,
56, 55, 54, 57, 59, 60, 64, 65, 67, 70, 71, 75, 77, 78, 82, 84, 85, 89,
89, 90, 91, 88, 87, 88, 89, 88, 86, 84, 64, 61, 59, 58, 57, 56, 55, 58,
59, 61, 64, 65, 68, 71, 72, 75, 78, 79, 82, 85, 86, 90, 90, 91, 93, 93,
94, 93, 90, 90, 92, 93, 67, 63, 62, 60, 59, 58, 57, 59, 60, 63, 64, 66,
70, 70, 73, 76, 77, 81, 83, 85, 89, 90, 93, 94, 94, 96, 96, 96, 97, 97,
95, 93, 68, 64, 63, 61, 60, 60, 58, 58, 61, 62, 64, 66, 67, 71, 71, 75,
77, 79, 82, 83, 87, 88, 91, 93, 95, 97, 97, 99, 99, 99, 100, 101, 69,
65, 65, 62, 62, 61, 60, 59, 61, 62, 64, 65, 68, 68, 72, 73, 76, 77, 81,
82, 85, 87, 89, 92, 93, 97, 98, 100, 100, 102, 102, 101, 69, 66, 66, 63,
63, 61, 61, 59, 60, 63, 63, 66, 66, 70, 70, 73, 74, 78, 78, 82, 82, 86,
87, 91, 91, 95, 96, 101, 101, 103, 103, 105,
/* Size 4x16 */
31, 48, 57, 68, 32, 46, 53, 63, 36, 46, 51, 60, 40, 46, 50, 58, 44, 51,
54, 61, 46, 54, 60, 64, 45, 56, 64, 67, 47, 57, 68, 71, 49, 58, 73, 77,
52, 60, 76, 82, 54, 62, 79, 87, 58, 64, 82, 91, 60, 66, 84, 95, 62, 64,
84, 97, 64, 66, 81, 99, 65, 68, 83, 100,
/* Size 16x4 */
31, 32, 36, 40, 44, 46, 45, 47, 49, 52, 54, 58, 60, 62, 64, 65, 48, 46,
46, 46, 51, 54, 56, 57, 58, 60, 62, 64, 66, 64, 66, 68, 57, 53, 51, 50,
54, 60, 64, 68, 73, 76, 79, 82, 84, 84, 81, 83, 68, 63, 60, 58, 61, 64,
67, 71, 77, 82, 87, 91, 95, 97, 99, 100,
/* Size 8x32 */
32, 34, 48, 49, 54, 63, 67, 69, 31, 35, 47, 47, 51, 60, 63, 65, 31, 36,
46, 46, 50, 58, 62, 65, 30, 36, 46, 45, 49, 57, 60, 62, 33, 40, 47, 46,
49, 56, 59, 62, 35, 42, 47, 45, 48, 55, 58, 61, 37, 44, 47, 45, 48, 54,
57, 60, 42, 45, 50, 49, 51, 57, 59, 59, 44, 46, 51, 51, 53, 59, 60, 61,
49, 47, 53, 53, 55, 60, 63, 62, 48, 46, 53, 56, 58, 64, 64, 64, 48, 46,
53, 56, 59, 65, 66, 65, 49, 45, 53, 58, 62, 67, 70, 68, 50, 46, 54, 59,
65, 70, 70, 68, 51, 47, 54, 60, 65, 71, 73, 72, 52, 47, 54, 61, 68, 75,
76, 73, 54, 49, 55, 62, 70, 77, 77, 76, 54, 49, 55, 62, 70, 78, 81, 77,
57, 51, 56, 64, 73, 82, 83, 81, 59, 52, 58, 65, 74, 84, 85, 82, 60, 53,
58, 65, 75, 85, 89, 85, 63, 56, 60, 67, 77, 89, 90, 87, 64, 57, 61, 68,
78, 89, 93, 89, 64, 57, 61, 68, 78, 90, 94, 92, 66, 59, 63, 69, 79, 91,
94, 93, 67, 60, 63, 70, 78, 88, 96, 97, 68, 61, 63, 71, 79, 87, 96, 98,
69, 62, 63, 71, 80, 88, 96, 100, 70, 63, 63, 70, 80, 89, 97, 100, 71,
64, 63, 70, 78, 88, 97, 102, 72, 65, 63, 69, 77, 86, 95, 102, 73, 66,
63, 69, 76, 84, 93, 101,
/* Size 32x8 */
32, 31, 31, 30, 33, 35, 37, 42, 44, 49, 48, 48, 49, 50, 51, 52, 54, 54,
57, 59, 60, 63, 64, 64, 66, 67, 68, 69, 70, 71, 72, 73, 34, 35, 36, 36,
40, 42, 44, 45, 46, 47, 46, 46, 45, 46, 47, 47, 49, 49, 51, 52, 53, 56,
57, 57, 59, 60, 61, 62, 63, 64, 65, 66, 48, 47, 46, 46, 47, 47, 50,
51, 53, 53, 53, 53, 54, 54, 54, 55, 55, 56, 58, 58, 60, 61, 61, 63, 63,
63, 63, 63, 63, 63, 49, 47, 46, 45, 46, 45, 45, 49, 51, 53, 56, 56,
58, 59, 60, 61, 62, 62, 64, 65, 65, 67, 68, 68, 69, 70, 71, 71, 70, 70,
69, 69, 54, 51, 50, 49, 49, 48, 48, 51, 53, 55, 58, 59, 62, 65, 65, 68,
70, 70, 73, 74, 75, 77, 78, 78, 79, 78, 79, 80, 80, 78, 77, 76, 63, 60,
58, 57, 56, 55, 54, 57, 59, 60, 64, 65, 67, 70, 71, 75, 77, 78, 82, 84,
85, 89, 89, 90, 91, 88, 87, 88, 89, 88, 86, 84, 67, 63, 62, 60, 59, 58,

```

```

57, 59, 60, 63, 64, 66, 70, 70, 73, 76, 77, 81, 83, 85, 89, 90, 93, 94,
94, 96, 96, 96, 97, 97, 95, 93, 69, 65, 65, 62, 62, 61, 60, 59, 61, 62,
64, 65, 68, 68, 72, 73, 76, 77, 81, 82, 85, 87, 89, 92, 93, 97, 98, 100,
100, 102, 102, 101 },
},
{
{ /* Luma */
/* Size 4x4 */
32, 37, 58, 81, 37, 54, 72, 91, 58, 72, 102, 121, 81, 91, 121, 156,
/* Size 8x8 */
32, 32, 35, 42, 53, 68, 78, 90, 32, 33, 36, 42, 51, 64, 74, 84, 35, 36,
46, 52, 60, 72, 80, 87, 42, 42, 52, 63, 73, 84, 92, 98, 53, 51, 60, 73,
86, 100, 109, 114, 68, 64, 72, 84, 100, 117, 128, 133, 78, 74, 80, 92,
109, 128, 140, 155, 90, 84, 87, 98, 114, 133, 155, 168,
/* Size 16x16 */
32, 31, 31, 32, 34, 36, 41, 47, 54, 59, 65, 74, 82, 87, 92, 97, 31, 32,
32, 32, 34, 35, 39, 45, 50, 55, 61, 69, 76, 81, 87, 92, 31, 32, 33, 33,
35, 36, 40, 44, 49, 54, 59, 67, 73, 78, 83, 88, 32, 32, 33, 35, 37, 38,
41, 45, 49, 53, 58, 65, 71, 75, 80, 86, 34, 34, 35, 37, 39, 42, 46, 50,
54, 58, 63, 70, 76, 80, 84, 85, 36, 35, 36, 38, 42, 48, 52, 56, 60, 64,
68, 75, 80, 85, 90, 91, 41, 39, 40, 41, 46, 52, 57, 62, 67, 71, 75, 83,
88, 92, 95, 97, 47, 45, 44, 45, 50, 56, 62, 69, 75, 79, 84, 91, 97, 100,
102, 104, 54, 50, 49, 49, 54, 60, 67, 75, 82, 87, 92, 100, 106, 110,
109, 112, 59, 55, 54, 53, 58, 64, 71, 79, 87, 92, 98, 106, 112, 117,
117, 121, 65, 61, 59, 58, 63, 68, 75, 84, 92, 98, 105, 114, 120, 125,
126, 130, 74, 69, 67, 65, 70, 75, 83, 91, 100, 106, 114, 123, 131, 135,
137, 140, 82, 76, 73, 71, 76, 80, 88, 97, 106, 112, 120, 131, 139, 144,
148, 150, 87, 81, 78, 75, 80, 85, 92, 100, 110, 117, 125, 135, 144, 150,
155, 162, 92, 87, 83, 80, 84, 90, 95, 102, 109, 117, 126, 137, 148, 155,
162, 168, 97, 92, 88, 86, 85, 91, 97, 104, 112, 121, 130, 140, 150, 162,
168, 174,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 32, 32, 34, 35, 36, 39, 41, 44, 47, 48, 54, 56,
59, 64, 65, 71, 74, 80, 82, 83, 87, 90, 92, 95, 97, 100, 31, 32, 32, 32,
32, 32, 32, 33, 34, 35, 35, 38, 40, 42, 45, 46, 51, 53, 56, 61, 62, 68,
71, 76, 78, 78, 83, 85, 88, 90, 92, 95, 31, 32, 32, 32, 32, 32, 32, 33,
34, 34, 35, 38, 39, 42, 45, 45, 50, 52, 55, 60, 61, 67, 69, 74, 76, 77,
81, 84, 87, 89, 92, 95, 31, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 37,
38, 41, 44, 44, 49, 51, 54, 58, 59, 65, 68, 72, 74, 75, 79, 81, 84, 86,
88, 90, 31, 32, 32, 32, 33, 33, 33, 34, 35, 36, 36, 39, 40, 42, 44, 45,
49, 51, 54, 58, 59, 64, 67, 71, 73, 74, 78, 80, 83, 85, 88, 90, 31, 32,
32, 32, 33, 33, 34, 34, 35, 36, 36, 39, 40, 42, 45, 45, 50, 51, 54, 58,
59, 64, 67, 71, 73, 74, 78, 80, 82, 84, 86, 89, 32, 32, 32, 32, 33, 34,
35, 36, 37, 38, 38, 40, 41, 42, 45, 46, 49, 51, 53, 57, 58, 63, 65, 69,
71, 72, 75, 78, 80, 83, 86, 89, 32, 33, 33, 33, 34, 34, 36, 36, 38, 39,
40, 42, 43, 44, 47, 47, 51, 53, 55, 59, 60, 65, 67, 71, 73, 73, 77, 78,
80, 82, 84, 86, 34, 34, 34, 33, 35, 35, 37, 38, 39, 42, 42, 45, 46, 47,
50, 51, 54, 56, 58, 62, 63, 68, 70, 74, 76, 76, 80, 82, 84, 85, 85, 86,
35, 35, 34, 34, 36, 36, 38, 39, 42, 46, 47, 49, 50, 52, 55, 55, 59, 60,
62, 66, 67, 72, 74, 78, 79, 80, 83, 84, 85, 87, 90, 92, 36, 35, 35, 34,

```

```

36, 36, 38, 40, 42, 47, 48, 50, 52, 54, 56, 57, 60, 61, 64, 67, 68, 73,
75, 79, 80, 81, 85, 87, 90, 91, 91, 92, 39, 38, 38, 37, 39, 39, 40, 42,
45, 49, 50, 54, 55, 58, 60, 61, 65, 66, 69, 72, 73, 78, 80, 84, 86, 86,
90, 91, 91, 92, 95, 97, 41, 40, 39, 38, 40, 40, 41, 43, 46, 50, 52, 55,
57, 60, 62, 63, 67, 69, 71, 75, 75, 80, 83, 86, 88, 89, 92, 93, 95, 97,
97, 98, 44, 42, 42, 41, 42, 42, 42, 44, 47, 52, 54, 58, 60, 63, 66, 67,
71, 73, 75, 79, 79, 84, 86, 90, 92, 92, 96, 98, 98, 98, 101, 104, 47,
45, 45, 44, 44, 45, 45, 47, 50, 55, 56, 60, 62, 66, 69, 70, 75, 77, 79,
83, 84, 89, 91, 95, 97, 97, 100, 99, 102, 105, 104, 104, 48, 46, 45, 44,
45, 45, 46, 47, 51, 55, 57, 61, 63, 67, 70, 71, 76, 78, 80, 84, 85, 90,
93, 96, 98, 99, 102, 106, 106, 105, 108, 111, 54, 51, 50, 49, 49, 50,
49, 51, 54, 59, 60, 65, 67, 71, 75, 76, 82, 84, 87, 91, 92, 97, 100,
104, 106, 106, 110, 108, 109, 112, 112, 111, 56, 53, 52, 51, 51, 51, 51,
53, 56, 60, 61, 66, 69, 73, 77, 78, 84, 86, 89, 93, 94, 100, 102, 106,
108, 109, 112, 113, 115, 114, 116, 119, 59, 56, 55, 54, 54, 54, 53, 55,
58, 62, 64, 69, 71, 75, 79, 80, 87, 89, 92, 97, 98, 103, 106, 110, 112,
113, 117, 118, 117, 121, 121, 119, 64, 61, 60, 58, 58, 58, 57, 59, 62,
66, 67, 72, 75, 79, 83, 84, 91, 93, 97, 102, 103, 109, 112, 116, 118,
119, 122, 121, 125, 123, 125, 128, 65, 62, 61, 59, 59, 59, 58, 60, 63,
67, 68, 73, 75, 79, 84, 85, 92, 94, 98, 103, 105, 111, 114, 118, 120,
121, 125, 129, 126, 129, 130, 129, 71, 68, 67, 65, 64, 64, 63, 65, 68,
72, 73, 78, 80, 84, 89, 90, 97, 100, 103, 109, 111, 117, 120, 125, 127,
128, 133, 130, 134, 133, 133, 137, 74, 71, 69, 68, 67, 67, 65, 67, 70,
74, 75, 80, 83, 86, 91, 93, 100, 102, 106, 112, 114, 120, 123, 128, 131,
131, 135, 137, 137, 138, 140, 137, 80, 76, 74, 72, 71, 71, 69, 71, 74,
78, 79, 84, 86, 90, 95, 96, 104, 106, 110, 116, 118, 125, 128, 134, 136,
137, 142, 141, 142, 143, 143, 147, 82, 78, 76, 74, 73, 73, 71, 73, 76,
79, 80, 86, 88, 92, 97, 98, 106, 108, 112, 118, 120, 127, 131, 136, 139,
139, 144, 147, 148, 147, 150, 148, 83, 78, 77, 75, 74, 74, 72, 73, 76,
80, 81, 86, 89, 92, 97, 99, 106, 109, 113, 119, 121, 128, 131, 137, 139,
140, 145, 150, 152, 155, 152, 157, 87, 83, 81, 79, 78, 78, 75, 77, 80,
83, 85, 90, 92, 96, 100, 102, 110, 112, 117, 122, 125, 133, 135, 142,
144, 145, 150, 151, 155, 158, 162, 158, 90, 85, 84, 81, 80, 80, 78, 78,
82, 84, 87, 91, 93, 98, 99, 106, 108, 113, 118, 121, 129, 130, 137, 141,
147, 150, 151, 156, 156, 161, 164, 169, 92, 88, 87, 84, 83, 82, 80, 80,
84, 85, 90, 91, 95, 98, 102, 106, 109, 115, 117, 125, 126, 134, 137,
142, 148, 152, 155, 156, 162, 162, 168, 170, 95, 90, 89, 86, 85, 84, 83,
82, 85, 87, 91, 92, 97, 98, 105, 105, 112, 114, 121, 123, 129, 133, 138,
143, 147, 155, 158, 161, 162, 168, 168, 174, 97, 92, 92, 88, 88, 86, 86,
84, 85, 90, 91, 95, 97, 101, 104, 108, 112, 116, 121, 125, 130, 133,
140, 143, 150, 152, 162, 164, 168, 168, 174, 175, 100, 95, 95, 90, 90,
89, 89, 86, 86, 92, 92, 97, 98, 104, 104, 111, 111, 119, 119, 128, 129,
137, 137, 147, 148, 157, 158, 169, 170, 174, 175, 181,
/* Size 4x8 */
32, 35, 59, 83, 32, 36, 57, 78, 34, 47, 65, 82, 41, 53, 78, 97, 51, 61,
92, 111, 65, 73, 108, 129, 75, 81, 117, 148, 86, 92, 119, 154,
/* Size 8x4 */
32, 32, 34, 41, 51, 65, 75, 86, 35, 36, 47, 53, 61, 73, 81, 92, 59, 57,
65, 78, 92, 108, 117, 119, 83, 78, 82, 97, 111, 129, 148, 154,
/* Size 8x16 */

```

```

32, 31, 35, 44, 53, 65, 82, 90, 31, 32, 34, 41, 50, 61, 76, 85, 31, 33,
35, 42, 49, 59, 73, 81, 32, 34, 37, 42, 49, 58, 71, 79, 34, 35, 41, 48,
54, 63, 76, 81, 36, 36, 46, 54, 60, 68, 80, 87, 41, 40, 49, 60, 67, 76,
88, 93, 47, 44, 53, 66, 75, 84, 97, 101, 53, 50, 57, 71, 82, 92, 106,
108, 58, 54, 61, 75, 87, 98, 112, 116, 65, 59, 66, 79, 92, 105, 120,
124, 74, 67, 73, 86, 100, 113, 131, 134, 82, 73, 79, 92, 105, 120, 139,
142, 87, 78, 83, 96, 110, 125, 144, 153, 92, 83, 84, 97, 114, 132, 150,
157, 97, 88, 86, 97, 111, 128, 147, 163,
/* Size 16x8 */
32, 31, 31, 32, 34, 36, 41, 47, 53, 58, 65, 74, 82, 87, 92, 97, 31, 32,
33, 34, 35, 36, 40, 44, 50, 54, 59, 67, 73, 78, 83, 88, 35, 34, 35, 37,
41, 46, 49, 53, 57, 61, 66, 73, 79, 83, 84, 86, 44, 41, 42, 42, 48, 54,
60, 66, 71, 75, 79, 86, 92, 96, 97, 97, 53, 50, 49, 49, 54, 60, 67, 75,
82, 87, 92, 100, 105, 110, 114, 111, 65, 61, 59, 58, 63, 68, 76, 84, 92,
98, 105, 113, 120, 125, 132, 128, 82, 76, 73, 71, 76, 80, 88, 97, 106,
112, 120, 131, 139, 144, 150, 147, 90, 85, 81, 79, 81, 87, 93, 101, 108,
116, 124, 134, 142, 153, 157, 163,
/* Size 16x32 */
32, 31, 31, 32, 35, 36, 44, 47, 53, 62, 65, 79, 82, 88, 90, 93, 31, 32,
32, 32, 35, 35, 42, 45, 51, 59, 62, 75, 78, 83, 86, 88, 31, 32, 32, 32,
34, 35, 41, 45, 50, 58, 61, 74, 76, 82, 85, 88, 31, 32, 32, 33, 34, 34,
41, 44, 49, 57, 59, 72, 74, 79, 82, 84, 31, 32, 33, 34, 35, 36, 42, 44,
49, 57, 59, 71, 73, 79, 81, 84, 32, 32, 33, 34, 36, 36, 42, 45, 50, 57,
59, 71, 73, 78, 80, 82, 32, 33, 34, 35, 37, 38, 42, 45, 49, 56, 58, 69,
71, 76, 79, 83, 32, 33, 34, 36, 39, 40, 44, 47, 51, 58, 60, 71, 73, 76,
78, 80, 34, 34, 35, 37, 41, 42, 48, 50, 54, 61, 63, 73, 76, 81, 81, 80,
35, 34, 36, 38, 45, 47, 52, 55, 59, 65, 67, 77, 79, 82, 83, 86, 36, 34,
36, 38, 46, 48, 54, 56, 60, 66, 68, 78, 80, 85, 87, 86, 39, 37, 39, 40,
48, 50, 58, 60, 65, 71, 73, 84, 86, 89, 88, 91, 41, 39, 40, 41, 49, 51,
60, 62, 67, 74, 76, 86, 88, 91, 93, 91, 44, 41, 42, 43, 51, 53, 63, 66,
71, 78, 79, 90, 92, 97, 94, 97, 47, 44, 44, 45, 53, 56, 66, 69, 75, 82,
84, 95, 97, 98, 101, 98, 48, 45, 45, 46, 54, 56, 67, 70, 76, 83, 85, 96,
98, 104, 101, 105, 53, 49, 50, 50, 57, 60, 71, 75, 82, 90, 92, 103, 106,
107, 108, 105, 55, 51, 51, 51, 59, 61, 72, 77, 84, 92, 94, 106, 108,
111, 110, 112, 58, 54, 54, 54, 61, 63, 75, 79, 87, 95, 98, 110, 112,
117, 116, 113, 63, 58, 58, 57, 65, 67, 78, 83, 91, 100, 103, 116, 118,
119, 119, 121, 65, 60, 59, 58, 66, 68, 79, 84, 92, 102, 105, 118, 120,
127, 124, 122, 71, 65, 64, 63, 71, 73, 84, 89, 97, 108, 111, 125, 127,
129, 129, 130, 74, 68, 67, 66, 73, 75, 86, 91, 100, 110, 113, 128, 131,
135, 134, 130, 79, 72, 71, 70, 77, 79, 90, 95, 104, 115, 118, 133, 136,
140, 139, 140, 82, 75, 73, 72, 79, 81, 92, 97, 105, 117, 120, 136, 139,
145, 142, 140, 82, 75, 74, 72, 79, 81, 92, 97, 106, 117, 121, 136, 139,
148, 150, 149, 87, 79, 78, 76, 83, 85, 96, 100, 110, 120, 125, 141, 144,
148, 153, 150, 89, 82, 81, 78, 83, 87, 97, 99, 113, 118, 128, 139, 145,
153, 157, 161, 92, 84, 83, 80, 84, 89, 97, 101, 114, 116, 132, 135, 150,
153, 157, 162, 94, 86, 85, 82, 85, 92, 97, 104, 112, 119, 130, 136, 151,
154, 163, 166, 97, 88, 88, 85, 86, 94, 97, 107, 111, 123, 128, 140, 147,
159, 163, 167, 99, 91, 91, 87, 87, 97, 97, 110, 110, 126, 126, 144, 144,
163, 163, 173,
/* Size 32x16 */

```

```

32, 31, 31, 31, 31, 32, 32, 32, 34, 35, 36, 39, 41, 44, 47, 48, 53, 55,
58, 63, 65, 71, 74, 79, 82, 82, 87, 89, 92, 94, 97, 99, 31, 32, 32, 32,
32, 32, 33, 33, 34, 34, 34, 37, 39, 41, 44, 45, 49, 51, 54, 58, 60, 65,
68, 72, 75, 75, 79, 82, 84, 86, 88, 91, 31, 32, 32, 32, 33, 33, 34, 34,
35, 36, 36, 39, 40, 42, 44, 45, 50, 51, 54, 58, 59, 64, 67, 71, 73, 74,
78, 81, 83, 85, 88, 91, 32, 32, 32, 33, 34, 34, 35, 36, 37, 38, 38, 40,
41, 43, 45, 46, 50, 51, 54, 57, 58, 63, 66, 70, 72, 72, 76, 78, 80, 82,
85, 87, 35, 35, 34, 34, 35, 36, 37, 39, 41, 45, 46, 48, 49, 51, 53, 54,
57, 59, 61, 65, 66, 71, 73, 77, 79, 79, 83, 83, 84, 85, 86, 87, 36, 35,
35, 34, 36, 36, 38, 40, 42, 47, 48, 50, 51, 53, 56, 56, 60, 61, 63, 67,
68, 73, 75, 79, 81, 81, 85, 87, 89, 92, 94, 97, 44, 42, 41, 41, 42, 42,
42, 44, 48, 52, 54, 58, 60, 63, 66, 67, 71, 72, 75, 78, 79, 84, 86, 90,
92, 92, 96, 97, 97, 97, 97, 47, 45, 45, 44, 44, 45, 45, 47, 50, 55,
56, 60, 62, 66, 69, 70, 75, 77, 79, 83, 84, 89, 91, 95, 97, 97, 100, 99,
101, 104, 107, 110, 53, 51, 50, 49, 49, 50, 49, 51, 54, 59, 60, 65, 67,
71, 75, 76, 82, 84, 87, 91, 92, 97, 100, 104, 105, 106, 110, 113, 114,
112, 111, 110, 62, 59, 58, 57, 57, 57, 56, 58, 61, 65, 66, 71, 74, 78,
82, 83, 90, 92, 95, 100, 102, 108, 110, 115, 117, 117, 120, 118, 116,
119, 123, 126, 65, 62, 61, 59, 59, 59, 58, 60, 63, 67, 68, 73, 76, 79,
84, 85, 92, 94, 98, 103, 105, 111, 113, 118, 120, 121, 125, 128, 132,
130, 128, 126, 79, 75, 74, 72, 71, 71, 69, 71, 73, 77, 78, 84, 86, 90,
95, 96, 103, 106, 110, 116, 118, 125, 128, 133, 136, 136, 141, 139, 135,
136, 140, 144, 82, 78, 76, 74, 73, 73, 71, 73, 76, 79, 80, 86, 88, 92,
97, 98, 106, 108, 112, 118, 120, 127, 131, 136, 139, 139, 144, 145, 150,
151, 147, 144, 88, 83, 82, 79, 79, 78, 76, 76, 81, 82, 85, 89, 91, 97,
98, 104, 107, 111, 117, 119, 127, 129, 135, 140, 145, 148, 148, 153,
153, 154, 159, 163, 90, 86, 85, 82, 81, 80, 79, 78, 81, 83, 87, 88, 93,
94, 101, 101, 108, 110, 116, 119, 124, 129, 134, 139, 142, 150, 153,
157, 157, 163, 163, 163, 93, 88, 88, 84, 84, 82, 83, 80, 80, 86, 86, 91,
91, 97, 98, 105, 105, 112, 113, 121, 122, 130, 130, 140, 140, 149, 150,
161, 162, 166, 167, 173,
/* Size 4x16 */
31, 36, 62, 88, 32, 35, 58, 82, 32, 36, 57, 79, 33, 38, 56, 76, 34, 42,
61, 81, 34, 48, 66, 85, 39, 51, 74, 91, 44, 56, 82, 98, 49, 60, 90, 107,
54, 63, 95, 117, 60, 68, 102, 127, 68, 75, 110, 135, 75, 81, 117, 145,
79, 85, 120, 148, 84, 89, 116, 153, 88, 94, 123, 159,
/* Size 16x4 */
31, 32, 32, 33, 34, 34, 39, 44, 49, 54, 60, 68, 75, 79, 84, 88, 36, 35,
36, 38, 42, 48, 51, 56, 60, 63, 68, 75, 81, 85, 89, 94, 62, 58, 57, 56,
61, 66, 74, 82, 90, 95, 102, 110, 117, 120, 116, 123, 88, 82, 79, 76,
81, 85, 91, 98, 107, 117, 127, 135, 145, 148, 153, 159,
/* Size 8x32 */
32, 31, 35, 44, 53, 65, 82, 90, 31, 32, 35, 42, 51, 62, 78, 86, 31, 32,
34, 41, 50, 61, 76, 85, 31, 32, 34, 41, 49, 59, 74, 82, 31, 33, 35, 42,
49, 59, 73, 81, 32, 33, 36, 42, 50, 59, 73, 80, 32, 34, 37, 42, 49, 58,
71, 79, 32, 34, 39, 44, 51, 60, 73, 78, 34, 35, 41, 48, 54, 63, 76, 81,
35, 36, 45, 52, 59, 67, 79, 83, 36, 36, 46, 54, 60, 68, 80, 87, 39, 39,
48, 58, 65, 73, 86, 88, 41, 40, 49, 60, 67, 76, 88, 93, 44, 42, 51, 63,
71, 79, 92, 94, 47, 44, 53, 66, 75, 84, 97, 101, 48, 45, 54, 67, 76, 85,
98, 101, 53, 50, 57, 71, 82, 92, 106, 108, 55, 51, 59, 72, 84, 94, 108,

```

```

110, 58, 54, 61, 75, 87, 98, 112, 116, 63, 58, 65, 78, 91, 103, 118,
119, 65, 59, 66, 79, 92, 105, 120, 124, 71, 64, 71, 84, 97, 111, 127,
129, 74, 67, 73, 86, 100, 113, 131, 134, 79, 71, 77, 90, 104, 118, 136,
139, 82, 73, 79, 92, 105, 120, 139, 142, 82, 74, 79, 92, 106, 121, 139,
150, 87, 78, 83, 96, 110, 125, 144, 153, 89, 81, 83, 97, 113, 128, 145,
157, 92, 83, 84, 97, 114, 132, 150, 157, 94, 85, 85, 97, 112, 130, 151,
163, 97, 88, 86, 97, 111, 128, 147, 163, 99, 91, 87, 97, 110, 126, 144,
163,
/* Size 32x8 */
32, 31, 31, 31, 31, 32, 32, 32, 34, 35, 36, 39, 41, 44, 47, 48, 53, 55,
58, 63, 65, 71, 74, 79, 82, 82, 87, 89, 92, 94, 97, 99, 31, 32, 32, 32,
33, 33, 34, 34, 35, 36, 36, 39, 40, 42, 44, 45, 50, 51, 54, 58, 59, 64,
67, 71, 73, 74, 78, 81, 83, 85, 88, 91, 35, 35, 34, 34, 35, 36, 37, 39,
41, 45, 46, 48, 49, 51, 53, 54, 57, 59, 61, 65, 66, 71, 73, 77, 79, 79,
83, 83, 84, 85, 86, 87, 44, 42, 41, 41, 42, 42, 42, 44, 48, 52, 54, 58,
60, 63, 66, 67, 71, 72, 75, 78, 79, 84, 86, 90, 92, 92, 96, 97, 97, 97,
97, 97, 53, 51, 50, 49, 49, 50, 49, 51, 54, 59, 60, 65, 67, 71, 75, 76,
82, 84, 87, 91, 92, 97, 100, 104, 105, 106, 110, 113, 114, 112, 111,
110, 65, 62, 61, 59, 59, 59, 58, 60, 63, 67, 68, 73, 76, 79, 84, 85, 92,
94, 98, 103, 105, 111, 113, 118, 120, 121, 125, 128, 132, 130, 128, 126,
82, 78, 76, 74, 73, 73, 71, 73, 76, 79, 80, 86, 88, 92, 97, 98, 106,
108, 112, 118, 120, 127, 131, 136, 139, 139, 144, 145, 150, 151, 147,
144, 90, 86, 85, 82, 81, 80, 79, 78, 81, 83, 87, 88, 93, 94, 101, 101,
108, 110, 116, 119, 124, 129, 134, 139, 142, 150, 153, 157, 157, 163,
163, 163 },
{ /* Chroma */
/* Size 4x4 */
32, 45, 51, 61, 45, 54, 59, 65, 51, 59, 75, 81, 61, 65, 81, 97,
/* Size 8x8 */
31, 34, 46, 47, 50, 57, 61, 65, 34, 39, 47, 45, 48, 53, 57, 61, 46, 47,
52, 52, 54, 58, 61, 62, 47, 45, 52, 58, 62, 65, 68, 68, 50, 48, 54, 62,
68, 73, 77, 76, 57, 53, 58, 65, 73, 82, 86, 86, 61, 57, 61, 68, 77, 86,
91, 95, 65, 61, 62, 68, 76, 86, 95, 100,
/* Size 16x16 */
32, 31, 33, 36, 41, 49, 49, 50, 52, 54, 57, 61, 64, 67, 68, 70, 31, 31,
34, 39, 42, 47, 46, 47, 49, 51, 53, 57, 60, 62, 64, 66, 33, 34, 37, 42,
44, 47, 46, 46, 47, 49, 51, 55, 57, 59, 61, 63, 36, 39, 42, 47, 47, 48,
46, 46, 47, 48, 50, 53, 55, 57, 59, 61, 41, 42, 44, 47, 48, 50, 49, 50,
50, 52, 53, 56, 58, 60, 61, 60, 49, 47, 47, 48, 50, 53, 53, 54, 54, 55,
56, 59, 61, 63, 64, 64, 49, 46, 46, 46, 49, 53, 55, 57, 59, 60, 61, 64,
66, 67, 67, 67, 50, 47, 46, 46, 50, 54, 57, 61, 63, 64, 66, 69, 70, 72,
71, 71, 52, 49, 47, 47, 50, 54, 59, 63, 66, 68, 70, 73, 75, 77, 75, 75,
54, 51, 49, 48, 52, 55, 60, 64, 68, 71, 73, 76, 79, 80, 79, 79, 57, 53,
51, 50, 53, 56, 61, 66, 70, 73, 76, 80, 82, 84, 83, 84, 61, 57, 55, 53,
56, 59, 64, 69, 73, 76, 80, 84, 87, 89, 88, 88, 64, 60, 57, 55, 58, 61,
66, 70, 75, 79, 82, 87, 91, 93, 93, 93, 67, 62, 59, 57, 60, 63, 67, 72,
77, 80, 84, 89, 93, 95, 96, 97, 68, 64, 61, 59, 61, 64, 67, 71, 75, 79,
83, 88, 93, 96, 99, 100, 70, 66, 63, 61, 60, 64, 67, 71, 75, 79, 84, 88,
93, 97, 100, 102,
/* Size 32x32 */

```

32, 31, 31, 30, 33, 33, 36, 38, 41, 47, 49, 48, 49, 49, 50, 50, 52, 53,  
 54, 56, 57, 60, 61, 63, 64, 65, 67, 67, 68, 69, 70, 71, 31, 31, 31, 31,  
 34, 34, 38, 40, 42, 46, 47, 47, 47, 47, 48, 48, 50, 50, 52, 54, 54, 57,  
 58, 60, 61, 61, 63, 64, 65, 65, 66, 67, 31, 31, 31, 31, 34, 35, 39, 40,  
 42, 46, 47, 46, 46, 46, 47, 47, 49, 50, 51, 53, 53, 56, 57, 59, 60, 60,  
 62, 63, 64, 65, 66, 67, 30, 31, 31, 32, 34, 35, 40, 41, 42, 45, 46, 45,  
 45, 45, 46, 46, 47, 48, 49, 51, 52, 54, 55, 57, 58, 58, 60, 61, 62, 62,  
 63, 64, 33, 34, 34, 34, 37, 38, 42, 43, 44, 46, 47, 46, 46, 45, 46, 46,  
 47, 48, 49, 51, 51, 53, 55, 56, 57, 57, 59, 60, 61, 62, 63, 64, 33, 34,  
 35, 35, 38, 39, 43, 44, 45, 47, 47, 46, 46, 45, 46, 46, 47, 48, 49, 51,  
 51, 53, 54, 56, 57, 57, 59, 60, 60, 61, 62, 62, 36, 38, 39, 40, 42, 43,  
 47, 47, 47, 47, 48, 46, 46, 45, 46, 46, 47, 47, 48, 49, 50, 52, 53, 54,  
 55, 55, 57, 58, 59, 60, 61, 62, 38, 40, 40, 41, 43, 44, 47, 47, 48, 48,  
 49, 48, 47, 47, 47, 47, 48, 49, 49, 51, 51, 53, 54, 55, 56, 56, 58, 58,  
 58, 59, 60, 60, 41, 42, 42, 42, 44, 45, 47, 48, 48, 50, 50, 49, 49, 49,  
 50, 50, 50, 51, 52, 53, 53, 55, 56, 57, 58, 58, 60, 61, 61, 61, 60, 60,  
 47, 46, 46, 45, 46, 47, 47, 48, 50, 52, 52, 52, 52, 52, 53, 53, 53, 54,  
 55, 55, 56, 58, 58, 60, 60, 61, 62, 61, 61, 62, 63, 64, 49, 47, 47, 46,  
 47, 47, 48, 49, 50, 52, 53, 53, 53, 53, 54, 54, 54, 55, 55, 56, 56, 58,  
 59, 60, 61, 61, 63, 63, 64, 64, 64, 64, 48, 47, 46, 45, 46, 46, 46, 48,  
 49, 52, 53, 54, 55, 55, 56, 56, 57, 58, 58, 59, 60, 61, 62, 63, 64, 64,  
 66, 65, 65, 65, 66, 67, 49, 47, 46, 45, 46, 46, 46, 47, 49, 52, 53, 55,  
 55, 57, 57, 58, 59, 59, 60, 61, 61, 63, 64, 65, 66, 66, 67, 67, 67, 68,  
 67, 67, 49, 47, 46, 45, 45, 45, 45, 47, 49, 52, 53, 55, 57, 58, 59, 60,  
 61, 62, 62, 63, 63, 65, 66, 67, 68, 68, 69, 70, 69, 68, 69, 70, 50, 48,  
 47, 46, 46, 46, 46, 47, 50, 53, 54, 56, 57, 59, 61, 61, 63, 64, 64, 66,  
 66, 68, 69, 70, 70, 71, 72, 70, 71, 72, 71, 70, 50, 48, 47, 46, 46, 46,  
 46, 47, 50, 53, 54, 56, 58, 60, 61, 61, 63, 64, 65, 66, 67, 68, 69, 71,  
 71, 71, 73, 74, 73, 72, 73, 74, 52, 50, 49, 47, 47, 47, 47, 48, 50, 53,  
 54, 57, 59, 61, 63, 63, 66, 67, 68, 70, 70, 72, 73, 75, 75, 75, 77, 75,  
 75, 76, 75, 74, 53, 50, 50, 48, 48, 48, 47, 49, 51, 54, 55, 58, 59, 62,  
 64, 64, 67, 68, 69, 71, 71, 73, 74, 76, 77, 77, 78, 78, 78, 76, 77, 78,  
 54, 52, 51, 49, 49, 49, 48, 49, 52, 55, 55, 58, 60, 62, 64, 65, 68, 69,  
 71, 73, 73, 75, 76, 78, 79, 79, 80, 80, 79, 80, 79, 78, 56, 54, 53, 51,  
 51, 51, 49, 51, 53, 55, 56, 59, 61, 63, 66, 66, 70, 71, 73, 75, 76, 78,  
 79, 81, 82, 82, 83, 81, 83, 81, 81, 82, 57, 54, 53, 52, 51, 51, 50, 51,  
 53, 56, 56, 60, 61, 63, 66, 67, 70, 71, 73, 76, 76, 79, 80, 82, 82, 83,  
 84, 85, 83, 84, 84, 82, 60, 57, 56, 54, 53, 53, 52, 53, 55, 58, 58, 61,  
 63, 65, 68, 68, 72, 73, 75, 78, 79, 82, 83, 85, 86, 86, 88, 86, 87, 86,  
 85, 86, 61, 58, 57, 55, 55, 54, 53, 54, 56, 58, 59, 62, 64, 66, 69, 69,  
 73, 74, 76, 79, 80, 83, 84, 86, 87, 88, 89, 89, 88, 88, 88, 86, 63, 60,  
 59, 57, 56, 56, 54, 55, 57, 60, 60, 63, 65, 67, 70, 71, 75, 76, 78, 81,  
 82, 85, 86, 89, 90, 90, 92, 91, 91, 90, 89, 91, 64, 61, 60, 58, 57, 57,  
 55, 56, 58, 60, 61, 64, 66, 68, 70, 71, 75, 77, 79, 82, 82, 86, 87, 90,  
 91, 91, 93, 93, 93, 92, 93, 91, 65, 61, 60, 58, 57, 57, 55, 56, 58, 61,  
 61, 64, 66, 68, 71, 71, 75, 77, 79, 82, 83, 86, 88, 90, 91, 91, 93, 94,  
 95, 95, 93, 95, 67, 63, 62, 60, 59, 59, 57, 58, 60, 62, 63, 66, 67, 69,  
 72, 73, 77, 78, 80, 83, 84, 88, 89, 92, 93, 93, 95, 95, 96, 96, 97, 95,  
 67, 64, 63, 61, 60, 60, 58, 58, 61, 61, 63, 65, 67, 70, 70, 74, 75, 78,  
 80, 81, 85, 86, 89, 91, 93, 94, 95, 97, 97, 98, 98, 100, 68, 65, 64, 62,

```

61, 60, 59, 58, 61, 61, 64, 65, 67, 69, 71, 73, 75, 78, 79, 83, 83, 87,
88, 91, 93, 95, 96, 97, 99, 98, 100, 100, 69, 65, 65, 62, 62, 61, 60,
59, 61, 62, 64, 65, 68, 68, 72, 72, 76, 76, 80, 81, 84, 86, 88, 90, 92,
95, 96, 98, 98, 100, 100, 101, 70, 66, 66, 63, 63, 62, 61, 60, 60, 63,
64, 66, 67, 69, 71, 73, 75, 77, 79, 81, 84, 85, 88, 89, 93, 93, 97, 98,
100, 100, 102, 101, 71, 67, 67, 64, 64, 62, 62, 60, 60, 64, 64, 67, 67,
70, 70, 74, 74, 78, 78, 82, 82, 86, 86, 91, 91, 95, 95, 100, 100, 101,
101, 104,
/* Size 4x8 */
31, 47, 53, 63, 36, 47, 50, 59, 46, 52, 55, 61, 45, 53, 63, 70, 49, 55,
71, 77, 54, 58, 77, 86, 59, 61, 81, 94, 63, 65, 80, 95,
/* Size 8x4 */
31, 36, 46, 45, 49, 54, 59, 63, 47, 47, 52, 53, 55, 58, 61, 65, 53, 50,
55, 63, 71, 77, 81, 80, 63, 59, 61, 70, 77, 86, 94, 95,
/* Size 8x16 */
32, 33, 45, 49, 52, 57, 64, 68, 31, 34, 45, 46, 49, 53, 60, 64, 33, 37,
46, 45, 47, 51, 57, 61, 37, 43, 47, 45, 47, 50, 55, 59, 42, 44, 49, 49,
50, 53, 58, 60, 49, 47, 52, 53, 54, 57, 61, 63, 48, 46, 51, 57, 59, 61,
66, 67, 50, 46, 52, 59, 63, 66, 71, 71, 52, 47, 53, 61, 66, 71, 75, 74,
54, 49, 54, 62, 68, 73, 79, 79, 57, 51, 55, 64, 70, 76, 83, 83, 61, 55,
58, 66, 73, 80, 87, 87, 64, 57, 60, 68, 75, 83, 91, 91, 66, 59, 61, 69,
77, 84, 93, 95, 68, 61, 61, 68, 77, 86, 94, 97, 70, 63, 61, 67, 75, 83,
92, 98,
/* Size 16x8 */
32, 31, 33, 37, 42, 49, 48, 50, 52, 54, 57, 61, 64, 66, 68, 70, 33, 34,
37, 43, 44, 47, 46, 46, 47, 49, 51, 55, 57, 59, 61, 63, 45, 45, 46, 47,
49, 52, 51, 52, 53, 54, 55, 58, 60, 61, 61, 61, 49, 46, 45, 45, 49, 53,
57, 59, 61, 62, 64, 66, 68, 69, 68, 67, 52, 49, 47, 47, 50, 54, 59, 63,
66, 68, 70, 73, 75, 77, 77, 75, 57, 53, 51, 50, 53, 57, 61, 66, 71, 73,
76, 80, 83, 84, 86, 83, 64, 60, 57, 55, 58, 61, 66, 71, 75, 79, 83, 87,
91, 93, 94, 92, 68, 64, 61, 59, 60, 63, 67, 71, 74, 79, 83, 87, 91, 95,
97, 98,
/* Size 16x32 */
32, 31, 33, 37, 45, 48, 49, 50, 52, 56, 57, 63, 64, 67, 68, 68, 31, 31,
34, 38, 45, 47, 47, 48, 50, 53, 54, 60, 61, 63, 64, 65, 31, 32, 34, 39,
45, 46, 46, 47, 49, 52, 53, 59, 60, 62, 64, 65, 30, 32, 35, 40, 44, 46,
45, 46, 48, 51, 52, 57, 58, 60, 61, 62, 33, 35, 37, 42, 46, 47, 45, 46,
47, 50, 51, 56, 57, 60, 61, 62, 33, 36, 38, 43, 46, 47, 46, 46, 47, 50,
51, 56, 57, 59, 60, 60, 37, 40, 43, 47, 47, 47, 45, 46, 47, 49, 50, 54,
55, 57, 59, 61, 39, 41, 43, 47, 48, 48, 47, 47, 48, 50, 51, 55, 56, 57,
58, 59, 42, 43, 44, 47, 49, 50, 49, 50, 50, 53, 53, 57, 58, 60, 60, 59,
47, 46, 46, 48, 51, 52, 53, 53, 53, 55, 56, 60, 61, 61, 61, 62, 49, 46,
47, 48, 52, 53, 53, 54, 54, 56, 57, 60, 61, 63, 63, 62, 48, 46, 46, 47,
51, 53, 56, 56, 57, 59, 60, 64, 64, 65, 64, 65, 48, 45, 46, 46, 51, 53,
57, 57, 59, 61, 61, 65, 66, 66, 67, 65, 49, 45, 45, 46, 51, 53, 58, 59,
61, 63, 64, 67, 68, 70, 67, 68, 50, 46, 46, 46, 52, 54, 59, 61, 63, 65,
66, 70, 71, 70, 71, 68, 50, 46, 46, 46, 52, 54, 59, 61, 64, 66, 67, 71,
71, 73, 71, 72, 52, 48, 47, 47, 53, 54, 61, 63, 66, 70, 71, 75, 75, 75,
74, 72, 53, 49, 48, 48, 53, 55, 61, 64, 67, 71, 72, 76, 77, 77, 75, 76,
54, 50, 49, 49, 54, 55, 62, 65, 68, 72, 73, 78, 79, 80, 79, 76, 56, 51,

```



```

51, 50, 55, 56, 63, 66, 70, 74, 76, 81, 82, 81, 80, 80, 57, 52, 51, 50,
55, 56, 64, 66, 70, 75, 76, 82, 83, 85, 83, 80, 60, 54, 54, 52, 57, 58,
65, 68, 72, 77, 79, 85, 86, 86, 85, 84, 61, 56, 55, 53, 58, 59, 66, 69,
73, 79, 80, 86, 87, 89, 87, 84, 63, 57, 56, 55, 59, 60, 67, 70, 75, 80,
82, 89, 90, 91, 89, 89, 64, 58, 57, 56, 60, 61, 68, 71, 75, 81, 83, 90,
91, 93, 91, 89, 64, 59, 58, 56, 60, 61, 68, 71, 75, 81, 83, 90, 91, 94,
94, 93, 66, 60, 59, 57, 61, 63, 69, 72, 77, 82, 84, 92, 93, 94, 95, 93,
67, 61, 60, 58, 61, 63, 69, 70, 78, 80, 85, 90, 93, 96, 97, 97, 68, 62,
61, 59, 61, 64, 68, 71, 77, 79, 86, 88, 94, 96, 97, 98, 69, 63, 62, 59,
61, 65, 68, 72, 76, 80, 85, 88, 94, 95, 99, 99, 70, 63, 63, 60, 61, 66,
67, 73, 75, 81, 83, 89, 92, 97, 98, 99, 70, 64, 64, 61, 61, 67, 67, 74,
74, 82, 82, 90, 90, 98, 98, 102,

```

```
/* Size 32x16 */
```

```

32, 31, 31, 30, 33, 33, 37, 39, 42, 47, 49, 48, 48, 49, 50, 50, 52, 53,
54, 56, 57, 60, 61, 63, 64, 64, 66, 67, 68, 69, 70, 70, 31, 31, 32, 32,
35, 36, 40, 41, 43, 46, 46, 46, 45, 45, 46, 46, 48, 49, 50, 51, 52, 54,
56, 57, 58, 59, 60, 61, 62, 63, 63, 64, 33, 34, 34, 35, 37, 38, 43, 43,
44, 46, 47, 46, 46, 45, 46, 46, 47, 48, 49, 51, 51, 54, 55, 56, 57, 58,
59, 60, 61, 62, 63, 64, 37, 38, 39, 40, 42, 43, 47, 47, 47, 48, 48, 47,
46, 46, 46, 46, 47, 48, 49, 50, 50, 52, 53, 55, 56, 56, 57, 58, 59, 59,
60, 61, 45, 45, 45, 44, 46, 46, 47, 48, 49, 51, 52, 51, 51, 51, 52, 52,
53, 53, 54, 55, 55, 57, 58, 59, 60, 60, 61, 61, 61, 61, 61, 61, 48, 47,
46, 46, 47, 47, 47, 48, 50, 52, 53, 53, 53, 54, 54, 54, 55, 55, 56,
56, 58, 59, 60, 61, 61, 63, 63, 64, 65, 66, 67, 49, 47, 46, 45, 45, 46,
45, 47, 49, 53, 53, 56, 57, 58, 59, 59, 61, 61, 62, 63, 64, 65, 66, 67,
68, 68, 69, 69, 68, 68, 67, 67, 50, 48, 47, 46, 46, 46, 46, 47, 50, 53,
54, 56, 57, 59, 61, 61, 63, 64, 65, 66, 66, 68, 69, 70, 71, 71, 72, 70,
71, 72, 73, 74, 52, 50, 49, 48, 47, 47, 47, 48, 50, 53, 54, 57, 59, 61,
63, 64, 66, 67, 68, 70, 70, 72, 73, 75, 75, 75, 77, 78, 77, 76, 75, 74,
56, 53, 52, 51, 50, 50, 49, 50, 53, 55, 56, 59, 61, 63, 65, 66, 70, 71,
72, 74, 75, 77, 79, 80, 81, 81, 82, 80, 79, 80, 81, 82, 57, 54, 53, 52,
51, 51, 50, 51, 53, 56, 57, 60, 61, 64, 66, 67, 71, 72, 73, 76, 76, 79,
80, 82, 83, 83, 84, 85, 86, 85, 83, 82, 63, 60, 59, 57, 56, 56, 54, 55,
57, 60, 60, 64, 65, 67, 70, 71, 75, 76, 78, 81, 82, 85, 86, 89, 90, 90,
92, 90, 88, 88, 89, 90, 64, 61, 60, 58, 57, 57, 55, 56, 58, 61, 61, 64,
66, 68, 71, 71, 75, 77, 79, 82, 83, 86, 87, 90, 91, 91, 93, 93, 94, 94,
92, 90, 67, 63, 62, 60, 60, 59, 57, 57, 60, 61, 63, 65, 66, 70, 70, 73,
75, 77, 80, 81, 85, 86, 89, 91, 93, 94, 94, 96, 96, 95, 97, 98, 68, 64,
64, 61, 61, 60, 59, 58, 60, 61, 63, 64, 67, 67, 71, 71, 74, 75, 79, 80,
83, 85, 87, 89, 91, 94, 95, 97, 97, 99, 98, 98, 68, 65, 65, 62, 62, 60,
61, 59, 59, 62, 62, 65, 65, 68, 68, 72, 72, 76, 76, 80, 80, 84, 84, 89,
89, 93, 93, 97, 98, 99, 99, 102,

```

```
/* Size 4x16 */
```

```

31, 48, 56, 67, 32, 46, 52, 62, 35, 47, 50, 60, 40, 47, 49, 57, 43, 50,
53, 60, 46, 53, 56, 63, 45, 53, 61, 66, 46, 54, 65, 70, 48, 54, 70, 75,
50, 55, 72, 80, 52, 56, 75, 85, 56, 59, 79, 89, 58, 61, 81, 93, 60, 63,
82, 94, 62, 64, 79, 96, 63, 66, 81, 97,

```

```
/* Size 16x4 */
```

```

31, 32, 35, 40, 43, 46, 45, 46, 48, 50, 52, 56, 58, 60, 62, 63, 48, 46,
47, 47, 50, 53, 53, 54, 54, 55, 56, 59, 61, 63, 64, 66, 56, 52, 50, 49,

```

```
53, 56, 61, 65, 70, 72, 75, 79, 81, 82, 79, 81, 67, 62, 60, 57, 60, 63,
66, 70, 75, 80, 85, 89, 93, 94, 96, 97,
```

```
/* Size 8x32 */
```

```
32, 33, 45, 49, 52, 57, 64, 68, 31, 34, 45, 47, 50, 54, 61, 64, 31, 34,
45, 46, 49, 53, 60, 64, 30, 35, 44, 45, 48, 52, 58, 61, 33, 37, 46, 45,
47, 51, 57, 61, 33, 38, 46, 46, 47, 51, 57, 60, 37, 43, 47, 45, 47, 50,
55, 59, 39, 43, 48, 47, 48, 51, 56, 58, 42, 44, 49, 49, 50, 53, 58, 60,
47, 46, 51, 53, 53, 56, 61, 61, 49, 47, 52, 53, 54, 57, 61, 63, 48, 46,
51, 56, 57, 60, 64, 64, 48, 46, 51, 57, 59, 61, 66, 67, 49, 45, 51, 58,
61, 64, 68, 67, 50, 46, 52, 59, 63, 66, 71, 71, 50, 46, 52, 59, 64, 67,
71, 71, 52, 47, 53, 61, 66, 71, 75, 74, 53, 48, 53, 61, 67, 72, 77, 75,
54, 49, 54, 62, 68, 73, 79, 79, 56, 51, 55, 63, 70, 76, 82, 80, 57, 51,
55, 64, 70, 76, 83, 83, 60, 54, 57, 65, 72, 79, 86, 85, 61, 55, 58, 66,
73, 80, 87, 87, 63, 56, 59, 67, 75, 82, 90, 89, 64, 57, 60, 68, 75, 83,
91, 91, 64, 58, 60, 68, 75, 83, 91, 94, 66, 59, 61, 69, 77, 84, 93, 95,
67, 60, 61, 69, 78, 85, 93, 97, 68, 61, 61, 68, 77, 86, 94, 97, 69, 62,
61, 68, 76, 85, 94, 99, 70, 63, 61, 67, 75, 83, 92, 98, 70, 64, 61, 67,
74, 82, 90, 98,
```

```
/* Size 32x8 */
```

```
32, 31, 31, 30, 33, 33, 37, 39, 42, 47, 49, 48, 48, 49, 50, 50, 52, 53,
54, 56, 57, 60, 61, 63, 64, 64, 66, 67, 68, 69, 70, 70, 33, 34, 34, 35,
37, 38, 43, 43, 44, 46, 47, 46, 46, 45, 46, 46, 47, 48, 49, 51, 51, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 45, 45, 45, 44, 46, 46, 47, 48,
49, 51, 52, 51, 51, 51, 52, 52, 53, 53, 54, 55, 55, 57, 58, 59, 60, 60,
61, 61, 61, 61, 61, 61, 49, 47, 46, 45, 45, 46, 45, 47, 49, 53, 53, 56,
57, 58, 59, 59, 61, 61, 62, 63, 64, 65, 66, 67, 68, 68, 69, 69, 68, 68,
67, 67, 52, 50, 49, 48, 47, 47, 47, 48, 50, 53, 54, 57, 59, 61, 63, 64,
66, 67, 68, 70, 70, 72, 73, 75, 75, 75, 77, 78, 77, 76, 75, 74, 57, 54,
53, 52, 51, 51, 50, 51, 53, 56, 57, 60, 61, 64, 66, 67, 71, 72, 73, 76,
76, 79, 80, 82, 83, 83, 84, 85, 86, 85, 83, 82, 64, 61, 60, 58, 57, 57,
55, 56, 58, 61, 61, 64, 66, 68, 71, 71, 75, 77, 79, 82, 83, 86, 87, 90,
91, 91, 93, 93, 94, 94, 92, 90, 68, 64, 64, 61, 61, 60, 59, 58, 60, 61,
63, 64, 67, 67, 71, 71, 74, 75, 79, 80, 83, 85, 87, 89, 91, 94, 95, 97,
97, 99, 98, 98 },
```

```
},
```

```
{
```

```
{ /* Luma */
```

```
/* Size 4x4 */
```

```
32, 34, 53, 75, 34, 49, 64, 81, 53, 64, 91, 112, 75, 81, 112, 140,
```

```
/* Size 8x8 */
```

```
32, 32, 34, 39, 50, 62, 76, 84, 32, 33, 35, 40, 48, 59, 71, 79, 34, 35,
39, 46, 53, 63, 74, 81, 39, 40, 46, 56, 65, 75, 86, 92, 50, 48, 53, 65,
78, 90, 101, 106, 62, 59, 63, 75, 90, 105, 118, 123, 76, 71, 74, 86,
101, 118, 134, 142, 84, 79, 81, 92, 106, 123, 142, 153,
```

```
/* Size 16x16 */
```

```
32, 31, 31, 32, 33, 36, 39, 44, 48, 54, 59, 66, 74, 81, 86, 91, 31, 32,
32, 32, 33, 35, 38, 42, 46, 51, 56, 63, 70, 77, 81, 86, 31, 32, 32, 33,
34, 35, 38, 41, 45, 49, 54, 60, 67, 73, 77, 82, 32, 32, 33, 34, 36, 37,
40, 42, 45, 49, 53, 59, 66, 71, 75, 80, 33, 33, 34, 36, 38, 42, 44, 46,
50, 53, 57, 63, 69, 74, 78, 80, 36, 35, 35, 37, 42, 48, 50, 54, 57, 60,
```

```

64, 69, 75, 80, 84, 85, 39, 38, 38, 40, 44, 50, 54, 58, 61, 65, 69, 74,
80, 85, 89, 91, 44, 42, 41, 42, 46, 54, 58, 63, 67, 71, 75, 80, 86, 91,
95, 97, 48, 46, 45, 45, 50, 57, 61, 67, 71, 76, 80, 86, 93, 98, 101,
104, 54, 51, 49, 49, 53, 60, 65, 71, 76, 82, 87, 93, 100, 105, 109, 112,
59, 56, 54, 53, 57, 64, 69, 75, 80, 87, 92, 99, 106, 112, 116, 120, 66,
63, 60, 59, 63, 69, 74, 80, 86, 93, 99, 107, 115, 121, 125, 129, 74, 70,
67, 66, 69, 75, 80, 86, 93, 100, 106, 115, 123, 130, 135, 138, 81, 77,
73, 71, 74, 80, 85, 91, 98, 105, 112, 121, 130, 137, 142, 148, 86, 81,
77, 75, 78, 84, 89, 95, 101, 109, 116, 125, 135, 142, 147, 153, 91, 86,
82, 80, 80, 85, 91, 97, 104, 112, 120, 129, 138, 148, 153, 159,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 32, 32, 33, 34, 36, 36, 39, 41, 44, 46, 48, 52,
54, 58, 59, 65, 66, 71, 74, 80, 81, 83, 86, 89, 91, 93, 31, 32, 32, 32,
32, 32, 32, 32, 33, 34, 35, 35, 38, 39, 42, 44, 46, 50, 51, 56, 56, 62,
63, 68, 71, 76, 77, 78, 82, 84, 86, 88, 31, 32, 32, 32, 32, 32, 32, 32,
33, 34, 35, 35, 38, 39, 42, 44, 46, 49, 51, 55, 56, 61, 63, 67, 70, 75,
77, 78, 81, 84, 86, 88, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34,
37, 38, 41, 42, 44, 48, 49, 53, 54, 59, 60, 65, 68, 72, 74, 75, 78, 80,
82, 84, 31, 32, 32, 32, 32, 33, 33, 33, 34, 34, 35, 35, 38, 39, 41, 43,
45, 48, 49, 53, 54, 59, 60, 65, 67, 72, 73, 74, 77, 80, 82, 84, 31, 32,
32, 32, 33, 33, 33, 34, 35, 35, 36, 36, 39, 40, 42, 44, 45, 48, 50, 53,
54, 59, 60, 64, 67, 71, 73, 74, 77, 79, 81, 83, 32, 32, 32, 32, 33, 33,
34, 35, 36, 36, 37, 38, 40, 40, 42, 44, 45, 48, 49, 53, 53, 58, 59, 63,
66, 70, 71, 72, 75, 78, 80, 83, 32, 32, 32, 32, 33, 34, 35, 35, 36, 37,
38, 38, 40, 41, 42, 44, 46, 48, 49, 53, 53, 58, 59, 63, 65, 69, 71, 72,
74, 77, 79, 80, 33, 33, 33, 33, 34, 35, 36, 36, 38, 39, 42, 42, 44, 45,
46, 48, 50, 52, 53, 57, 57, 62, 63, 67, 69, 73, 74, 75, 78, 79, 80, 81,
34, 34, 34, 33, 34, 35, 36, 37, 39, 39, 42, 43, 45, 46, 47, 49, 51, 53,
54, 58, 58, 63, 64, 68, 70, 74, 75, 76, 79, 81, 84, 86, 36, 35, 35, 34,
35, 36, 37, 38, 42, 42, 48, 48, 50, 51, 54, 55, 57, 59, 60, 63, 64, 68,
69, 73, 75, 79, 80, 81, 84, 85, 85, 86, 36, 35, 35, 34, 35, 36, 38, 38,
42, 43, 48, 49, 51, 52, 54, 55, 57, 59, 60, 64, 64, 68, 69, 73, 75, 79,
80, 81, 84, 86, 88, 91, 39, 38, 38, 37, 38, 39, 40, 40, 44, 45, 50, 51,
54, 55, 58, 59, 61, 64, 65, 68, 69, 73, 74, 78, 80, 84, 85, 86, 89, 91,
91, 91, 41, 39, 39, 38, 39, 40, 40, 41, 45, 46, 51, 52, 55, 56, 59, 61,
63, 65, 67, 70, 70, 75, 76, 80, 82, 86, 87, 88, 91, 92, 94, 96, 44, 42,
42, 41, 41, 42, 42, 42, 46, 47, 54, 54, 58, 59, 63, 65, 67, 70, 71, 75,
75, 79, 80, 84, 86, 90, 91, 92, 95, 97, 97, 97, 46, 44, 44, 42, 43, 44,
44, 44, 48, 49, 55, 55, 59, 61, 65, 67, 69, 72, 74, 77, 78, 82, 83, 87,
89, 93, 94, 95, 98, 98, 100, 103, 48, 46, 46, 44, 45, 45, 45, 46, 50,
51, 57, 57, 61, 63, 67, 69, 71, 74, 76, 80, 80, 85, 86, 90, 93, 96, 98,
99, 101, 104, 104, 103, 52, 50, 49, 48, 48, 48, 48, 48, 52, 53, 59, 59,
64, 65, 70, 72, 74, 78, 80, 84, 85, 90, 91, 95, 97, 101, 103, 104, 106,
106, 107, 110, 54, 51, 51, 49, 49, 50, 49, 49, 53, 54, 60, 60, 65, 67,
71, 74, 76, 80, 82, 86, 87, 92, 93, 97, 100, 104, 105, 106, 109, 112,
112, 110, 58, 56, 55, 53, 53, 53, 53, 53, 57, 58, 63, 64, 68, 70, 75,
77, 80, 84, 86, 91, 91, 97, 98, 103, 105, 110, 111, 112, 115, 114, 115,
118, 59, 56, 56, 54, 54, 54, 53, 53, 57, 58, 64, 64, 69, 70, 75, 78, 80,
85, 87, 91, 92, 98, 99, 103, 106, 110, 112, 113, 116, 119, 120, 119, 65,
62, 61, 59, 59, 59, 58, 58, 62, 63, 68, 68, 73, 75, 79, 82, 85, 90, 92,

```

```

97, 98, 105, 106, 111, 114, 118, 120, 121, 124, 123, 123, 126, 66, 63,
63, 60, 60, 60, 59, 59, 63, 64, 69, 69, 74, 76, 80, 83, 86, 91, 93, 98,
99, 106, 107, 112, 115, 119, 121, 122, 125, 128, 129, 126, 71, 68, 67,
65, 65, 64, 63, 63, 67, 68, 73, 73, 78, 80, 84, 87, 90, 95, 97, 103,
103, 111, 112, 117, 120, 125, 127, 128, 131, 132, 132, 135, 74, 71, 70,
68, 67, 67, 66, 65, 69, 70, 75, 75, 80, 82, 86, 89, 93, 97, 100, 105,
106, 114, 115, 120, 123, 128, 130, 131, 135, 135, 138, 136, 80, 76, 75,
72, 72, 71, 70, 69, 73, 74, 79, 79, 84, 86, 90, 93, 96, 101, 104, 110,
110, 118, 119, 125, 128, 134, 136, 137, 140, 142, 140, 144, 81, 77, 77,
74, 73, 73, 71, 71, 74, 75, 80, 80, 85, 87, 91, 94, 98, 103, 105, 111,
112, 120, 121, 127, 130, 136, 137, 139, 142, 145, 148, 144, 83, 78, 78,
75, 74, 74, 72, 72, 75, 76, 81, 81, 86, 88, 92, 95, 99, 104, 106, 112,
113, 121, 122, 128, 131, 137, 139, 140, 144, 148, 150, 155, 86, 82, 81,
78, 77, 77, 75, 74, 78, 79, 84, 84, 89, 91, 95, 98, 101, 106, 109, 115,
116, 124, 125, 131, 135, 140, 142, 144, 147, 149, 153, 155, 89, 84, 84,
80, 80, 79, 78, 77, 79, 81, 85, 86, 91, 92, 97, 98, 104, 106, 112, 114,
119, 123, 128, 132, 135, 142, 145, 148, 149, 153, 154, 159, 91, 86, 86,
82, 82, 81, 80, 79, 80, 84, 85, 88, 91, 94, 97, 100, 104, 107, 112, 115,
120, 123, 129, 132, 138, 140, 148, 150, 153, 154, 159, 159, 93, 88, 88,
84, 84, 83, 83, 80, 81, 86, 86, 91, 91, 96, 97, 103, 103, 110, 110, 118,
119, 126, 126, 135, 136, 144, 144, 155, 155, 159, 159, 164,
/* Size 4x8 */
32, 35, 51, 77, 32, 36, 50, 72, 34, 42, 54, 75, 38, 51, 67, 87, 48, 59,
80, 103, 60, 68, 92, 119, 72, 79, 104, 135, 81, 86, 112, 144,
/* Size 8x4 */
32, 32, 34, 38, 48, 60, 72, 81, 35, 36, 42, 51, 59, 68, 79, 86, 51, 50,
54, 67, 80, 92, 104, 112, 77, 72, 75, 87, 103, 119, 135, 144,
/* Size 8x16 */
32, 31, 33, 40, 51, 65, 79, 87, 31, 32, 33, 39, 49, 61, 74, 82, 31, 32,
34, 38, 47, 59, 71, 79, 32, 33, 36, 40, 48, 58, 69, 77, 33, 34, 38, 44,
52, 62, 72, 78, 36, 35, 42, 51, 58, 68, 78, 84, 39, 38, 44, 54, 63, 73,
84, 89, 44, 41, 46, 59, 69, 79, 90, 96, 48, 45, 50, 62, 74, 85, 96, 103,
53, 49, 53, 66, 79, 92, 103, 111, 58, 54, 57, 70, 84, 98, 110, 118, 66,
60, 63, 75, 90, 106, 119, 126, 74, 67, 69, 81, 97, 113, 128, 134, 81,
73, 75, 86, 102, 120, 135, 143, 86, 78, 78, 90, 106, 124, 140, 147, 91,
82, 80, 90, 103, 119, 137, 151,
/* Size 16x8 */
32, 31, 31, 32, 33, 36, 39, 44, 48, 53, 58, 66, 74, 81, 86, 91, 31, 32,
32, 33, 34, 35, 38, 41, 45, 49, 54, 60, 67, 73, 78, 82, 33, 33, 34, 36,
38, 42, 44, 46, 50, 53, 57, 63, 69, 75, 78, 80, 40, 39, 38, 40, 44, 51,
54, 59, 62, 66, 70, 75, 81, 86, 90, 90, 51, 49, 47, 48, 52, 58, 63, 69,
74, 79, 84, 90, 97, 102, 106, 103, 65, 61, 59, 58, 62, 68, 73, 79, 85,
92, 98, 106, 113, 120, 124, 119, 79, 74, 71, 69, 72, 78, 84, 90, 96,
103, 110, 119, 128, 135, 140, 137, 87, 82, 79, 77, 78, 84, 89, 96, 103,
111, 118, 126, 134, 143, 147, 151,
/* Size 16x32 */
32, 31, 31, 32, 33, 36, 40, 44, 51, 53, 65, 66, 79, 81, 87, 90, 31, 32,
32, 32, 33, 35, 39, 42, 49, 51, 62, 63, 75, 77, 83, 85, 31, 32, 32, 32,
33, 35, 39, 42, 49, 51, 61, 62, 74, 76, 82, 85, 31, 32, 32, 33, 33, 34,
38, 41, 47, 49, 59, 60, 72, 74, 79, 81, 31, 32, 32, 33, 34, 35, 38, 41,

```

```

47, 49, 59, 60, 71, 73, 79, 81, 32, 32, 33, 34, 35, 36, 39, 42, 48, 50,
59, 60, 71, 72, 78, 80, 32, 32, 33, 35, 36, 37, 40, 42, 48, 49, 58, 59,
69, 71, 77, 80, 32, 33, 33, 35, 36, 38, 41, 42, 48, 49, 58, 59, 69, 70,
75, 77, 33, 33, 34, 36, 38, 41, 44, 46, 52, 53, 62, 63, 72, 74, 78, 78,
34, 34, 34, 37, 39, 42, 45, 48, 53, 54, 63, 64, 73, 75, 80, 83, 36, 34,
35, 38, 42, 48, 51, 54, 58, 60, 68, 69, 78, 80, 84, 83, 36, 35, 35, 38,
42, 48, 51, 54, 59, 60, 68, 69, 79, 80, 85, 87, 39, 37, 38, 40, 44, 50,
54, 58, 63, 65, 73, 74, 84, 85, 89, 88, 40, 38, 39, 41, 45, 51, 56, 59,
65, 67, 75, 76, 85, 87, 90, 93, 44, 41, 41, 43, 46, 53, 59, 63, 69, 71,
79, 80, 90, 91, 96, 93, 46, 43, 43, 44, 48, 55, 60, 65, 72, 73, 82, 83,
93, 94, 97, 100, 48, 45, 45, 46, 50, 56, 62, 67, 74, 76, 85, 86, 96, 98,
103, 100, 52, 48, 48, 49, 52, 59, 65, 70, 78, 80, 90, 91, 101, 103, 105,
107, 53, 49, 49, 50, 53, 60, 66, 71, 79, 82, 92, 93, 103, 105, 111, 107,
58, 53, 53, 53, 57, 63, 69, 74, 83, 86, 97, 98, 109, 111, 113, 115, 58,
54, 54, 54, 57, 63, 70, 75, 84, 87, 98, 99, 110, 112, 118, 115, 65, 60,
59, 58, 62, 68, 74, 79, 89, 92, 105, 106, 118, 119, 122, 123, 66, 61,
60, 59, 63, 69, 75, 80, 90, 93, 106, 107, 119, 121, 126, 123, 71, 65,
65, 63, 67, 73, 79, 84, 94, 97, 111, 112, 125, 127, 131, 132, 74, 68,
67, 66, 69, 75, 81, 86, 97, 100, 113, 115, 128, 130, 134, 132, 79, 72,
72, 70, 73, 79, 85, 90, 101, 104, 118, 119, 133, 135, 141, 140, 81, 74,
73, 71, 75, 80, 86, 91, 102, 105, 120, 121, 135, 137, 143, 140, 82, 75,
74, 72, 75, 81, 87, 92, 103, 106, 121, 122, 136, 139, 147, 151, 86, 78,
78, 75, 78, 84, 90, 95, 106, 109, 124, 125, 140, 142, 147, 151, 88, 81,
80, 77, 80, 86, 90, 98, 105, 112, 122, 127, 140, 144, 152, 155, 91, 83,
82, 79, 80, 88, 90, 100, 103, 114, 119, 130, 137, 148, 151, 155, 93, 85,
85, 81, 81, 90, 90, 102, 103, 117, 117, 134, 134, 151, 152, 160,
/* Size 32x16 */
32, 31, 31, 31, 31, 32, 32, 32, 33, 34, 36, 36, 39, 40, 44, 46, 48, 52,
53, 58, 58, 65, 66, 71, 74, 79, 81, 82, 86, 88, 91, 93, 31, 32, 32, 32,
32, 32, 32, 33, 33, 34, 34, 35, 37, 38, 41, 43, 45, 48, 49, 53, 54, 60,
61, 65, 68, 72, 74, 75, 78, 81, 83, 85, 31, 32, 32, 32, 32, 33, 33, 33,
34, 34, 35, 35, 38, 39, 41, 43, 45, 48, 49, 53, 54, 59, 60, 65, 67, 72,
73, 74, 78, 80, 82, 85, 32, 32, 32, 33, 33, 34, 35, 35, 36, 37, 38, 38,
40, 41, 43, 44, 46, 49, 50, 53, 54, 58, 59, 63, 66, 70, 71, 72, 75, 77,
79, 81, 33, 33, 33, 33, 34, 35, 36, 36, 38, 39, 42, 42, 44, 45, 46, 48,
50, 52, 53, 57, 57, 62, 63, 67, 69, 73, 75, 75, 78, 80, 80, 81, 36, 35,
35, 34, 35, 36, 37, 38, 41, 42, 48, 48, 50, 51, 53, 55, 56, 59, 60, 63,
63, 68, 69, 73, 75, 79, 80, 81, 84, 86, 88, 90, 40, 39, 39, 38, 38, 39,
40, 41, 44, 45, 51, 51, 54, 56, 59, 60, 62, 65, 66, 69, 70, 74, 75, 79,
81, 85, 86, 87, 90, 90, 90, 90, 44, 42, 42, 41, 41, 42, 42, 42, 46, 48,
54, 54, 58, 59, 63, 65, 67, 70, 71, 74, 75, 79, 80, 84, 86, 90, 91, 92,
95, 98, 100, 102, 51, 49, 49, 47, 47, 48, 48, 48, 52, 53, 58, 59, 63,
65, 69, 72, 74, 78, 79, 83, 84, 89, 90, 94, 97, 101, 102, 103, 106, 105,
103, 103, 53, 51, 51, 49, 49, 50, 49, 49, 53, 54, 60, 60, 65, 67, 71,
73, 76, 80, 82, 86, 87, 92, 93, 97, 100, 104, 105, 106, 109, 112, 114,
117, 65, 62, 61, 59, 59, 59, 58, 58, 62, 63, 68, 68, 73, 75, 79, 82, 85,
90, 92, 97, 98, 105, 106, 111, 113, 118, 120, 121, 124, 122, 119, 117,
66, 63, 62, 60, 60, 60, 59, 59, 63, 64, 69, 69, 74, 76, 80, 83, 86, 91,
93, 98, 99, 106, 107, 112, 115, 119, 121, 122, 125, 127, 130, 134, 79,
75, 74, 72, 71, 71, 69, 69, 72, 73, 78, 79, 84, 85, 90, 93, 96, 101,

```

```

103, 109, 110, 118, 119, 125, 128, 133, 135, 136, 140, 140, 137, 134,
81, 77, 76, 74, 73, 72, 71, 70, 74, 75, 80, 80, 85, 87, 91, 94, 98, 103,
105, 111, 112, 119, 121, 127, 130, 135, 137, 139, 142, 144, 148, 151,
87, 83, 82, 79, 79, 78, 77, 75, 78, 80, 84, 85, 89, 90, 96, 97, 103,
105, 111, 113, 118, 122, 126, 131, 134, 141, 143, 147, 147, 152, 151,
152, 90, 85, 85, 81, 81, 80, 80, 77, 78, 83, 83, 87, 88, 93, 93, 100,
100, 107, 107, 115, 115, 123, 123, 132, 132, 140, 140, 151, 151, 155,
155, 160,
/* Size 4x16 */
31, 36, 53, 81, 32, 35, 51, 76, 32, 35, 49, 73, 32, 37, 49, 71, 33, 41,
53, 74, 34, 48, 60, 80, 37, 50, 65, 85, 41, 53, 71, 91, 45, 56, 76, 98,
49, 60, 82, 105, 54, 63, 87, 112, 61, 69, 93, 121, 68, 75, 100, 130, 74,
80, 105, 137, 78, 84, 109, 142, 83, 88, 114, 148,
/* Size 16x4 */
31, 32, 32, 32, 33, 34, 37, 41, 45, 49, 54, 61, 68, 74, 78, 83, 36, 35,
35, 37, 41, 48, 50, 53, 56, 60, 63, 69, 75, 80, 84, 88, 53, 51, 49, 49,
53, 60, 65, 71, 76, 82, 87, 93, 100, 105, 109, 114, 81, 76, 73, 71, 74,
80, 85, 91, 98, 105, 112, 121, 130, 137, 142, 148,
/* Size 8x32 */
32, 31, 33, 40, 51, 65, 79, 87, 31, 32, 33, 39, 49, 62, 75, 83, 31, 32,
33, 39, 49, 61, 74, 82, 31, 32, 33, 38, 47, 59, 72, 79, 31, 32, 34, 38,
47, 59, 71, 79, 32, 33, 35, 39, 48, 59, 71, 78, 32, 33, 36, 40, 48, 58,
69, 77, 32, 33, 36, 41, 48, 58, 69, 75, 33, 34, 38, 44, 52, 62, 72, 78,
34, 34, 39, 45, 53, 63, 73, 80, 36, 35, 42, 51, 58, 68, 78, 84, 36, 35,
42, 51, 59, 68, 79, 85, 39, 38, 44, 54, 63, 73, 84, 89, 40, 39, 45, 56,
65, 75, 85, 90, 44, 41, 46, 59, 69, 79, 90, 96, 46, 43, 48, 60, 72, 82,
93, 97, 48, 45, 50, 62, 74, 85, 96, 103, 52, 48, 52, 65, 78, 90, 101,
105, 53, 49, 53, 66, 79, 92, 103, 111, 58, 53, 57, 69, 83, 97, 109, 113,
58, 54, 57, 70, 84, 98, 110, 118, 65, 59, 62, 74, 89, 105, 118, 122, 66,
60, 63, 75, 90, 106, 119, 126, 71, 65, 67, 79, 94, 111, 125, 131, 74,
67, 69, 81, 97, 113, 128, 134, 79, 72, 73, 85, 101, 118, 133, 141, 81,
73, 75, 86, 102, 120, 135, 143, 82, 74, 75, 87, 103, 121, 136, 147, 86,
78, 78, 90, 106, 124, 140, 147, 88, 80, 80, 90, 105, 122, 140, 152, 91,
82, 80, 90, 103, 119, 137, 151, 93, 85, 81, 90, 103, 117, 134, 152,
/* Size 32x8 */
32, 31, 31, 31, 31, 32, 32, 32, 33, 34, 36, 36, 39, 40, 44, 46, 48, 52,
53, 58, 58, 65, 66, 71, 74, 79, 81, 82, 86, 88, 91, 93, 31, 32, 32, 32,
32, 33, 33, 33, 34, 34, 35, 35, 38, 39, 41, 43, 45, 48, 49, 53, 54, 59,
60, 65, 67, 72, 73, 74, 78, 80, 82, 85, 33, 33, 33, 33, 34, 35, 36, 36,
38, 39, 42, 42, 44, 45, 46, 48, 50, 52, 53, 57, 57, 62, 63, 67, 69, 73,
75, 75, 78, 80, 80, 81, 40, 39, 39, 38, 38, 39, 40, 41, 44, 45, 51, 51,
54, 56, 59, 60, 62, 65, 66, 69, 70, 74, 75, 79, 81, 85, 86, 87, 90, 90,
90, 90, 51, 49, 49, 47, 47, 48, 48, 48, 52, 53, 58, 59, 63, 65, 69, 72,
74, 78, 79, 83, 84, 89, 90, 94, 97, 101, 102, 103, 106, 105, 103, 103,
65, 62, 61, 59, 59, 59, 58, 58, 62, 63, 68, 68, 73, 75, 79, 82, 85, 90,
92, 97, 98, 105, 106, 111, 113, 118, 120, 121, 124, 122, 119, 117, 79,
75, 74, 72, 71, 71, 69, 69, 72, 73, 78, 79, 84, 85, 90, 93, 96, 101,
103, 109, 110, 118, 119, 125, 128, 133, 135, 136, 140, 140, 137, 134,
87, 83, 82, 79, 79, 78, 77, 75, 78, 80, 84, 85, 89, 90, 96, 97, 103,
105, 111, 113, 118, 122, 126, 131, 134, 141, 143, 147, 147, 152, 151,

```

```

152 },
{ /* Chroma */
  /* Size 4x4 */
  32, 46, 49, 58, 46, 53, 55, 62, 49, 55, 70, 78, 58, 62, 78, 91,
  /* Size 8x8 */
  31, 34, 42, 47, 49, 54, 60, 64, 34, 39, 45, 46, 47, 51, 56, 59, 42, 45,
  48, 49, 50, 53, 57, 60, 47, 46, 49, 55, 58, 61, 65, 66, 49, 47, 50, 58,
  65, 69, 73, 74, 54, 51, 53, 61, 69, 76, 82, 83, 60, 56, 57, 65, 73, 82,
  89, 92, 64, 59, 60, 66, 74, 83, 92, 96,
  /* Size 16x16 */
  32, 31, 31, 35, 40, 49, 48, 49, 50, 52, 54, 57, 61, 64, 66, 68, 31, 31,
  32, 37, 41, 47, 47, 46, 48, 49, 51, 54, 57, 60, 62, 64, 31, 32, 34, 39,
  43, 46, 46, 45, 46, 47, 49, 52, 55, 57, 59, 61, 35, 37, 39, 44, 46, 47,
  46, 45, 46, 47, 48, 51, 53, 56, 57, 59, 40, 41, 43, 46, 48, 50, 49, 48,
  49, 49, 51, 53, 55, 57, 59, 59, 49, 47, 46, 47, 50, 53, 53, 53, 54, 54,
  55, 57, 59, 61, 62, 62, 48, 47, 46, 46, 49, 53, 54, 55, 56, 57, 58, 60,
  62, 64, 65, 65, 49, 46, 45, 45, 48, 53, 55, 58, 60, 61, 62, 64, 66, 68,
  69, 69, 50, 48, 46, 46, 49, 54, 56, 60, 61, 63, 65, 67, 69, 71, 72, 72,
  52, 49, 47, 47, 49, 54, 57, 61, 63, 66, 68, 71, 73, 75, 76, 77, 54, 51,
  49, 48, 51, 55, 58, 62, 65, 68, 71, 74, 76, 78, 80, 81, 57, 54, 52, 51,
  53, 57, 60, 64, 67, 71, 74, 77, 80, 83, 84, 85, 61, 57, 55, 53, 55, 59,
  62, 66, 69, 73, 76, 80, 84, 87, 89, 89, 64, 60, 57, 56, 57, 61, 64, 68,
  71, 75, 78, 83, 87, 90, 92, 94, 66, 62, 59, 57, 59, 62, 65, 69, 72, 76,
  80, 84, 89, 92, 94, 96, 68, 64, 61, 59, 59, 62, 65, 69, 72, 77, 81, 85,
  89, 94, 96, 98,
  /* Size 32x32 */
  32, 31, 31, 30, 31, 33, 35, 36, 40, 41, 49, 49, 48, 48, 49, 50, 50, 52,
  52, 54, 54, 57, 57, 60, 61, 63, 64, 65, 66, 67, 68, 69, 31, 31, 31, 31,
  32, 34, 37, 38, 41, 42, 47, 47, 47, 47, 47, 47, 48, 49, 50, 52, 52, 54,
  55, 57, 58, 60, 61, 61, 63, 64, 64, 65, 31, 31, 31, 31, 32, 35, 37, 39,
  41, 42, 47, 47, 47, 46, 46, 47, 48, 49, 49, 51, 51, 54, 54, 56, 57, 59,
  60, 61, 62, 63, 64, 65, 30, 31, 31, 32, 33, 35, 38, 40, 42, 42, 46, 46,
  45, 45, 45, 45, 46, 47, 47, 49, 49, 52, 52, 54, 55, 57, 58, 58, 60, 61,
  61, 62, 31, 32, 32, 33, 34, 37, 39, 41, 43, 43, 46, 46, 46, 45, 45, 46,
  46, 47, 47, 49, 49, 51, 52, 54, 55, 57, 57, 58, 59, 60, 61, 62, 33, 34,
  35, 35, 37, 39, 41, 43, 44, 45, 47, 47, 46, 46, 45, 46, 46, 47, 47, 49,
  49, 51, 51, 53, 54, 56, 57, 57, 58, 59, 60, 61, 35, 37, 37, 38, 39, 41,
  44, 46, 46, 46, 47, 47, 46, 46, 45, 46, 46, 47, 47, 48, 48, 50, 51, 52,
  53, 55, 56, 56, 57, 58, 59, 61, 36, 38, 39, 40, 41, 43, 46, 47, 47, 47,
  48, 47, 46, 46, 45, 46, 46, 46, 47, 48, 48, 50, 50, 52, 53, 54, 55, 55,
  56, 57, 58, 58, 40, 41, 41, 42, 43, 44, 46, 47, 48, 48, 50, 49, 49, 49,
  48, 49, 49, 49, 49, 51, 51, 52, 53, 54, 55, 57, 57, 58, 59, 59, 59, 59,
  41, 42, 42, 42, 43, 45, 46, 47, 48, 48, 50, 50, 49, 49, 49, 49, 50, 50,
  50, 52, 52, 53, 53, 55, 56, 57, 58, 58, 59, 60, 61, 62, 49, 47, 47, 46,
  46, 47, 47, 48, 50, 50, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56,
  57, 58, 59, 60, 61, 61, 62, 62, 62, 62, 49, 47, 47, 46, 46, 47, 47, 47,
  49, 50, 53, 53, 53, 53, 54, 54, 54, 54, 54, 55, 56, 57, 57, 59, 59, 61,
  61, 62, 63, 63, 64, 65, 48, 47, 47, 45, 46, 46, 46, 46, 49, 49, 53, 53,
  54, 54, 55, 56, 56, 57, 57, 58, 58, 60, 60, 61, 62, 63, 64, 64, 65, 66,
  65, 65, 48, 47, 46, 45, 45, 46, 46, 46, 49, 49, 53, 53, 54, 55, 56, 57,

```

```

57, 58, 58, 59, 60, 61, 61, 63, 63, 65, 65, 65, 66, 66, 67, 68, 49, 47,
46, 45, 45, 45, 45, 45, 48, 49, 53, 54, 55, 56, 58, 59, 60, 61, 61, 62,
62, 63, 64, 65, 66, 67, 68, 68, 69, 70, 69, 68, 50, 47, 47, 45, 46, 46,
46, 46, 49, 49, 54, 54, 56, 57, 59, 60, 60, 62, 62, 63, 64, 65, 65, 67,
68, 69, 69, 70, 70, 70, 71, 71, 50, 48, 48, 46, 46, 46, 46, 46, 49, 50,
54, 54, 56, 57, 60, 60, 61, 63, 63, 65, 65, 67, 67, 68, 69, 71, 71, 71,
72, 73, 72, 71, 52, 49, 49, 47, 47, 47, 47, 46, 49, 50, 54, 54, 57, 58,
61, 62, 63, 65, 65, 67, 67, 69, 70, 71, 72, 73, 74, 74, 75, 74, 74, 75,
52, 50, 49, 47, 47, 47, 47, 47, 49, 50, 54, 54, 57, 58, 61, 62, 63, 65,
66, 68, 68, 70, 71, 72, 73, 75, 75, 75, 76, 77, 77, 75, 54, 52, 51, 49,
49, 49, 48, 48, 51, 52, 55, 55, 58, 59, 62, 63, 65, 67, 68, 70, 70, 73,
73, 75, 76, 78, 78, 78, 79, 78, 78, 79, 54, 52, 51, 49, 49, 49, 48, 48,
51, 52, 55, 56, 58, 60, 62, 64, 65, 67, 68, 70, 71, 73, 74, 75, 76, 78,
78, 79, 80, 81, 81, 79, 57, 54, 54, 52, 51, 51, 50, 50, 52, 53, 56, 57,
60, 61, 63, 65, 67, 69, 70, 73, 73, 76, 77, 79, 80, 82, 82, 83, 84, 83,
82, 83, 57, 55, 54, 52, 52, 51, 51, 50, 53, 53, 57, 57, 60, 61, 64, 65,
67, 70, 71, 73, 74, 77, 77, 79, 80, 82, 83, 83, 84, 85, 85, 83, 60, 57,
56, 54, 54, 53, 52, 52, 54, 55, 58, 59, 61, 63, 65, 67, 68, 71, 72, 75,
75, 79, 79, 82, 83, 85, 86, 86, 87, 87, 86, 87, 61, 58, 57, 55, 55, 54,
53, 53, 55, 56, 59, 59, 62, 63, 66, 68, 69, 72, 73, 76, 76, 80, 80, 83,
84, 86, 87, 88, 89, 89, 89, 87, 63, 60, 59, 57, 57, 56, 55, 54, 57, 57,
60, 61, 63, 65, 67, 69, 71, 73, 75, 78, 78, 82, 82, 85, 86, 89, 89, 90,
91, 92, 90, 91, 64, 61, 60, 58, 57, 57, 56, 55, 57, 58, 61, 61, 64, 65,
68, 69, 71, 74, 75, 78, 78, 82, 83, 86, 87, 89, 90, 91, 92, 93, 94, 91,
65, 61, 61, 58, 58, 57, 56, 55, 58, 58, 61, 62, 64, 65, 68, 70, 71, 74,
75, 78, 79, 83, 83, 86, 88, 90, 91, 91, 93, 94, 94, 96, 66, 63, 62, 60,
59, 58, 57, 56, 59, 59, 62, 63, 65, 66, 69, 70, 72, 75, 76, 79, 80, 84,
84, 87, 89, 91, 92, 93, 94, 94, 96, 96, 67, 64, 63, 61, 60, 59, 58, 57,
59, 60, 62, 63, 66, 66, 70, 70, 73, 74, 77, 78, 81, 83, 85, 87, 89, 92,
93, 94, 94, 96, 96, 97, 68, 64, 64, 61, 61, 60, 59, 58, 59, 61, 62, 64,
65, 67, 69, 71, 72, 74, 77, 78, 81, 82, 85, 86, 89, 90, 94, 94, 96, 96,
98, 97, 69, 65, 65, 62, 62, 61, 61, 58, 59, 62, 62, 65, 65, 68, 68, 71,
71, 75, 75, 79, 79, 83, 83, 87, 87, 91, 91, 96, 96, 97, 97, 99,
/* Size 4x8 */
31, 47, 50, 61, 36, 47, 47, 57, 43, 50, 50, 58, 45, 53, 58, 65, 47, 54,
66, 74, 52, 56, 70, 82, 57, 60, 75, 90, 61, 63, 77, 93,
/* Size 8x4 */
31, 36, 43, 45, 47, 52, 57, 61, 47, 47, 50, 53, 54, 56, 60, 63, 50, 47,
50, 58, 66, 70, 75, 77, 61, 57, 58, 65, 74, 82, 90, 93,
/* Size 8x16 */
32, 32, 40, 49, 51, 57, 63, 67, 31, 33, 41, 47, 49, 54, 59, 63, 31, 35,
43, 46, 47, 51, 57, 60, 35, 39, 46, 46, 47, 50, 55, 58, 41, 43, 48, 49,
49, 52, 57, 59, 49, 47, 50, 53, 54, 57, 60, 62, 48, 46, 49, 54, 57, 60,
64, 65, 49, 45, 48, 56, 61, 64, 67, 69, 50, 46, 49, 57, 63, 67, 71, 73,
52, 48, 50, 58, 65, 71, 75, 77, 54, 50, 51, 59, 67, 73, 78, 81, 57, 52,
53, 61, 69, 77, 82, 85, 61, 55, 56, 63, 72, 80, 86, 88, 64, 58, 58, 65,
73, 82, 89, 92, 66, 59, 59, 66, 75, 84, 91, 94, 68, 61, 59, 65, 72, 81,
89, 95,
/* Size 16x8 */
32, 31, 31, 35, 41, 49, 48, 49, 50, 52, 54, 57, 61, 64, 66, 68, 32, 33,

```



```

35, 39, 43, 47, 46, 45, 46, 48, 50, 52, 55, 58, 59, 61, 40, 41, 43, 46,
48, 50, 49, 48, 49, 50, 51, 53, 56, 58, 59, 59, 49, 47, 46, 46, 49, 53,
54, 56, 57, 58, 59, 61, 63, 65, 66, 65, 51, 49, 47, 47, 49, 54, 57, 61,
63, 65, 67, 69, 72, 73, 75, 72, 57, 54, 51, 50, 52, 57, 60, 64, 67, 71,
73, 77, 80, 82, 84, 81, 63, 59, 57, 55, 57, 60, 64, 67, 71, 75, 78, 82,
86, 89, 91, 89, 67, 63, 60, 58, 59, 62, 65, 69, 73, 77, 81, 85, 88, 92,
94, 95,

```

```

/* Size 16x32 */

```

```

32, 31, 32, 37, 40, 48, 49, 49, 51, 52, 57, 58, 63, 64, 67, 67, 31, 31,
33, 38, 41, 47, 47, 47, 49, 50, 54, 55, 60, 61, 63, 64, 31, 31, 33, 38,
41, 47, 47, 47, 49, 49, 54, 54, 59, 60, 63, 64, 30, 32, 33, 40, 42, 46,
45, 45, 47, 48, 52, 52, 57, 58, 60, 61, 31, 33, 35, 41, 43, 46, 46, 45,
47, 48, 51, 52, 57, 57, 60, 61, 33, 36, 37, 43, 44, 47, 46, 46, 47, 47,
51, 52, 56, 57, 59, 60, 35, 38, 39, 45, 46, 47, 46, 45, 47, 47, 50, 51,
55, 56, 58, 60, 37, 40, 41, 47, 47, 47, 46, 45, 46, 47, 50, 50, 54, 55,
57, 58, 41, 42, 43, 47, 48, 49, 49, 48, 49, 50, 52, 53, 57, 57, 59, 58,
42, 43, 43, 47, 48, 50, 49, 49, 50, 50, 53, 54, 57, 58, 60, 61, 49, 46,
47, 48, 50, 53, 53, 53, 54, 54, 57, 57, 60, 61, 62, 61, 49, 46, 47, 48,
50, 53, 53, 54, 54, 55, 57, 57, 61, 61, 63, 64, 48, 46, 46, 47, 49, 53,
54, 56, 57, 57, 60, 60, 64, 64, 65, 64, 48, 45, 46, 46, 49, 53, 55, 56,
58, 58, 61, 61, 65, 65, 66, 67, 49, 45, 45, 46, 48, 53, 56, 58, 61, 61,
64, 64, 67, 68, 69, 67, 49, 46, 46, 46, 49, 53, 57, 59, 62, 62, 65, 66,
69, 69, 70, 70, 50, 46, 46, 46, 49, 54, 57, 59, 63, 64, 67, 67, 71, 71,
73, 71, 51, 47, 47, 47, 49, 54, 58, 61, 64, 66, 69, 70, 73, 74, 74, 74,
52, 48, 48, 47, 50, 54, 58, 61, 65, 66, 71, 71, 75, 75, 77, 74, 54, 50,
49, 48, 51, 55, 59, 62, 67, 68, 73, 73, 77, 78, 78, 78, 54, 50, 50, 49,
51, 55, 59, 62, 67, 68, 73, 74, 78, 78, 81, 78, 57, 52, 52, 50, 52, 56,
60, 64, 69, 70, 76, 77, 82, 82, 83, 82, 57, 52, 52, 51, 53, 57, 61, 64,
69, 71, 77, 77, 82, 83, 85, 82, 60, 54, 54, 52, 55, 58, 62, 65, 71, 72,
79, 79, 85, 86, 87, 86, 61, 56, 55, 53, 56, 59, 63, 66, 72, 73, 80, 81,
86, 87, 88, 86, 63, 57, 57, 55, 57, 60, 64, 67, 73, 75, 82, 82, 89, 90,
92, 90, 64, 58, 58, 55, 58, 61, 65, 68, 73, 75, 82, 83, 89, 90, 92, 90,
64, 59, 58, 56, 58, 61, 65, 68, 74, 75, 83, 83, 90, 91, 94, 95, 66, 60,
59, 57, 59, 62, 66, 69, 75, 76, 84, 85, 91, 92, 94, 95, 67, 61, 60, 58,
59, 63, 66, 70, 74, 77, 82, 85, 91, 93, 96, 96, 68, 62, 61, 58, 59, 64,
65, 71, 72, 78, 81, 86, 89, 94, 95, 96, 68, 62, 62, 59, 59, 65, 65, 71,
71, 79, 79, 87, 87, 95, 95, 98,

```

```

/* Size 32x16 */

```

```

32, 31, 31, 30, 31, 33, 35, 37, 41, 42, 49, 49, 48, 48, 49, 49, 50, 51,
52, 54, 54, 57, 57, 60, 61, 63, 64, 64, 66, 67, 68, 68, 31, 31, 31, 32,
33, 36, 38, 40, 42, 43, 46, 46, 46, 45, 45, 46, 46, 47, 48, 50, 50, 52,
52, 54, 56, 57, 58, 59, 60, 61, 62, 62, 32, 33, 33, 33, 35, 37, 39, 41,
43, 43, 47, 47, 46, 46, 45, 46, 46, 47, 48, 49, 50, 52, 52, 54, 55, 57,
58, 58, 59, 60, 61, 62, 37, 38, 38, 40, 41, 43, 45, 47, 47, 47, 48, 48,
47, 46, 46, 46, 46, 47, 47, 48, 49, 50, 51, 52, 53, 55, 55, 56, 57, 58,
58, 59, 40, 41, 41, 42, 43, 44, 46, 47, 48, 48, 50, 50, 49, 49, 48, 49,
49, 49, 50, 51, 51, 52, 53, 55, 56, 57, 58, 58, 59, 59, 59, 59, 48, 47,
47, 46, 46, 47, 47, 47, 49, 50, 53, 53, 53, 53, 53, 54, 54, 54, 55,
55, 56, 57, 58, 59, 60, 61, 61, 62, 63, 64, 65, 49, 47, 47, 45, 46, 46,
46, 46, 49, 49, 53, 53, 54, 55, 56, 57, 57, 58, 58, 59, 59, 60, 61, 62,

```

```

63, 64, 65, 65, 66, 66, 65, 65, 49, 47, 47, 45, 45, 46, 45, 45, 48, 49,
53, 54, 56, 56, 58, 59, 59, 61, 61, 62, 62, 64, 64, 65, 66, 67, 68, 68,
69, 70, 71, 71, 51, 49, 49, 47, 47, 47, 47, 46, 49, 50, 54, 54, 57, 58,
61, 62, 63, 64, 65, 67, 67, 69, 69, 71, 72, 73, 73, 74, 75, 74, 72, 71,
52, 50, 49, 48, 48, 47, 47, 47, 50, 50, 54, 55, 57, 58, 61, 62, 64, 66,
66, 68, 68, 70, 71, 72, 73, 75, 75, 75, 76, 77, 78, 79, 57, 54, 54, 52,
51, 51, 50, 50, 52, 53, 57, 57, 60, 61, 64, 65, 67, 69, 71, 73, 73, 76,
77, 79, 80, 82, 82, 83, 84, 82, 81, 79, 58, 55, 54, 52, 52, 52, 51, 50,
53, 54, 57, 57, 60, 61, 64, 66, 67, 70, 71, 73, 74, 77, 77, 79, 81, 82,
83, 83, 85, 85, 86, 87, 63, 60, 59, 57, 57, 56, 55, 54, 57, 57, 60, 61,
64, 65, 67, 69, 71, 73, 75, 77, 78, 82, 82, 85, 86, 89, 89, 90, 91, 91,
89, 87, 64, 61, 60, 58, 57, 57, 56, 55, 57, 58, 61, 61, 64, 65, 68, 69,
71, 74, 75, 78, 78, 82, 83, 86, 87, 90, 90, 91, 92, 93, 94, 95, 67, 63,
63, 60, 60, 59, 58, 57, 59, 60, 62, 63, 65, 66, 69, 70, 73, 74, 77, 78,
81, 83, 85, 87, 88, 92, 92, 94, 94, 96, 95, 95, 67, 64, 64, 61, 61, 60,
60, 58, 58, 61, 61, 64, 64, 67, 67, 70, 71, 74, 74, 78, 78, 82, 82, 86,
86, 90, 90, 95, 95, 96, 96, 98,
/* Size 4x16 */
31, 48, 52, 64, 31, 47, 49, 60, 33, 46, 48, 57, 38, 47, 47, 56, 42, 49,
50, 57, 46, 53, 54, 61, 46, 53, 57, 64, 45, 53, 61, 68, 46, 54, 64, 71,
48, 54, 66, 75, 50, 55, 68, 78, 52, 57, 71, 83, 56, 59, 73, 87, 58, 61,
75, 90, 60, 62, 76, 92, 62, 64, 78, 94,
/* Size 16x4 */
31, 31, 33, 38, 42, 46, 46, 45, 46, 48, 50, 52, 56, 58, 60, 62, 48, 47,
46, 47, 49, 53, 53, 53, 54, 54, 55, 57, 59, 61, 62, 64, 52, 49, 48, 47,
50, 54, 57, 61, 64, 66, 68, 71, 73, 75, 76, 78, 64, 60, 57, 56, 57, 61,
64, 68, 71, 75, 78, 83, 87, 90, 92, 94,
/* Size 8x32 */
32, 32, 40, 49, 51, 57, 63, 67, 31, 33, 41, 47, 49, 54, 60, 63, 31, 33,
41, 47, 49, 54, 59, 63, 30, 33, 42, 45, 47, 52, 57, 60, 31, 35, 43, 46,
47, 51, 57, 60, 33, 37, 44, 46, 47, 51, 56, 59, 35, 39, 46, 46, 47, 50,
55, 58, 37, 41, 47, 46, 46, 50, 54, 57, 41, 43, 48, 49, 49, 52, 57, 59,
42, 43, 48, 49, 50, 53, 57, 60, 49, 47, 50, 53, 54, 57, 60, 62, 49, 47,
50, 53, 54, 57, 61, 63, 48, 46, 49, 54, 57, 60, 64, 65, 48, 46, 49, 55,
58, 61, 65, 66, 49, 45, 48, 56, 61, 64, 67, 69, 49, 46, 49, 57, 62, 65,
69, 70, 50, 46, 49, 57, 63, 67, 71, 73, 51, 47, 49, 58, 64, 69, 73, 74,
52, 48, 50, 58, 65, 71, 75, 77, 54, 49, 51, 59, 67, 73, 77, 78, 54, 50,
51, 59, 67, 73, 78, 81, 57, 52, 52, 60, 69, 76, 82, 83, 57, 52, 53, 61,
69, 77, 82, 85, 60, 54, 55, 62, 71, 79, 85, 87, 61, 55, 56, 63, 72, 80,
86, 88, 63, 57, 57, 64, 73, 82, 89, 92, 64, 58, 58, 65, 73, 82, 89, 92,
64, 58, 58, 65, 74, 83, 90, 94, 66, 59, 59, 66, 75, 84, 91, 94, 67, 60,
59, 66, 74, 82, 91, 96, 68, 61, 59, 65, 72, 81, 89, 95, 68, 62, 59, 65,
71, 79, 87, 95,
/* Size 32x8 */
32, 31, 31, 30, 31, 33, 35, 37, 41, 42, 49, 49, 48, 48, 49, 49, 50, 51,
52, 54, 54, 57, 57, 60, 61, 63, 64, 64, 66, 67, 68, 68, 32, 33, 33, 33,
35, 37, 39, 41, 43, 43, 47, 47, 46, 46, 45, 46, 46, 47, 48, 49, 50, 52,
52, 54, 55, 57, 58, 58, 59, 60, 61, 62, 40, 41, 41, 42, 43, 44, 46, 47,
48, 48, 50, 50, 49, 49, 48, 49, 49, 49, 50, 51, 51, 52, 53, 55, 56, 57,
58, 58, 59, 59, 59, 59, 49, 47, 47, 45, 46, 46, 46, 46, 49, 49, 53, 53,

```

```

54, 55, 56, 57, 57, 58, 58, 59, 59, 60, 61, 62, 63, 64, 65, 65, 66, 66,
65, 65, 51, 49, 49, 47, 47, 47, 47, 46, 49, 50, 54, 54, 57, 58, 61, 62,
63, 64, 65, 67, 67, 69, 69, 71, 72, 73, 73, 74, 75, 74, 72, 71, 57, 54,
54, 52, 51, 51, 50, 50, 52, 53, 57, 57, 60, 61, 64, 65, 67, 69, 71, 73,
73, 76, 77, 79, 80, 82, 82, 83, 84, 82, 81, 79, 63, 60, 59, 57, 57, 56,
55, 54, 57, 57, 60, 61, 64, 65, 67, 69, 71, 73, 75, 77, 78, 82, 82, 85,
86, 89, 89, 90, 91, 91, 89, 87, 67, 63, 63, 60, 60, 59, 58, 57, 59, 60,
62, 63, 65, 66, 69, 70, 73, 74, 77, 78, 81, 83, 85, 87, 88, 92, 92, 94,
94, 96, 95, 95 },
},
{
  /* Luma */
  /* Size 4x4 */
  32, 34, 49, 72, 34, 48, 60, 79, 49, 60, 82, 104, 72, 79, 104, 134,
  /* Size 8x8 */
  32, 32, 34, 38, 46, 56, 68, 78, 32, 33, 35, 39, 45, 54, 64, 74, 34, 35,
  39, 45, 51, 58, 68, 76, 38, 39, 45, 54, 61, 69, 78, 86, 46, 45, 51, 61,
  71, 80, 90, 99, 56, 54, 58, 69, 80, 92, 103, 113, 68, 64, 68, 78, 90,
  103, 117, 128, 78, 74, 76, 86, 99, 113, 128, 140,
  /* Size 16x16 */
  32, 31, 31, 31, 32, 34, 36, 39, 44, 48, 54, 59, 65, 71, 80, 83, 31, 32,
  32, 32, 32, 34, 35, 38, 42, 46, 51, 56, 62, 68, 76, 78, 31, 32, 32, 32,
  32, 33, 34, 37, 41, 44, 49, 54, 59, 65, 72, 75, 31, 32, 32, 33, 34, 35,
  36, 39, 42, 45, 50, 54, 59, 64, 71, 74, 32, 32, 32, 34, 35, 37, 38, 40,
  42, 46, 49, 53, 58, 63, 69, 72, 34, 34, 33, 35, 37, 39, 42, 45, 47, 51,
  54, 58, 63, 68, 74, 76, 36, 35, 34, 36, 38, 42, 48, 50, 54, 57, 60, 64,
  68, 73, 79, 81, 39, 38, 37, 39, 40, 45, 50, 54, 58, 61, 65, 69, 73, 78,
  84, 86, 44, 42, 41, 42, 42, 47, 54, 58, 63, 67, 71, 75, 79, 84, 90, 92,
  48, 46, 44, 45, 46, 51, 57, 61, 67, 71, 76, 80, 85, 90, 96, 99, 54, 51,
  49, 50, 49, 54, 60, 65, 71, 76, 82, 87, 92, 97, 104, 106, 59, 56, 54,
  54, 53, 58, 64, 69, 75, 80, 87, 92, 98, 103, 110, 113, 65, 62, 59, 59,
  58, 63, 68, 73, 79, 85, 92, 98, 105, 111, 118, 121, 71, 68, 65, 64, 63,
  68, 73, 78, 84, 90, 97, 103, 111, 117, 125, 128, 80, 76, 72, 71, 69, 74,
  79, 84, 90, 96, 104, 110, 118, 125, 134, 137, 83, 78, 75, 74, 72, 76,
  81, 86, 92, 99, 106, 113, 121, 128, 137, 140,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 32, 32, 34, 34, 36, 36, 39, 39, 44, 44, 48,
  48, 54, 54, 59, 59, 65, 65, 71, 71, 80, 80, 83, 83, 87, 31, 32, 32, 32,
  32, 32, 32, 32, 32, 34, 34, 35, 35, 38, 38, 42, 42, 46, 46, 51, 51, 56,
  56, 62, 62, 68, 68, 76, 76, 78, 78, 83, 31, 32, 32, 32, 32, 32, 32, 32,
  32, 34, 34, 35, 35, 38, 38, 42, 42, 46, 46, 51, 51, 56, 56, 62, 62, 68,
  68, 76, 76, 78, 78, 83, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34,
  34, 37, 37, 41, 41, 44, 44, 49, 49, 54, 54, 59, 59, 65, 65, 72, 72, 75,
  75, 79, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 37, 37, 41,
  41, 44, 44, 49, 49, 54, 54, 59, 59, 65, 65, 72, 72, 75, 75, 79, 31, 32,
  32, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 39, 39, 42, 42, 45, 45, 50,
  50, 54, 54, 59, 59, 64, 64, 71, 71, 74, 74, 77, 31, 32, 32, 32, 32, 33,
  33, 34, 34, 35, 35, 36, 36, 39, 39, 42, 42, 45, 45, 50, 50, 54, 54, 59,
  59, 64, 64, 71, 71, 74, 74, 77, 32, 32, 32, 32, 32, 34, 34, 35, 35, 37,
  37, 38, 38, 40, 40, 42, 42, 46, 46, 49, 49, 53, 53, 58, 58, 63, 63, 69,

```

```

69, 72, 72, 75, 32, 32, 32, 32, 32, 34, 34, 35, 35, 37, 37, 38, 38, 40,
40, 42, 42, 46, 46, 49, 49, 53, 53, 58, 58, 63, 63, 69, 69, 72, 72, 75,
34, 34, 34, 33, 33, 35, 35, 37, 37, 39, 39, 42, 42, 45, 45, 47, 47, 51,
51, 54, 54, 58, 58, 63, 63, 68, 68, 74, 74, 76, 76, 80, 34, 34, 34, 33,
33, 35, 35, 37, 37, 39, 39, 42, 42, 45, 45, 47, 47, 51, 51, 54, 54, 58,
58, 63, 63, 68, 68, 74, 74, 76, 76, 80, 36, 35, 35, 34, 34, 36, 36, 38,
38, 42, 42, 48, 48, 50, 50, 54, 54, 57, 57, 60, 60, 64, 64, 68, 68, 73,
73, 79, 79, 81, 81, 84, 36, 35, 35, 34, 34, 36, 36, 38, 38, 42, 42, 48,
48, 50, 50, 54, 54, 57, 57, 60, 60, 64, 64, 68, 68, 73, 73, 79, 79, 81,
81, 84, 39, 38, 38, 37, 37, 39, 39, 40, 40, 45, 45, 50, 50, 54, 54, 58,
58, 61, 61, 65, 65, 69, 69, 73, 73, 78, 78, 84, 84, 86, 86, 90, 39, 38,
38, 37, 37, 39, 39, 40, 40, 45, 45, 50, 50, 54, 54, 58, 58, 61, 61, 65,
65, 69, 69, 73, 73, 78, 78, 84, 84, 86, 86, 90, 44, 42, 42, 41, 41, 42,
42, 42, 42, 47, 47, 54, 54, 58, 58, 63, 63, 67, 67, 71, 71, 75, 75, 79,
79, 84, 84, 90, 90, 92, 92, 96, 44, 42, 42, 41, 41, 42, 42, 42, 42, 47,
47, 54, 54, 58, 58, 63, 63, 67, 67, 71, 71, 75, 75, 79, 79, 84, 84, 90,
90, 92, 92, 96, 48, 46, 46, 44, 44, 45, 45, 46, 46, 51, 51, 57, 57, 61,
61, 57, 61, 67, 71, 71, 76, 76, 80, 80, 85, 85, 90, 90, 96, 96, 99, 99, 102,
48, 46, 46, 44, 44, 45, 45, 46, 46, 51, 51, 57, 57, 61, 61, 67, 67, 71,
71, 76, 76, 80, 80, 85, 85, 90, 90, 96, 96, 99, 99, 102, 54, 51, 51, 49,
49, 50, 50, 49, 49, 54, 54, 60, 60, 65, 65, 71, 71, 76, 76, 82, 82, 87,
87, 92, 92, 97, 97, 104, 104, 106, 106, 109, 54, 51, 51, 49, 49, 50, 50,
49, 49, 54, 54, 60, 60, 65, 65, 71, 71, 76, 76, 82, 82, 87, 87, 92, 92,
97, 97, 104, 104, 106, 106, 109, 59, 56, 56, 54, 54, 54, 54, 53, 53, 58,
58, 64, 64, 69, 69, 75, 75, 80, 80, 87, 87, 92, 92, 98, 98, 103, 103,
110, 110, 113, 113, 116, 59, 56, 56, 54, 54, 54, 54, 53, 53, 58, 58, 64,
64, 69, 69, 75, 75, 80, 80, 87, 87, 92, 92, 98, 98, 103, 103, 110, 110,
113, 113, 116, 65, 62, 62, 59, 59, 59, 59, 58, 58, 63, 63, 68, 68, 73,
73, 79, 79, 85, 85, 92, 92, 98, 98, 105, 105, 111, 111, 118, 118, 121,
121, 124, 65, 62, 62, 59, 59, 59, 59, 58, 58, 63, 63, 68, 68, 73, 73,
79, 79, 85, 85, 92, 92, 98, 98, 105, 105, 111, 111, 118, 118, 121, 121,
124, 71, 68, 68, 65, 65, 64, 64, 63, 63, 68, 68, 73, 73, 78, 78, 84, 84,
90, 90, 97, 97, 103, 103, 111, 111, 117, 117, 125, 125, 128, 128, 132,
71, 68, 68, 65, 65, 64, 64, 63, 63, 68, 68, 73, 73, 78, 78, 84, 84, 90,
90, 97, 97, 103, 103, 111, 111, 117, 117, 125, 125, 128, 128, 132, 80,
76, 76, 72, 72, 71, 71, 69, 69, 74, 74, 79, 79, 84, 84, 90, 90, 96, 96,
104, 104, 110, 110, 118, 118, 125, 125, 134, 134, 137, 137, 141, 80, 76,
76, 72, 72, 71, 71, 69, 69, 74, 74, 79, 79, 84, 84, 90, 90, 96, 96, 104,
104, 110, 110, 118, 118, 125, 125, 134, 134, 137, 137, 141, 83, 78, 78,
75, 75, 74, 74, 72, 72, 76, 76, 81, 81, 86, 86, 92, 92, 99, 99, 106,
106, 113, 113, 121, 121, 128, 128, 137, 137, 140, 140, 144, 83, 78, 78,
75, 75, 74, 74, 72, 72, 76, 76, 81, 81, 86, 86, 92, 92, 99, 99, 106,
106, 113, 113, 121, 121, 128, 128, 137, 137, 140, 140, 144, 87, 83, 83,
79, 79, 77, 77, 75, 75, 80, 80, 84, 84, 90, 90, 96, 96, 102, 102, 109,
109, 116, 116, 124, 124, 132, 132, 141, 141, 144, 144, 149,
/* Size 4x8 */
32, 35, 51, 75, 32, 36, 50, 71, 34, 42, 54, 73, 37, 50, 65, 84, 45, 56,
76, 96, 54, 63, 87, 110, 65, 73, 97, 125, 75, 81, 106, 136,
/* Size 8x4 */
32, 32, 34, 37, 45, 54, 65, 75, 35, 36, 42, 50, 56, 63, 73, 81, 51, 50,

```

```

54, 65, 76, 87, 97, 106, 75, 71, 73, 84, 96, 110, 125, 136,
/* Size 8x16 */
32, 31, 32, 36, 44, 53, 65, 79, 31, 32, 32, 35, 42, 51, 62, 75, 31, 32,
33, 34, 41, 49, 59, 72, 32, 32, 34, 36, 42, 50, 59, 71, 32, 33, 35, 38,
42, 49, 58, 69, 34, 34, 37, 42, 48, 54, 63, 73, 36, 34, 38, 48, 54, 60,
68, 78, 39, 37, 40, 50, 58, 65, 73, 84, 44, 41, 43, 53, 63, 71, 79, 90,
48, 45, 46, 56, 67, 76, 85, 96, 53, 49, 50, 60, 71, 82, 92, 103, 58, 54,
54, 63, 75, 87, 98, 110, 65, 60, 58, 68, 79, 92, 105, 118, 71, 65, 63,
73, 84, 97, 111, 125, 79, 72, 70, 79, 90, 104, 118, 133, 82, 75, 72, 81,
92, 106, 121, 136,
/* Size 16x8 */
32, 31, 31, 32, 32, 34, 36, 39, 44, 48, 53, 58, 65, 71, 79, 82, 31, 32,
32, 32, 33, 34, 34, 37, 41, 45, 49, 54, 60, 65, 72, 75, 32, 32, 33, 34,
35, 37, 38, 40, 43, 46, 50, 54, 58, 63, 70, 72, 36, 35, 34, 36, 38, 42,
48, 50, 53, 56, 60, 63, 68, 73, 79, 81, 44, 42, 41, 42, 42, 48, 54, 58,
63, 67, 71, 75, 79, 84, 90, 92, 53, 51, 49, 50, 49, 54, 60, 65, 71, 76,
82, 87, 92, 97, 104, 106, 65, 62, 59, 59, 58, 63, 68, 73, 79, 85, 92,
98, 105, 111, 118, 121, 79, 75, 72, 71, 69, 73, 78, 84, 90, 96, 103,
110, 118, 125, 133, 136,
/* Size 16x32 */
32, 31, 31, 32, 32, 36, 36, 44, 44, 53, 53, 65, 65, 79, 79, 87, 31, 32,
32, 32, 32, 35, 35, 42, 42, 51, 51, 62, 62, 75, 75, 82, 31, 32, 32, 32,
32, 35, 35, 42, 42, 51, 51, 62, 62, 75, 75, 82, 31, 32, 32, 33, 33, 34,
34, 41, 41, 49, 49, 59, 59, 72, 72, 78, 31, 32, 32, 33, 33, 34, 34, 41,
41, 49, 49, 59, 59, 72, 72, 78, 32, 32, 32, 34, 34, 36, 36, 42, 42, 50,
50, 59, 59, 71, 71, 77, 32, 32, 32, 34, 34, 36, 36, 42, 42, 50, 50, 59,
59, 71, 71, 77, 32, 33, 33, 35, 35, 38, 38, 42, 42, 49, 49, 58, 58, 69,
69, 75, 32, 33, 33, 35, 35, 38, 38, 42, 42, 49, 49, 58, 58, 69, 69, 75,
34, 34, 34, 37, 37, 42, 42, 48, 48, 54, 54, 63, 63, 73, 73, 79, 34, 34,
34, 37, 37, 42, 42, 48, 48, 54, 54, 63, 63, 73, 73, 79, 36, 34, 34, 38,
38, 48, 48, 54, 54, 60, 60, 68, 68, 78, 78, 84, 36, 34, 34, 38, 38, 48,
48, 54, 54, 60, 60, 68, 68, 78, 78, 84, 39, 37, 37, 40, 40, 50, 50, 58,
58, 65, 65, 73, 73, 84, 84, 89, 39, 37, 37, 40, 40, 50, 50, 58, 58, 65,
65, 73, 73, 84, 84, 89, 44, 41, 41, 43, 43, 53, 53, 63, 63, 71, 71, 79,
79, 90, 90, 95, 44, 41, 41, 43, 43, 53, 53, 63, 63, 71, 71, 79, 79, 90,
90, 95, 48, 45, 45, 46, 46, 56, 56, 67, 67, 76, 76, 85, 85, 96, 96, 102,
48, 45, 45, 46, 46, 56, 56, 67, 67, 76, 76, 85, 85, 96, 96, 102, 53, 49,
49, 50, 50, 60, 60, 71, 71, 82, 82, 92, 92, 103, 103, 109, 53, 49, 49,
50, 50, 60, 60, 71, 71, 82, 82, 92, 92, 103, 103, 109, 58, 54, 54, 54,
54, 63, 63, 75, 75, 87, 87, 98, 98, 110, 110, 116, 58, 54, 54, 54, 54,
63, 63, 75, 75, 87, 87, 98, 98, 110, 110, 116, 65, 60, 60, 58, 58, 68,
68, 79, 79, 92, 92, 105, 105, 118, 118, 124, 65, 60, 60, 58, 58, 68, 68,
79, 79, 92, 92, 105, 105, 118, 118, 124, 71, 65, 65, 63, 63, 73, 73, 84,
84, 97, 97, 111, 111, 125, 125, 132, 71, 65, 65, 63, 63, 73, 73, 84, 84,
97, 97, 111, 111, 125, 125, 132, 79, 72, 72, 70, 70, 79, 79, 90, 90,
104, 104, 118, 118, 133, 133, 141, 79, 72, 72, 70, 70, 79, 79, 90, 90,
104, 104, 118, 118, 133, 133, 141, 82, 75, 75, 72, 72, 81, 81, 92, 92,
106, 106, 121, 121, 136, 136, 144, 82, 75, 75, 72, 72, 81, 81, 92, 92,
106, 106, 121, 121, 136, 136, 144, 87, 79, 79, 76, 76, 84, 84, 96, 96,
109, 109, 124, 124, 141, 141, 149,

```

```
/* Size 32x16 */
```

```
32, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 36, 36, 39, 39, 44, 44, 48,
48, 53, 53, 58, 58, 65, 65, 71, 71, 79, 79, 82, 82, 87, 31, 32, 32, 32,
32, 32, 32, 33, 33, 34, 34, 34, 34, 37, 37, 41, 41, 45, 45, 49, 49, 54,
54, 60, 60, 65, 65, 72, 72, 75, 75, 79, 31, 32, 32, 32, 32, 32, 32, 33,
33, 34, 34, 34, 34, 37, 37, 41, 41, 45, 45, 49, 49, 54, 54, 60, 60, 65,
65, 72, 72, 75, 75, 79, 32, 32, 32, 33, 33, 34, 34, 35, 35, 37, 37, 38,
38, 40, 40, 43, 43, 46, 46, 50, 50, 54, 54, 58, 58, 63, 63, 70, 70, 72,
72, 76, 32, 32, 32, 33, 33, 34, 34, 35, 35, 37, 37, 38, 38, 40, 40, 43,
43, 46, 46, 50, 50, 54, 54, 58, 58, 63, 63, 70, 70, 72, 72, 76, 36, 35,
35, 34, 34, 36, 36, 38, 38, 42, 42, 48, 48, 50, 50, 53, 53, 56, 56, 60,
60, 63, 63, 68, 68, 73, 73, 79, 79, 81, 81, 84, 36, 35, 35, 34, 34, 36,
36, 38, 38, 42, 42, 48, 48, 50, 50, 53, 53, 56, 56, 60, 60, 63, 63, 68,
68, 73, 73, 79, 79, 81, 81, 84, 44, 42, 42, 41, 41, 42, 42, 42, 42, 48,
48, 54, 54, 58, 58, 63, 63, 67, 67, 71, 71, 75, 75, 79, 79, 84, 84, 90,
90, 92, 92, 96, 44, 42, 42, 41, 41, 42, 42, 42, 42, 48, 48, 54, 54, 58,
58, 63, 63, 67, 67, 71, 71, 75, 75, 79, 79, 84, 84, 90, 90, 92, 92, 96,
53, 51, 51, 49, 49, 50, 50, 49, 49, 54, 54, 60, 60, 65, 65, 71, 71, 76,
76, 82, 82, 87, 87, 92, 92, 97, 97, 104, 104, 106, 106, 109, 53, 51, 51,
49, 49, 50, 50, 49, 49, 54, 54, 60, 60, 65, 65, 71, 71, 76, 76, 82, 82,
87, 87, 92, 92, 97, 97, 104, 104, 106, 106, 109, 65, 62, 62, 59, 59, 59,
59, 58, 58, 63, 63, 68, 68, 73, 73, 79, 79, 85, 85, 92, 92, 98, 98, 105,
105, 111, 111, 118, 118, 121, 121, 124, 65, 62, 62, 59, 59, 59, 59, 58,
58, 63, 63, 68, 68, 73, 73, 79, 79, 85, 85, 92, 92, 98, 98, 105, 105,
111, 111, 118, 118, 121, 121, 124, 79, 75, 75, 72, 72, 71, 71, 69, 69,
73, 73, 78, 78, 84, 84, 90, 90, 96, 96, 103, 103, 110, 110, 118, 118,
125, 125, 133, 133, 136, 136, 141, 79, 75, 75, 72, 72, 71, 71, 69, 69,
73, 73, 78, 78, 84, 84, 90, 90, 96, 96, 103, 103, 110, 110, 118, 118,
125, 125, 133, 133, 136, 136, 141, 87, 82, 82, 78, 78, 77, 77, 75, 75,
79, 79, 84, 84, 89, 89, 95, 95, 102, 102, 109, 109, 116, 116, 124, 124,
132, 132, 141, 141, 144, 144, 149,
```

```
/* Size 4x16 */
```

```
31, 36, 53, 79, 32, 35, 51, 75, 32, 34, 49, 72, 32, 36, 50, 71, 33, 38,
49, 69, 34, 42, 54, 73, 34, 48, 60, 78, 37, 50, 65, 84, 41, 53, 71, 90,
45, 56, 76, 96, 49, 60, 82, 103, 54, 63, 87, 110, 60, 68, 92, 118, 65,
73, 97, 125, 72, 79, 104, 133, 75, 81, 106, 136,
```

```
/* Size 16x4 */
```

```
31, 32, 32, 32, 33, 34, 34, 37, 41, 45, 49, 54, 60, 65, 72, 75, 36, 35,
34, 36, 38, 42, 48, 50, 53, 56, 60, 63, 68, 73, 79, 81, 53, 51, 49, 50,
49, 54, 60, 65, 71, 76, 82, 87, 92, 97, 104, 106, 79, 75, 72, 71, 69,
73, 78, 84, 90, 96, 103, 110, 118, 125, 133, 136,
```

```
/* Size 8x32 */
```

```
32, 31, 32, 36, 44, 53, 65, 79, 31, 32, 32, 35, 42, 51, 62, 75, 31, 32,
32, 35, 42, 51, 62, 75, 31, 32, 33, 34, 41, 49, 59, 72, 31, 32, 33, 34,
41, 49, 59, 72, 32, 32, 34, 36, 42, 50, 59, 71, 32, 32, 34, 36, 42, 50,
59, 71, 32, 33, 35, 38, 42, 49, 58, 69, 32, 33, 35, 38, 42, 49, 58, 69,
34, 34, 37, 42, 48, 54, 63, 73, 34, 34, 37, 42, 48, 54, 63, 73, 36, 34,
38, 48, 54, 60, 68, 78, 36, 34, 38, 48, 54, 60, 68, 78, 39, 37, 40, 50,
58, 65, 73, 84, 39, 37, 40, 50, 58, 65, 73, 84, 44, 41, 43, 53, 63, 71,
79, 90, 44, 41, 43, 53, 63, 71, 79, 90, 48, 45, 46, 56, 67, 76, 85, 96,
```

```

48, 45, 46, 56, 67, 76, 85, 96, 53, 49, 50, 60, 71, 82, 92, 103, 53, 49,
50, 60, 71, 82, 92, 103, 58, 54, 54, 63, 75, 87, 98, 110, 58, 54, 54,
63, 75, 87, 98, 110, 65, 60, 58, 68, 79, 92, 105, 118, 65, 60, 58, 68,
79, 92, 105, 118, 71, 65, 63, 73, 84, 97, 111, 125, 71, 65, 63, 73, 84,
97, 111, 125, 79, 72, 70, 79, 90, 104, 118, 133, 79, 72, 70, 79, 90,
104, 118, 133, 82, 75, 72, 81, 92, 106, 121, 136, 82, 75, 72, 81, 92,
106, 121, 136, 87, 79, 76, 84, 96, 109, 124, 141,
/* Size 32x8 */
32, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 36, 36, 39, 39, 44, 44, 48,
48, 53, 53, 58, 58, 65, 65, 71, 71, 79, 79, 82, 82, 87, 31, 32, 32, 32,
32, 32, 32, 33, 33, 34, 34, 34, 34, 37, 37, 41, 41, 45, 45, 49, 49, 54,
54, 60, 60, 65, 65, 72, 72, 75, 75, 79, 32, 32, 32, 33, 33, 34, 34, 35,
35, 37, 37, 38, 38, 40, 40, 43, 43, 46, 46, 50, 50, 54, 54, 58, 58, 63,
63, 70, 70, 72, 72, 76, 36, 35, 35, 34, 34, 36, 36, 38, 38, 42, 42, 48,
48, 50, 50, 53, 53, 56, 56, 60, 60, 63, 63, 68, 68, 73, 73, 79, 79, 81,
81, 84, 44, 42, 42, 41, 41, 42, 42, 42, 42, 48, 48, 54, 54, 58, 58, 63,
63, 67, 67, 71, 71, 75, 75, 79, 79, 84, 84, 90, 90, 92, 92, 96, 53, 51,
51, 49, 49, 50, 50, 49, 49, 54, 54, 60, 60, 65, 65, 71, 71, 76, 76, 82,
82, 87, 87, 92, 92, 97, 97, 104, 104, 106, 106, 109, 65, 62, 62, 59, 59,
59, 59, 58, 58, 63, 63, 68, 68, 73, 73, 79, 79, 85, 85, 92, 92, 98, 98,
105, 105, 111, 111, 118, 118, 121, 121, 124, 79, 75, 75, 72, 72, 71, 71,
69, 69, 73, 73, 78, 78, 84, 84, 90, 90, 96, 96, 103, 103, 110, 110, 118,
118, 125, 125, 133, 133, 136, 136, 141 },
{ /* Chroma */
/* Size 4x4 */
32, 46, 47, 57, 46, 53, 54, 60, 47, 54, 66, 75, 57, 60, 75, 89,
/* Size 8x8 */
31, 34, 42, 47, 48, 52, 57, 61, 34, 39, 45, 46, 46, 49, 53, 57, 42, 45,
48, 49, 50, 52, 55, 58, 47, 46, 49, 54, 56, 58, 61, 64, 48, 46, 50, 56,
61, 65, 68, 71, 52, 49, 52, 58, 65, 71, 75, 79, 57, 53, 55, 61, 68, 75,
82, 86, 61, 57, 58, 64, 71, 79, 86, 91,
/* Size 16x16 */
32, 31, 30, 33, 36, 41, 49, 48, 49, 50, 52, 54, 57, 60, 63, 65, 31, 31,
31, 34, 38, 42, 47, 47, 47, 48, 50, 52, 54, 57, 60, 61, 30, 31, 32, 35,
40, 42, 46, 45, 45, 46, 47, 49, 52, 54, 57, 58, 33, 34, 35, 39, 43, 45,
47, 46, 45, 46, 47, 49, 51, 53, 56, 57, 36, 38, 40, 43, 47, 47, 48, 46,
45, 46, 47, 48, 50, 52, 54, 55, 41, 42, 42, 45, 47, 48, 50, 49, 49, 50,
50, 52, 53, 55, 57, 58, 49, 47, 46, 47, 48, 50, 53, 53, 53, 54, 54, 55,
56, 58, 60, 61, 48, 47, 45, 46, 46, 49, 53, 54, 55, 56, 57, 58, 60, 61,
63, 64, 49, 47, 45, 45, 45, 49, 53, 55, 58, 60, 61, 62, 63, 65, 67, 68,
50, 48, 46, 46, 46, 50, 54, 56, 60, 61, 63, 65, 67, 68, 71, 71, 52, 50,
47, 47, 47, 50, 54, 57, 61, 63, 66, 68, 70, 72, 75, 75, 54, 52, 49, 49,
48, 52, 55, 58, 62, 65, 68, 71, 73, 75, 78, 79, 57, 54, 52, 51, 50, 53,
56, 60, 63, 67, 70, 73, 76, 79, 82, 83, 60, 57, 54, 53, 52, 55, 58, 61,
65, 68, 72, 75, 79, 82, 85, 86, 63, 60, 57, 56, 54, 57, 60, 63, 67, 71,
75, 78, 82, 85, 89, 90, 65, 61, 58, 57, 55, 58, 61, 64, 68, 71, 75, 79,
83, 86, 90, 91,
/* Size 32x32 */
32, 31, 31, 30, 30, 33, 33, 36, 36, 41, 41, 49, 49, 48, 48, 49, 49, 50,
50, 52, 52, 54, 54, 57, 57, 60, 60, 63, 63, 65, 65, 67, 31, 31, 31, 31,

```

31, 34, 34, 38, 38, 42, 42, 47, 47, 47, 47, 47, 47, 48, 48, 50, 50, 52,  
 52, 54, 54, 57, 57, 60, 60, 61, 61, 63, 31, 31, 31, 31, 31, 34, 34, 38,  
 38, 42, 42, 47, 47, 47, 47, 47, 47, 48, 48, 50, 50, 52, 52, 54, 54, 57,  
 57, 60, 60, 61, 61, 63, 30, 31, 31, 32, 32, 35, 35, 40, 40, 42, 42, 46,  
 46, 45, 45, 45, 45, 46, 46, 47, 47, 49, 49, 52, 52, 54, 54, 57, 57, 58,  
 58, 60, 30, 31, 31, 32, 32, 35, 35, 40, 40, 42, 42, 46, 46, 45, 45, 45,  
 45, 46, 46, 47, 47, 49, 49, 52, 52, 54, 54, 57, 57, 58, 58, 60, 33, 34,  
 34, 35, 35, 39, 39, 43, 43, 45, 45, 47, 47, 46, 46, 45, 45, 46, 46, 47,  
 47, 49, 49, 51, 51, 53, 53, 56, 56, 57, 57, 59, 33, 34, 34, 35, 35, 39,  
 39, 43, 43, 45, 45, 47, 47, 46, 46, 45, 45, 46, 46, 47, 47, 49, 49, 51,  
 51, 53, 53, 56, 56, 57, 57, 59, 36, 38, 38, 40, 40, 43, 43, 47, 47, 47,  
 47, 48, 48, 46, 46, 45, 45, 46, 46, 47, 47, 48, 48, 50, 50, 52, 52, 54,  
 54, 55, 55, 57, 36, 38, 38, 40, 40, 43, 43, 47, 47, 47, 47, 48, 48, 46,  
 46, 45, 45, 46, 46, 47, 47, 48, 48, 50, 50, 52, 52, 54, 54, 55, 55, 57,  
 41, 42, 42, 42, 42, 45, 45, 47, 47, 48, 48, 50, 50, 49, 49, 49, 49, 50,  
 50, 50, 50, 52, 52, 53, 53, 55, 55, 57, 57, 58, 58, 60, 41, 42, 42, 42,  
 42, 45, 45, 47, 47, 48, 48, 50, 50, 49, 49, 49, 49, 50, 50, 50, 50, 52,  
 52, 53, 53, 55, 55, 57, 57, 58, 58, 60, 49, 47, 47, 46, 46, 47, 47, 48,  
 48, 50, 50, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 58,  
 58, 60, 60, 61, 61, 62, 49, 47, 47, 46, 46, 47, 47, 48, 48, 50, 50, 53,  
 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 58, 58, 60, 60, 61,  
 61, 62, 48, 47, 47, 45, 45, 46, 46, 46, 46, 49, 49, 53, 53, 54, 54, 55,  
 55, 56, 56, 57, 57, 58, 58, 60, 60, 61, 61, 63, 63, 64, 64, 66, 48, 47,  
 47, 45, 45, 46, 46, 46, 46, 49, 49, 53, 53, 54, 54, 55, 55, 56, 56, 57,  
 57, 58, 58, 60, 60, 61, 61, 63, 63, 64, 64, 66, 49, 47, 47, 45, 45, 45,  
 45, 45, 45, 49, 49, 53, 53, 55, 55, 58, 58, 60, 60, 61, 61, 62, 62, 63,  
 63, 65, 65, 67, 67, 68, 68, 69, 49, 47, 47, 45, 45, 45, 45, 45, 45, 49,  
 49, 53, 53, 55, 55, 58, 58, 60, 60, 61, 61, 62, 62, 63, 63, 65, 65, 67,  
 67, 68, 68, 69, 50, 48, 48, 46, 46, 46, 46, 46, 46, 50, 50, 54, 54, 56,  
 56, 60, 60, 61, 61, 63, 63, 65, 65, 67, 67, 68, 68, 71, 71, 71, 71, 72,  
 50, 48, 48, 46, 46, 46, 46, 46, 46, 50, 50, 54, 54, 56, 56, 60, 60, 61,  
 61, 63, 63, 65, 65, 67, 67, 68, 68, 71, 71, 71, 71, 72, 52, 50, 50, 47,  
 47, 47, 47, 47, 47, 50, 50, 54, 54, 57, 57, 61, 61, 63, 63, 66, 66, 68,  
 68, 70, 70, 72, 72, 75, 75, 75, 75, 76, 52, 50, 50, 47, 47, 47, 47, 47,  
 47, 50, 50, 54, 54, 57, 57, 61, 61, 63, 63, 66, 66, 68, 68, 70, 70, 72,  
 72, 75, 75, 75, 75, 76, 54, 52, 52, 49, 49, 49, 49, 48, 48, 52, 52, 55,  
 55, 58, 58, 62, 62, 65, 65, 68, 68, 71, 71, 73, 73, 75, 75, 78, 78, 79,  
 79, 80, 54, 52, 52, 49, 49, 49, 49, 48, 48, 52, 52, 55, 55, 58, 58, 62,  
 62, 65, 65, 68, 68, 71, 71, 73, 73, 75, 75, 78, 78, 79, 79, 80, 57, 54,  
 54, 52, 52, 51, 51, 50, 50, 53, 53, 56, 56, 60, 60, 63, 63, 67, 67, 70,  
 70, 73, 73, 76, 76, 79, 79, 82, 82, 83, 83, 84, 57, 54, 54, 52, 52, 51,  
 51, 50, 50, 53, 53, 56, 56, 60, 60, 63, 63, 67, 67, 70, 70, 73, 73, 76,  
 76, 79, 79, 82, 82, 83, 83, 84, 60, 57, 57, 54, 54, 53, 53, 52, 52, 55,  
 55, 58, 58, 61, 61, 65, 65, 68, 68, 72, 72, 75, 75, 79, 79, 82, 82, 85,  
 85, 86, 86, 88, 60, 57, 57, 54, 54, 53, 53, 52, 52, 55, 55, 58, 58, 61,  
 61, 65, 65, 68, 68, 72, 72, 75, 75, 79, 79, 82, 82, 85, 85, 86, 86, 88,  
 63, 60, 60, 57, 57, 56, 56, 54, 54, 57, 57, 60, 60, 63, 63, 67, 67, 71,  
 71, 75, 75, 78, 78, 82, 82, 85, 85, 89, 89, 90, 90, 92, 63, 60, 60, 57,  
 57, 56, 56, 54, 54, 57, 57, 60, 60, 63, 63, 67, 67, 71, 71, 75, 75, 78,  
 78, 82, 82, 85, 85, 89, 89, 90, 90, 92, 65, 61, 61, 58, 58, 57, 57, 55,



```

55, 58, 58, 61, 61, 64, 64, 68, 68, 71, 71, 75, 75, 79, 79, 83, 83, 86,
86, 90, 90, 91, 91, 93, 65, 61, 61, 58, 58, 57, 57, 55, 55, 58, 58, 61,
61, 64, 64, 68, 68, 71, 71, 75, 75, 79, 79, 83, 83, 86, 86, 90, 90, 91,
91, 93, 67, 63, 63, 60, 60, 59, 59, 57, 57, 60, 60, 62, 62, 66, 66, 69,
69, 72, 72, 76, 76, 80, 80, 84, 84, 88, 88, 92, 92, 93, 93, 95,
/* Size 4x8 */
31, 47, 50, 60, 36, 47, 47, 56, 43, 50, 50, 57, 46, 53, 57, 64, 46, 54,
64, 71, 50, 55, 68, 78, 54, 58, 72, 85, 59, 61, 75, 90,
/* Size 8x4 */
31, 36, 43, 46, 46, 50, 54, 59, 47, 47, 50, 53, 54, 55, 58, 61, 50, 47,
50, 57, 64, 68, 72, 75, 60, 56, 57, 64, 71, 78, 85, 90,
/* Size 8x16 */
32, 31, 37, 48, 49, 52, 57, 63, 31, 31, 38, 47, 47, 50, 54, 60, 30, 32,
40, 46, 45, 48, 52, 57, 33, 36, 43, 47, 46, 47, 51, 56, 37, 40, 47, 47,
45, 47, 50, 54, 42, 43, 47, 50, 49, 50, 53, 57, 49, 46, 48, 53, 53, 54,
57, 60, 48, 46, 47, 53, 56, 57, 60, 64, 49, 45, 46, 53, 58, 61, 64, 67,
50, 46, 46, 54, 59, 64, 67, 71, 52, 48, 47, 54, 61, 66, 71, 75, 54, 50,
49, 55, 62, 68, 73, 78, 57, 52, 50, 56, 64, 70, 76, 82, 60, 54, 52, 58,
65, 72, 79, 85, 63, 57, 55, 60, 67, 75, 82, 89, 64, 59, 56, 61, 68, 75,
83, 90,
/* Size 16x8 */
32, 31, 30, 33, 37, 42, 49, 48, 49, 50, 52, 54, 57, 60, 63, 64, 31, 31,
32, 36, 40, 43, 46, 46, 45, 46, 48, 50, 52, 54, 57, 59, 37, 38, 40, 43,
47, 47, 48, 47, 46, 46, 47, 49, 50, 52, 55, 56, 48, 47, 46, 47, 47, 50,
53, 53, 53, 54, 54, 55, 56, 58, 60, 61, 49, 47, 45, 46, 45, 49, 53, 56,
58, 59, 61, 62, 64, 65, 67, 68, 52, 50, 48, 47, 47, 50, 54, 57, 61, 64,
66, 68, 70, 72, 75, 75, 57, 54, 52, 51, 50, 53, 57, 60, 64, 67, 71, 73,
76, 79, 82, 83, 63, 60, 57, 56, 54, 57, 60, 64, 67, 71, 75, 78, 82, 85,
89, 90,
/* Size 16x32 */
32, 31, 31, 37, 37, 48, 48, 49, 49, 52, 52, 57, 57, 63, 63, 66, 31, 31,
31, 38, 38, 47, 47, 47, 47, 50, 50, 54, 54, 60, 60, 63, 31, 31, 31, 38,
38, 47, 47, 47, 47, 50, 50, 54, 54, 60, 60, 63, 30, 32, 32, 40, 40, 46,
46, 45, 45, 48, 48, 52, 52, 57, 57, 60, 30, 32, 32, 40, 40, 46, 46, 45,
45, 48, 48, 52, 52, 57, 57, 60, 33, 36, 36, 43, 43, 47, 47, 46, 46, 47,
47, 51, 51, 56, 56, 59, 33, 36, 36, 43, 43, 47, 47, 46, 46, 47, 47, 51,
51, 56, 56, 59, 37, 40, 40, 47, 47, 47, 47, 45, 45, 47, 47, 50, 50, 54,
54, 57, 37, 40, 40, 47, 47, 47, 47, 45, 45, 47, 47, 50, 50, 54, 54, 57,
42, 43, 43, 47, 47, 50, 50, 49, 49, 50, 50, 53, 53, 57, 57, 60, 42, 43,
43, 47, 47, 50, 50, 49, 49, 50, 50, 53, 53, 57, 57, 60, 49, 46, 46, 48,
48, 53, 53, 53, 53, 54, 54, 57, 57, 60, 60, 62, 49, 46, 46, 48, 48, 53,
53, 53, 53, 54, 54, 57, 57, 60, 60, 62, 48, 46, 46, 47, 47, 53, 53, 56,
56, 57, 57, 60, 60, 64, 64, 66, 48, 46, 46, 47, 47, 53, 53, 56, 56, 57,
57, 60, 60, 64, 64, 66, 49, 45, 45, 46, 46, 53, 53, 58, 58, 61, 61, 64,
64, 67, 67, 69, 49, 45, 45, 46, 46, 53, 53, 58, 58, 61, 61, 64, 64, 67,
67, 69, 50, 46, 46, 46, 46, 54, 54, 59, 59, 64, 64, 67, 67, 71, 71, 73,
50, 46, 46, 46, 46, 54, 54, 59, 59, 64, 64, 67, 67, 71, 71, 73, 52, 48,
48, 47, 47, 54, 54, 61, 61, 66, 66, 71, 71, 75, 75, 77, 52, 48, 48, 47,
47, 54, 54, 61, 61, 66, 66, 71, 71, 75, 75, 77, 54, 50, 50, 49, 49, 55,
55, 62, 62, 68, 68, 73, 73, 78, 78, 80, 54, 50, 50, 49, 49, 55, 55, 62,

```

```

62, 68, 68, 73, 73, 78, 78, 80, 57, 52, 52, 50, 50, 56, 56, 64, 64, 70,
70, 76, 76, 82, 82, 84, 57, 52, 52, 50, 50, 56, 56, 64, 64, 70, 70, 76,
76, 82, 82, 84, 60, 54, 54, 52, 52, 58, 58, 65, 65, 72, 72, 79, 79, 85,
85, 88, 60, 54, 54, 52, 52, 58, 58, 65, 65, 72, 72, 79, 79, 85, 85, 88,
63, 57, 57, 55, 55, 60, 60, 67, 67, 75, 75, 82, 82, 89, 89, 92, 63, 57,
57, 55, 55, 60, 60, 67, 67, 75, 75, 82, 82, 89, 89, 92, 64, 59, 59, 56,
56, 61, 61, 68, 68, 75, 75, 83, 83, 90, 90, 93, 64, 59, 59, 56, 56, 61,
61, 68, 68, 75, 75, 83, 83, 90, 90, 93, 66, 60, 60, 57, 57, 63, 63, 69,
69, 77, 77, 84, 84, 92, 92, 95,
/* Size 32x16 */
32, 31, 31, 30, 30, 33, 33, 37, 37, 42, 42, 49, 49, 48, 48, 49, 49, 50,
50, 52, 52, 54, 54, 57, 57, 60, 60, 63, 63, 64, 64, 66, 31, 31, 31, 32,
32, 36, 36, 40, 40, 43, 43, 46, 46, 46, 46, 45, 45, 46, 46, 48, 48, 50,
50, 52, 52, 54, 54, 57, 57, 59, 59, 60, 31, 31, 31, 32, 32, 36, 36, 40,
40, 43, 43, 46, 46, 46, 46, 45, 45, 46, 46, 48, 48, 50, 50, 52, 52, 54,
54, 57, 57, 59, 59, 60, 37, 38, 38, 40, 40, 43, 43, 47, 47, 47, 47, 48,
48, 47, 47, 46, 46, 46, 46, 47, 47, 49, 49, 50, 50, 52, 52, 55, 55, 56,
56, 57, 37, 38, 38, 40, 40, 43, 43, 47, 47, 47, 47, 48, 48, 47, 47, 46,
46, 46, 46, 47, 47, 49, 49, 50, 50, 52, 52, 55, 55, 56, 56, 57, 48, 47,
47, 46, 46, 47, 47, 47, 47, 50, 50, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56,
54, 55, 55, 56, 56, 58, 58, 60, 60, 61, 61, 63, 48, 47, 47, 46, 46, 47,
47, 47, 47, 50, 50, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56,
56, 58, 58, 60, 60, 61, 61, 63, 49, 47, 47, 45, 45, 46, 46, 45, 45, 49,
49, 53, 53, 56, 56, 58, 58, 59, 59, 61, 61, 62, 62, 64, 64, 65, 65, 67,
67, 68, 68, 69, 49, 47, 47, 45, 45, 46, 46, 45, 45, 49, 49, 53, 53, 56,
56, 58, 58, 59, 59, 61, 61, 62, 62, 64, 64, 65, 65, 67, 67, 68, 68, 69,
52, 50, 50, 48, 48, 47, 47, 47, 47, 50, 50, 54, 54, 57, 57, 61, 61, 64,
64, 66, 66, 68, 68, 70, 70, 72, 72, 75, 75, 75, 75, 77, 52, 50, 50, 48,
48, 47, 47, 47, 47, 50, 50, 54, 54, 57, 57, 61, 61, 64, 64, 66, 66, 68,
68, 70, 70, 72, 72, 75, 75, 75, 75, 77, 57, 54, 54, 52, 52, 51, 51, 50,
50, 53, 53, 57, 57, 60, 60, 64, 64, 67, 67, 71, 71, 73, 73, 76, 76, 79,
79, 82, 82, 83, 83, 84, 57, 54, 54, 52, 52, 51, 51, 50, 50, 53, 53, 57,
57, 60, 60, 64, 64, 67, 67, 71, 71, 73, 73, 76, 76, 79, 79, 82, 82, 83,
83, 84, 63, 60, 60, 57, 57, 56, 56, 54, 54, 57, 57, 60, 60, 64, 64, 67,
67, 71, 71, 75, 75, 78, 78, 82, 82, 85, 85, 89, 89, 90, 90, 92, 63, 60,
60, 57, 57, 56, 56, 54, 54, 57, 57, 60, 60, 64, 64, 67, 67, 71, 71, 75,
75, 78, 78, 82, 82, 85, 85, 89, 89, 90, 90, 92, 66, 63, 63, 60, 60, 59,
59, 57, 57, 60, 60, 62, 62, 66, 66, 69, 69, 73, 73, 77, 77, 80, 80, 84,
84, 88, 88, 92, 92, 93, 93, 95,
/* Size 4x16 */
31, 48, 52, 63, 31, 47, 50, 60, 32, 46, 48, 57, 36, 47, 47, 56, 40, 47,
47, 54, 43, 50, 50, 57, 46, 53, 54, 60, 46, 53, 57, 64, 45, 53, 61, 67,
46, 54, 64, 71, 48, 54, 66, 75, 50, 55, 68, 78, 52, 56, 70, 82, 54, 58,
72, 85, 57, 60, 75, 89, 59, 61, 75, 90,
/* Size 16x4 */
31, 31, 32, 36, 40, 43, 46, 46, 45, 46, 48, 50, 52, 54, 57, 59, 48, 47,
46, 47, 47, 50, 53, 53, 53, 54, 54, 55, 56, 58, 60, 61, 52, 50, 48, 47,
47, 50, 54, 57, 61, 64, 66, 68, 70, 72, 75, 75, 63, 60, 57, 56, 54, 57,
60, 64, 67, 71, 75, 78, 82, 85, 89, 90,
/* Size 8x32 */

```

```

32, 31, 37, 48, 49, 52, 57, 63, 31, 31, 38, 47, 47, 50, 54, 60, 31, 31,
38, 47, 47, 50, 54, 60, 30, 32, 40, 46, 45, 48, 52, 57, 30, 32, 40, 46,
45, 48, 52, 57, 33, 36, 43, 47, 46, 47, 51, 56, 33, 36, 43, 47, 46, 47,
51, 56, 37, 40, 47, 47, 45, 47, 50, 54, 37, 40, 47, 47, 45, 47, 50, 54,
42, 43, 47, 50, 49, 50, 53, 57, 42, 43, 47, 50, 49, 50, 53, 57, 49, 46,
48, 53, 53, 54, 57, 60, 49, 46, 48, 53, 53, 54, 57, 60, 48, 46, 47, 53,
56, 57, 60, 64, 48, 46, 47, 53, 56, 57, 60, 64, 49, 45, 46, 53, 58, 61,
64, 67, 49, 45, 46, 53, 58, 61, 64, 67, 50, 46, 46, 54, 59, 64, 67, 71,
50, 46, 46, 54, 59, 64, 67, 71, 52, 48, 47, 54, 61, 66, 71, 75, 52, 48,
47, 54, 61, 66, 71, 75, 54, 50, 49, 55, 62, 68, 73, 78, 54, 50, 49, 55,
62, 68, 73, 78, 57, 52, 50, 56, 64, 70, 76, 82, 57, 52, 50, 56, 64, 70,
76, 82, 60, 54, 52, 58, 65, 72, 79, 85, 60, 54, 52, 58, 65, 72, 79, 85,
63, 57, 55, 60, 67, 75, 82, 89, 63, 57, 55, 60, 67, 75, 82, 89, 64, 59,
56, 61, 68, 75, 83, 90, 64, 59, 56, 61, 68, 75, 83, 90, 66, 60, 57, 63,
69, 77, 84, 92,

```

```

/* Size 32x8 */

```

```

32, 31, 31, 30, 30, 33, 33, 37, 37, 42, 42, 49, 49, 48, 48, 49, 49, 50,
50, 52, 52, 54, 54, 57, 57, 60, 60, 63, 63, 64, 64, 66, 31, 31, 31, 32,
32, 36, 36, 40, 40, 43, 43, 46, 46, 46, 46, 45, 45, 46, 46, 48, 48, 50,
50, 52, 52, 54, 54, 57, 57, 59, 59, 60, 37, 38, 38, 40, 40, 43, 43, 47,
47, 47, 47, 48, 48, 47, 47, 46, 46, 46, 46, 47, 47, 49, 49, 50, 50, 52,
52, 55, 55, 56, 56, 57, 48, 47, 47, 46, 46, 47, 47, 47, 47, 50, 50, 53,
53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 58, 58, 60, 60, 61,
61, 63, 49, 47, 47, 45, 45, 46, 46, 45, 45, 49, 49, 53, 53, 56, 56, 58,
58, 59, 59, 61, 61, 62, 62, 64, 64, 65, 65, 67, 67, 68, 68, 69, 52, 50,
50, 48, 48, 47, 47, 47, 47, 50, 50, 54, 54, 57, 57, 61, 61, 64, 64, 66,
66, 68, 68, 70, 70, 72, 72, 75, 75, 75, 75, 77, 57, 54, 54, 52, 52, 51,
51, 50, 50, 53, 53, 57, 57, 60, 60, 64, 64, 67, 67, 71, 71, 73, 73, 76,
76, 79, 79, 82, 82, 83, 83, 84, 63, 60, 60, 57, 57, 56, 56, 54, 54, 57,
57, 60, 60, 64, 64, 67, 67, 71, 71, 75, 75, 78, 78, 82, 82, 85, 85, 89,
89, 90, 90, 92 },

```

```

},

```

```

{

```

```

{ /* Luma */

```

```

/* Size 4x4 */

```

```

32, 33, 45, 62, 33, 39, 51, 64, 45, 51, 71, 87, 62, 64, 87, 108,

```

```

/* Size 8x8 */

```

```

31, 32, 32, 35, 42, 51, 59, 69, 32, 32, 33, 35, 41, 49, 56, 65, 32, 33,
35, 38, 43, 49, 56, 64, 35, 35, 38, 48, 54, 59, 66, 73, 42, 41, 43, 54,
63, 71, 77, 85, 51, 49, 49, 59, 71, 81, 89, 97, 59, 56, 56, 66, 77, 89,
98, 108, 69, 65, 64, 73, 85, 97, 108, 119,

```

```

/* Size 16x16 */

```

```

32, 31, 31, 31, 32, 34, 35, 38, 41, 45, 48, 54, 59, 65, 71, 80, 31, 32,
32, 32, 32, 34, 35, 37, 40, 43, 46, 51, 56, 62, 68, 76, 31, 32, 32, 32,
32, 33, 34, 36, 38, 41, 44, 49, 54, 59, 65, 72, 31, 32, 32, 33, 34, 35,
36, 38, 40, 42, 45, 50, 54, 59, 64, 71, 32, 32, 32, 34, 35, 37, 38, 39,
41, 43, 46, 49, 53, 58, 63, 69, 34, 34, 33, 35, 37, 39, 42, 44, 46, 48,
51, 54, 58, 63, 68, 74, 35, 35, 34, 36, 38, 42, 46, 48, 50, 53, 55, 59,
62, 67, 72, 78, 38, 37, 36, 38, 39, 44, 48, 51, 54, 57, 59, 63, 67, 71,
76, 82, 41, 40, 38, 40, 41, 46, 50, 54, 57, 60, 63, 67, 71, 75, 80, 86,

```

```

45, 43, 41, 42, 43, 48, 53, 57, 60, 65, 68, 72, 76, 81, 85, 91, 48, 46,
44, 45, 46, 51, 55, 59, 63, 68, 71, 76, 80, 85, 90, 96, 54, 51, 49, 50,
49, 54, 59, 63, 67, 72, 76, 82, 87, 92, 97, 104, 59, 56, 54, 54, 53, 58,
62, 67, 71, 76, 80, 87, 92, 98, 103, 110, 65, 62, 59, 59, 58, 63, 67,
71, 75, 81, 85, 92, 98, 105, 111, 118, 71, 68, 65, 64, 63, 68, 72, 76,
80, 85, 90, 97, 103, 111, 117, 125, 80, 76, 72, 71, 69, 74, 78, 82, 86,
91, 96, 104, 110, 118, 125, 134,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 34, 34, 35, 36, 38, 39, 41, 44,
45, 48, 48, 53, 54, 57, 59, 62, 65, 67, 71, 72, 80, 80, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 34, 34, 35, 35, 37, 38, 40, 42, 43, 46, 46, 51,
52, 55, 56, 59, 62, 64, 68, 69, 76, 76, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 34, 34, 35, 35, 37, 38, 40, 42, 43, 46, 46, 51, 51, 55, 56, 59,
62, 64, 68, 69, 76, 76, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33,
34, 34, 36, 38, 39, 41, 42, 45, 45, 49, 50, 53, 54, 57, 60, 62, 66, 66,
73, 73, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 36, 37,
38, 41, 41, 44, 44, 49, 49, 52, 54, 56, 59, 61, 65, 65, 72, 72, 31, 32,
32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 35, 35, 37, 38, 39, 41, 42, 45,
45, 49, 49, 52, 54, 56, 59, 61, 64, 65, 72, 72, 31, 32, 32, 32, 32, 33,
33, 33, 34, 34, 35, 35, 36, 36, 38, 39, 40, 42, 42, 45, 45, 49, 50, 52,
54, 56, 59, 60, 64, 65, 71, 71, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34,
35, 35, 36, 37, 38, 39, 40, 42, 43, 45, 45, 49, 49, 52, 54, 56, 59, 60,
64, 64, 70, 70, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 37, 37, 38, 38,
39, 40, 41, 42, 43, 46, 46, 49, 49, 52, 53, 55, 58, 59, 63, 63, 69, 69,
32, 32, 32, 32, 33, 33, 34, 34, 35, 35, 37, 37, 38, 38, 40, 41, 41, 43,
43, 46, 46, 49, 50, 52, 54, 56, 58, 60, 63, 64, 70, 70, 34, 34, 34, 33,
33, 34, 35, 35, 37, 37, 39, 39, 42, 42, 44, 45, 46, 47, 48, 51, 51, 54,
54, 57, 58, 60, 63, 64, 68, 68, 74, 74, 34, 34, 34, 33, 33, 34, 35, 35,
37, 37, 39, 39, 42, 42, 44, 45, 46, 47, 48, 51, 51, 54, 54, 57, 58, 60,
63, 64, 68, 68, 74, 74, 35, 35, 35, 34, 34, 35, 36, 36, 38, 38, 42, 42,
46, 47, 48, 49, 50, 52, 53, 55, 55, 58, 59, 61, 62, 64, 67, 68, 72, 72,
78, 78, 36, 35, 35, 34, 34, 35, 36, 37, 38, 38, 42, 42, 47, 48, 50, 50,
52, 54, 54, 57, 57, 59, 60, 62, 64, 66, 68, 69, 73, 73, 79, 79, 38, 37,
37, 36, 36, 37, 38, 38, 39, 40, 44, 44, 48, 50, 51, 52, 54, 56, 57, 59,
59, 62, 63, 65, 67, 69, 71, 72, 76, 76, 82, 82, 39, 38, 38, 38, 37, 38,
39, 39, 40, 41, 45, 45, 49, 50, 52, 54, 55, 58, 58, 61, 61, 64, 65, 67,
69, 71, 73, 74, 78, 78, 84, 84, 41, 40, 40, 39, 38, 39, 40, 40, 41, 41,
46, 46, 50, 52, 54, 55, 57, 60, 60, 63, 63, 67, 67, 70, 71, 73, 75, 77,
80, 81, 86, 86, 44, 42, 42, 41, 41, 41, 42, 42, 42, 43, 47, 47, 52, 54,
56, 58, 60, 63, 64, 67, 67, 71, 71, 74, 75, 77, 79, 81, 84, 85, 90, 90,
45, 43, 43, 42, 41, 42, 42, 43, 43, 43, 48, 48, 53, 54, 57, 58, 60, 64,
65, 68, 68, 72, 72, 75, 76, 78, 81, 82, 85, 86, 91, 91, 48, 46, 46, 45,
44, 45, 45, 45, 46, 46, 51, 51, 55, 57, 59, 61, 63, 67, 68, 71, 71, 75,
76, 79, 80, 83, 85, 87, 90, 91, 96, 96, 48, 46, 46, 45, 44, 45, 45, 45,
46, 46, 51, 51, 55, 57, 59, 61, 63, 67, 68, 71, 71, 75, 76, 79, 80, 83,
85, 87, 90, 91, 96, 96, 53, 51, 51, 49, 49, 49, 49, 49, 49, 54, 54,
58, 59, 62, 64, 67, 71, 72, 75, 75, 81, 81, 85, 86, 89, 91, 93, 97, 97,
103, 103, 54, 52, 51, 50, 49, 49, 50, 49, 49, 50, 54, 54, 59, 60, 63,
65, 67, 71, 72, 76, 76, 81, 82, 85, 87, 89, 92, 94, 97, 98, 104, 104,
57, 55, 55, 53, 52, 52, 52, 52, 52, 52, 57, 57, 61, 62, 65, 67, 70, 74,

```

```

75, 79, 79, 85, 85, 89, 90, 93, 96, 98, 102, 102, 108, 108, 59, 56, 56,
54, 54, 54, 54, 54, 53, 54, 58, 58, 62, 64, 67, 69, 71, 75, 76, 80, 80,
86, 87, 90, 92, 95, 98, 99, 103, 104, 110, 110, 62, 59, 59, 57, 56, 56,
56, 56, 55, 56, 60, 60, 64, 66, 69, 71, 73, 77, 78, 83, 83, 89, 89, 93,
95, 98, 101, 103, 107, 108, 114, 114, 65, 62, 62, 60, 59, 59, 59, 59,
58, 58, 63, 63, 67, 68, 71, 73, 75, 79, 81, 85, 85, 91, 92, 96, 98, 101,
105, 106, 111, 111, 118, 118, 67, 64, 64, 62, 61, 61, 60, 60, 59, 60,
64, 64, 68, 69, 72, 74, 77, 81, 82, 87, 87, 93, 94, 98, 99, 103, 106,
108, 113, 113, 120, 120, 71, 68, 68, 66, 65, 64, 64, 64, 63, 63, 68, 68,
72, 73, 76, 78, 80, 84, 85, 90, 90, 97, 97, 102, 103, 107, 111, 113,
117, 118, 125, 125, 72, 69, 69, 66, 65, 65, 65, 64, 63, 64, 68, 68, 72,
73, 76, 78, 81, 85, 86, 91, 91, 97, 98, 102, 104, 108, 111, 113, 118,
119, 126, 126, 80, 76, 76, 73, 72, 72, 71, 70, 69, 70, 74, 74, 78, 79,
82, 84, 86, 90, 91, 96, 96, 103, 104, 108, 110, 114, 118, 120, 125, 126,
134, 134, 80, 76, 76, 73, 72, 72, 71, 70, 69, 70, 74, 74, 78, 79, 82,
84, 86, 90, 91, 96, 96, 103, 104, 108, 110, 114, 118, 120, 125, 126,
134, 134,
/* Size 4x8 */
32, 34, 43, 62, 32, 34, 42, 59, 33, 37, 44, 58, 35, 43, 54, 68, 41, 48,
64, 79, 49, 54, 71, 91, 57, 60, 78, 101, 66, 68, 86, 111,
/* Size 8x4 */
32, 32, 33, 35, 41, 49, 57, 66, 34, 34, 37, 43, 48, 54, 60, 68, 43, 42,
44, 54, 64, 71, 78, 86, 62, 59, 58, 68, 79, 91, 101, 111,
/* Size 8x16 */
32, 31, 32, 36, 44, 53, 62, 73, 31, 32, 32, 35, 42, 51, 59, 69, 31, 32,
33, 34, 41, 49, 57, 66, 32, 32, 34, 36, 42, 50, 57, 65, 32, 33, 35, 38,
42, 49, 56, 64, 34, 34, 37, 42, 48, 54, 61, 69, 35, 34, 38, 47, 52, 59,
65, 73, 38, 36, 40, 49, 56, 63, 69, 77, 41, 39, 41, 51, 60, 67, 74, 81,
44, 42, 43, 54, 64, 72, 79, 86, 48, 45, 46, 56, 67, 76, 83, 91, 53, 49,
50, 60, 71, 82, 90, 99, 58, 54, 54, 63, 75, 87, 95, 105, 65, 60, 58, 68,
79, 92, 102, 112, 71, 65, 63, 73, 84, 97, 108, 119, 79, 72, 70, 79, 90,
104, 115, 127,
/* Size 16x8 */
32, 31, 31, 32, 32, 34, 35, 38, 41, 44, 48, 53, 58, 65, 71, 79, 31, 32,
32, 32, 33, 34, 34, 36, 39, 42, 45, 49, 54, 60, 65, 72, 32, 32, 33, 34,
35, 37, 38, 40, 41, 43, 46, 50, 54, 58, 63, 70, 36, 35, 34, 36, 38, 42,
47, 49, 51, 54, 56, 60, 63, 68, 73, 79, 44, 42, 41, 42, 42, 48, 52, 56,
60, 64, 67, 71, 75, 79, 84, 90, 53, 51, 49, 50, 49, 54, 59, 63, 67, 72,
76, 82, 87, 92, 97, 104, 62, 59, 57, 57, 56, 61, 65, 69, 74, 79, 83, 90,
95, 102, 108, 115, 73, 69, 66, 65, 64, 69, 73, 77, 81, 86, 91, 99, 105,
112, 119, 127,
/* Size 16x32 */
32, 31, 31, 32, 32, 34, 36, 38, 44, 44, 53, 53, 62, 65, 73, 79, 31, 32,
32, 32, 32, 34, 35, 37, 42, 43, 51, 51, 60, 62, 70, 75, 31, 32, 32, 32,
32, 34, 35, 37, 42, 43, 51, 51, 59, 62, 69, 75, 31, 32, 32, 32, 32, 33,
35, 36, 41, 42, 50, 50, 58, 60, 67, 73, 31, 32, 32, 32, 33, 33, 34, 36,
41, 41, 49, 49, 57, 59, 66, 72, 31, 32, 32, 33, 33, 34, 35, 37, 41, 42,
49, 49, 57, 59, 66, 71, 32, 32, 32, 33, 34, 35, 36, 38, 42, 43, 50, 50,
57, 59, 65, 71, 32, 32, 32, 34, 34, 35, 37, 38, 42, 43, 49, 49, 56, 59,
65, 70, 32, 32, 33, 34, 35, 37, 38, 39, 42, 43, 49, 49, 56, 58, 64, 69,

```

```

32, 33, 33, 34, 35, 37, 39, 40, 43, 44, 50, 50, 56, 58, 64, 69, 34, 34,
34, 36, 37, 39, 42, 44, 48, 48, 54, 54, 61, 63, 69, 73, 34, 34, 34, 36,
37, 39, 42, 44, 48, 48, 54, 54, 61, 63, 69, 73, 35, 34, 34, 37, 38, 42,
47, 48, 52, 53, 59, 59, 65, 67, 73, 77, 36, 35, 34, 37, 38, 43, 48, 49,
54, 54, 60, 60, 66, 68, 74, 78, 38, 36, 36, 38, 40, 44, 49, 51, 56, 57,
63, 63, 69, 71, 77, 81, 39, 38, 37, 40, 40, 45, 50, 52, 58, 58, 65, 65,
71, 73, 79, 84, 41, 39, 39, 41, 41, 46, 51, 54, 60, 60, 67, 67, 74, 76,
81, 86, 44, 41, 41, 42, 43, 48, 53, 56, 63, 64, 71, 71, 78, 79, 85, 90,
44, 42, 42, 43, 43, 48, 54, 56, 64, 64, 72, 72, 79, 81, 86, 91, 48, 45,
45, 46, 46, 51, 56, 59, 67, 67, 76, 76, 83, 85, 91, 96, 48, 45, 45, 46,
46, 51, 56, 59, 67, 67, 76, 76, 83, 85, 91, 96, 53, 49, 49, 49, 49, 54,
59, 62, 71, 71, 81, 81, 89, 91, 98, 103, 53, 50, 49, 50, 50, 54, 60, 63,
71, 72, 82, 82, 90, 92, 99, 103, 57, 53, 52, 52, 52, 57, 62, 65, 74, 75,
85, 85, 94, 96, 103, 108, 58, 54, 54, 54, 54, 58, 63, 67, 75, 76, 87,
87, 95, 98, 105, 110, 61, 57, 57, 56, 56, 60, 66, 69, 77, 78, 89, 89,
98, 101, 108, 114, 65, 60, 60, 59, 58, 63, 68, 71, 79, 80, 92, 92, 102,
105, 112, 118, 67, 62, 61, 60, 60, 64, 69, 72, 81, 82, 94, 94, 103, 106,
114, 120, 71, 66, 65, 64, 63, 68, 73, 76, 84, 85, 97, 97, 108, 111, 119,
125, 72, 66, 66, 64, 64, 68, 73, 76, 85, 86, 98, 98, 108, 111, 119, 125,
79, 73, 72, 71, 70, 74, 79, 82, 90, 91, 104, 104, 115, 118, 127, 133,
79, 73, 72, 71, 70, 74, 79, 82, 90, 91, 104, 104, 115, 118, 127, 133,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 35, 36, 38, 39, 41, 44,
44, 48, 48, 53, 53, 57, 58, 61, 65, 67, 71, 72, 79, 79, 31, 32, 32, 32,
32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 36, 38, 39, 41, 42, 45, 45, 49,
50, 53, 54, 57, 60, 62, 66, 66, 73, 73, 31, 32, 32, 32, 32, 32, 32, 32,
33, 33, 34, 34, 34, 34, 36, 37, 39, 41, 42, 45, 45, 49, 49, 52, 54, 57,
60, 61, 65, 66, 72, 72, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34, 36, 36,
37, 37, 38, 40, 41, 42, 43, 46, 46, 49, 50, 52, 54, 56, 59, 60, 64, 64,
71, 71, 32, 32, 32, 32, 33, 33, 34, 34, 35, 35, 37, 37, 38, 38, 40, 40,
41, 43, 43, 46, 46, 49, 50, 52, 54, 56, 58, 60, 63, 64, 70, 70, 34, 34,
34, 33, 33, 34, 35, 35, 37, 37, 39, 39, 42, 43, 44, 45, 46, 48, 48, 51,
51, 54, 54, 57, 58, 60, 63, 64, 68, 68, 74, 74, 36, 35, 35, 35, 34, 35,
36, 37, 38, 39, 42, 42, 47, 48, 49, 50, 51, 53, 54, 56, 56, 59, 60, 62,
63, 66, 68, 69, 73, 73, 79, 79, 38, 37, 37, 36, 36, 37, 38, 38, 39, 40,
44, 44, 48, 49, 51, 52, 54, 56, 56, 59, 59, 62, 63, 65, 67, 69, 71, 72,
76, 76, 82, 82, 44, 42, 42, 41, 41, 41, 42, 42, 42, 43, 48, 48, 52, 54,
56, 58, 60, 63, 64, 67, 67, 71, 71, 74, 75, 77, 79, 81, 84, 85, 90, 90,
44, 43, 43, 42, 41, 42, 43, 43, 43, 44, 48, 48, 53, 54, 57, 58, 60, 64,
64, 67, 67, 71, 72, 75, 76, 78, 80, 82, 85, 86, 91, 91, 53, 51, 51, 50,
49, 49, 50, 49, 49, 50, 54, 54, 59, 60, 63, 65, 67, 71, 72, 76, 76, 81,
82, 85, 87, 89, 92, 94, 97, 98, 104, 104, 53, 51, 51, 50, 49, 49, 50,
49, 49, 50, 54, 54, 59, 60, 63, 65, 67, 71, 72, 76, 76, 81, 82, 85, 87,
89, 92, 94, 97, 98, 104, 104, 62, 60, 59, 58, 57, 57, 57, 56, 56, 56,
61, 61, 65, 66, 69, 71, 74, 78, 79, 83, 83, 89, 90, 94, 95, 98, 102,
103, 108, 108, 115, 115, 65, 62, 62, 60, 59, 59, 59, 59, 58, 58, 63, 63,
67, 68, 71, 73, 76, 79, 81, 85, 85, 91, 92, 96, 98, 101, 105, 106, 111,
111, 118, 118, 73, 70, 69, 67, 66, 66, 65, 65, 64, 64, 69, 69, 73, 74,
77, 79, 81, 85, 86, 91, 91, 98, 99, 103, 105, 108, 112, 114, 119, 119,
127, 127, 79, 75, 75, 73, 72, 71, 71, 70, 69, 69, 73, 73, 77, 78, 81,

```

```

84, 86, 90, 91, 96, 96, 103, 103, 108, 110, 114, 118, 120, 125, 125,
133, 133,
/* Size 4x16 */
31, 34, 44, 65, 32, 34, 43, 62, 32, 33, 41, 59, 32, 35, 43, 59, 32, 37,
43, 58, 34, 39, 48, 63, 34, 42, 53, 67, 36, 44, 57, 71, 39, 46, 60, 76,
42, 48, 64, 81, 45, 51, 67, 85, 50, 54, 72, 92, 54, 58, 76, 98, 60, 63,
80, 105, 66, 68, 85, 111, 73, 74, 91, 118,
/* Size 16x4 */
31, 32, 32, 32, 32, 34, 34, 36, 39, 42, 45, 50, 54, 60, 66, 73, 34, 34,
33, 35, 37, 39, 42, 44, 46, 48, 51, 54, 58, 63, 68, 74, 44, 43, 41, 43,
43, 48, 53, 57, 60, 64, 67, 72, 76, 80, 85, 91, 65, 62, 59, 59, 58, 63,
67, 71, 76, 81, 85, 92, 98, 105, 111, 118,
/* Size 8x32 */
32, 31, 32, 36, 44, 53, 62, 73, 31, 32, 32, 35, 42, 51, 60, 70, 31, 32,
32, 35, 42, 51, 59, 69, 31, 32, 32, 35, 41, 50, 58, 67, 31, 32, 33, 34,
41, 49, 57, 66, 31, 32, 33, 35, 41, 49, 57, 66, 32, 32, 34, 36, 42, 50,
57, 65, 32, 32, 34, 37, 42, 49, 56, 65, 32, 33, 35, 38, 42, 49, 56, 64,
32, 33, 35, 39, 43, 50, 56, 64, 34, 34, 37, 42, 48, 54, 61, 69, 34, 34,
37, 42, 48, 54, 61, 69, 35, 34, 38, 47, 52, 59, 65, 73, 36, 34, 38, 48,
54, 60, 66, 74, 38, 36, 40, 49, 56, 63, 69, 77, 39, 37, 40, 50, 58, 65,
71, 79, 41, 39, 41, 51, 60, 67, 74, 81, 44, 41, 43, 53, 63, 71, 78, 85,
44, 42, 43, 54, 64, 72, 79, 86, 48, 45, 46, 56, 67, 76, 83, 91, 48, 45,
46, 56, 67, 76, 83, 91, 53, 49, 49, 59, 71, 81, 89, 98, 53, 49, 50, 60,
71, 82, 90, 99, 57, 52, 52, 62, 74, 85, 94, 103, 58, 54, 54, 63, 75, 87,
95, 105, 61, 57, 56, 66, 77, 89, 98, 108, 65, 60, 58, 68, 79, 92, 102,
112, 67, 61, 60, 69, 81, 94, 103, 114, 71, 65, 63, 73, 84, 97, 108, 119,
72, 66, 64, 73, 85, 98, 108, 119, 79, 72, 70, 79, 90, 104, 115, 127, 79,
72, 70, 79, 90, 104, 115, 127,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 35, 36, 38, 39, 41, 44,
44, 48, 48, 53, 53, 57, 58, 61, 65, 67, 71, 72, 79, 79, 31, 32, 32, 32,
32, 32, 32, 32, 33, 33, 34, 34, 34, 34, 36, 37, 39, 41, 42, 45, 45, 49,
49, 52, 54, 57, 60, 61, 65, 66, 72, 72, 32, 32, 32, 32, 33, 33, 34, 34,
35, 35, 37, 37, 38, 38, 40, 40, 41, 43, 43, 46, 46, 49, 50, 52, 54, 56,
58, 60, 63, 64, 70, 70, 36, 35, 35, 35, 34, 35, 36, 37, 38, 39, 42, 42,
47, 48, 49, 50, 51, 53, 54, 56, 56, 59, 60, 62, 63, 66, 68, 69, 73, 73,
79, 79, 44, 42, 42, 41, 41, 41, 42, 42, 42, 43, 48, 48, 52, 54, 56, 58,
60, 63, 64, 67, 67, 71, 71, 74, 75, 77, 79, 81, 84, 85, 90, 90, 53, 51,
51, 50, 49, 49, 50, 49, 49, 50, 54, 54, 59, 60, 63, 65, 67, 71, 72, 76,
76, 81, 82, 85, 87, 89, 92, 94, 97, 98, 104, 104, 62, 60, 59, 58, 57,
57, 57, 56, 56, 56, 61, 61, 65, 66, 69, 71, 74, 78, 79, 83, 83, 89, 90,
94, 95, 98, 102, 103, 108, 108, 115, 115, 73, 70, 69, 67, 66, 66, 65,
65, 64, 64, 69, 69, 73, 74, 77, 79, 81, 85, 86, 91, 91, 98, 99, 103,
105, 108, 112, 114, 119, 119, 127, 127 },
{ /* Chroma */
/* Size 4x4 */
31, 42, 47, 53, 42, 48, 50, 54, 47, 50, 61, 67, 53, 54, 67, 78,
/* Size 8x8 */
31, 32, 38, 48, 47, 50, 53, 57, 32, 35, 42, 47, 45, 47, 50, 54, 38, 42,
47, 48, 45, 47, 49, 52, 48, 47, 48, 53, 53, 54, 56, 58, 47, 45, 45, 53,

```

```

58, 61, 63, 65, 50, 47, 47, 54, 61, 66, 69, 72, 53, 50, 49, 56, 63, 69,
73, 77, 57, 54, 52, 58, 65, 72, 77, 82,
/* Size 16x16 */
32, 31, 30, 33, 36, 41, 47, 49, 49, 49, 50, 52, 54, 57, 60, 63, 31, 31,
31, 34, 38, 42, 46, 47, 47, 47, 48, 50, 52, 54, 57, 60, 30, 31, 32, 35,
40, 42, 45, 46, 45, 45, 46, 47, 49, 52, 54, 57, 33, 34, 35, 39, 43, 45,
47, 46, 46, 45, 46, 47, 49, 51, 53, 56, 36, 38, 40, 43, 47, 47, 47, 47,
46, 45, 46, 47, 48, 50, 52, 54, 41, 42, 42, 45, 47, 48, 50, 50, 49, 49,
50, 50, 52, 53, 55, 57, 47, 46, 45, 47, 47, 50, 52, 52, 52, 52, 53, 53,
55, 56, 58, 60, 49, 47, 46, 46, 47, 50, 52, 53, 54, 55, 55, 56, 57, 58,
60, 62, 49, 47, 45, 46, 46, 49, 52, 54, 55, 57, 58, 59, 60, 61, 63, 65,
49, 47, 45, 45, 45, 49, 52, 55, 57, 59, 60, 61, 63, 64, 66, 68, 50, 48,
46, 46, 46, 50, 53, 55, 58, 60, 61, 63, 65, 67, 68, 71, 52, 50, 47, 47,
47, 50, 53, 56, 59, 61, 63, 66, 68, 70, 72, 75, 54, 52, 49, 49, 48, 52,
55, 57, 60, 63, 65, 68, 71, 73, 75, 78, 57, 54, 52, 51, 50, 53, 56, 58,
61, 64, 67, 70, 73, 76, 79, 82, 60, 57, 54, 53, 52, 55, 58, 60, 63, 66,
68, 72, 75, 79, 82, 85, 63, 60, 57, 56, 54, 57, 60, 62, 65, 68, 71, 75,
78, 82, 85, 89,
/* Size 32x32 */
32, 31, 31, 30, 30, 32, 33, 34, 36, 37, 41, 41, 47, 49, 49, 48, 49, 49,
49, 50, 50, 52, 52, 54, 54, 56, 57, 58, 60, 60, 63, 63, 31, 31, 31, 31,
31, 32, 34, 35, 38, 38, 42, 42, 46, 48, 47, 47, 47, 47, 47, 48, 48, 50,
50, 51, 52, 53, 54, 55, 57, 57, 60, 60, 31, 31, 31, 31, 31, 33, 34, 35,
38, 39, 42, 42, 46, 47, 47, 47, 47, 47, 47, 48, 48, 49, 50, 51, 52, 53,
54, 55, 57, 57, 60, 60, 30, 31, 31, 31, 31, 33, 35, 36, 39, 40, 42, 42,
46, 47, 46, 46, 46, 45, 46, 47, 47, 48, 48, 50, 50, 51, 52, 53, 55, 55,
58, 58, 30, 31, 31, 31, 32, 33, 35, 36, 40, 40, 42, 42, 45, 46, 46, 45,
45, 45, 45, 46, 46, 47, 47, 49, 49, 51, 52, 52, 54, 54, 57, 57, 32, 32,
33, 33, 33, 35, 37, 38, 41, 42, 43, 43, 46, 47, 46, 46, 45, 45, 45, 46,
46, 47, 47, 49, 49, 50, 51, 52, 54, 54, 57, 57, 33, 34, 34, 35, 35, 37,
39, 40, 43, 43, 45, 45, 47, 47, 46, 46, 46, 45, 45, 46, 46, 47, 47, 49,
49, 50, 51, 52, 53, 54, 56, 56, 34, 35, 35, 36, 36, 38, 40, 41, 44, 44,
45, 45, 47, 47, 47, 46, 46, 45, 45, 46, 46, 47, 47, 48, 49, 50, 51, 51,
53, 53, 55, 55, 36, 38, 38, 39, 40, 41, 43, 44, 47, 47, 47, 47, 47, 48,
47, 46, 46, 45, 45, 46, 46, 46, 47, 48, 48, 49, 50, 50, 52, 52, 54, 54,
37, 38, 39, 40, 40, 42, 43, 44, 47, 47, 47, 47, 48, 48, 47, 47, 46, 45,
46, 46, 46, 47, 47, 48, 48, 49, 50, 51, 52, 52, 55, 55, 41, 42, 42, 42,
42, 43, 45, 45, 47, 47, 48, 48, 50, 50, 50, 49, 49, 49, 49, 50, 50, 50,
50, 51, 52, 52, 53, 54, 55, 55, 57, 57, 41, 42, 42, 42, 42, 43, 45, 45,
47, 47, 48, 48, 50, 50, 50, 49, 49, 49, 49, 50, 50, 50, 50, 51, 52, 52,
53, 54, 55, 55, 57, 57, 47, 46, 46, 46, 45, 46, 47, 47, 47, 48, 50, 50,
52, 52, 52, 52, 52, 52, 52, 53, 53, 53, 53, 54, 55, 55, 56, 56, 58, 58,
60, 60, 49, 48, 47, 47, 46, 47, 47, 47, 48, 48, 50, 50, 52, 53, 53, 53,
53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 57, 58, 58, 60, 60, 49, 47,
47, 46, 46, 46, 46, 47, 47, 47, 50, 50, 52, 53, 53, 54, 54, 55, 55,
55, 56, 56, 57, 57, 58, 58, 59, 60, 60, 62, 62, 48, 47, 47, 46, 45, 46,
46, 46, 46, 47, 49, 49, 52, 53, 54, 54, 55, 55, 56, 56, 57, 57, 58,
58, 59, 60, 60, 61, 62, 63, 63, 49, 47, 47, 46, 45, 45, 46, 46, 46, 46,
49, 49, 52, 53, 54, 55, 55, 57, 57, 58, 58, 59, 59, 60, 60, 61, 61, 62,
63, 63, 65, 65, 49, 47, 47, 45, 45, 45, 45, 45, 45, 45, 49, 49, 52, 53,

```



```

55, 55, 57, 58, 59, 60, 60, 61, 61, 62, 62, 63, 63, 64, 65, 65, 67, 67,
49, 47, 47, 46, 45, 45, 45, 45, 45, 46, 49, 49, 52, 53, 55, 56, 57, 59,
59, 60, 60, 61, 61, 62, 63, 63, 64, 65, 66, 66, 68, 68, 50, 48, 48, 47,
46, 46, 46, 46, 46, 46, 50, 50, 53, 54, 55, 56, 58, 60, 60, 61, 61, 63,
63, 65, 65, 66, 67, 67, 68, 69, 71, 71, 50, 48, 48, 47, 46, 46, 46, 46,
46, 46, 50, 50, 53, 54, 55, 56, 58, 60, 60, 61, 61, 63, 63, 65, 65, 66,
67, 67, 68, 69, 71, 71, 52, 50, 49, 48, 47, 47, 47, 47, 46, 47, 50, 50,
53, 54, 56, 57, 59, 61, 61, 63, 63, 66, 66, 67, 68, 69, 70, 71, 72, 72,
74, 74, 52, 50, 50, 48, 47, 47, 47, 47, 47, 47, 50, 50, 53, 54, 56, 57,
59, 61, 61, 63, 63, 66, 66, 68, 68, 69, 70, 71, 72, 73, 75, 75, 54, 51,
51, 50, 49, 49, 49, 48, 48, 48, 51, 51, 54, 55, 57, 58, 60, 62, 62, 65,
65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 77, 54, 52, 52, 50, 49, 49,
49, 49, 48, 48, 52, 52, 55, 55, 57, 58, 60, 62, 63, 65, 65, 68, 68, 70,
71, 72, 73, 74, 75, 76, 78, 78, 56, 53, 53, 51, 51, 50, 50, 50, 49, 49,
52, 52, 55, 56, 58, 59, 61, 63, 63, 66, 66, 69, 69, 71, 72, 73, 75, 75,
77, 77, 80, 80, 57, 54, 54, 52, 52, 51, 51, 51, 50, 50, 53, 53, 56, 56,
58, 60, 61, 63, 64, 67, 67, 70, 70, 72, 73, 75, 76, 77, 79, 79, 82, 82,
58, 55, 55, 53, 52, 52, 52, 51, 50, 51, 54, 54, 56, 57, 59, 60, 62, 64,
65, 67, 67, 71, 71, 73, 74, 75, 77, 78, 80, 80, 83, 83, 60, 57, 57, 55,
54, 54, 53, 53, 52, 52, 55, 55, 58, 58, 60, 61, 63, 65, 66, 68, 68, 72,
72, 74, 75, 77, 79, 80, 82, 82, 85, 85, 60, 57, 57, 55, 54, 54, 54, 53,
52, 52, 55, 55, 58, 58, 60, 62, 63, 65, 66, 69, 69, 72, 73, 75, 76, 77,
79, 80, 82, 82, 85, 85, 63, 60, 60, 58, 57, 57, 56, 55, 54, 55, 57, 57,
60, 60, 62, 63, 65, 67, 68, 71, 71, 74, 75, 77, 78, 80, 82, 83, 85, 85,
89, 89, 63, 60, 60, 58, 57, 57, 56, 55, 54, 55, 57, 57, 60, 60, 62, 63,
65, 67, 68, 71, 71, 74, 75, 77, 78, 80, 82, 83, 85, 85, 89, 89,
/* Size 4x8 */
31, 42, 47, 54, 33, 44, 45, 51, 40, 47, 46, 50, 47, 50, 54, 57, 45, 49,
59, 64, 48, 50, 61, 70, 51, 52, 63, 75, 55, 55, 66, 79,
/* Size 8x4 */
31, 33, 40, 47, 45, 48, 51, 55, 42, 44, 47, 50, 49, 50, 52, 55, 47, 45,
46, 54, 59, 61, 63, 66, 54, 51, 50, 57, 64, 70, 75, 79,
/* Size 8x16 */
32, 31, 37, 48, 49, 52, 56, 61, 31, 31, 38, 47, 47, 50, 53, 57, 30, 32,
40, 46, 45, 48, 51, 55, 33, 36, 43, 47, 46, 47, 50, 54, 37, 40, 47, 47,
45, 47, 49, 52, 42, 43, 47, 50, 49, 50, 53, 56, 47, 46, 48, 52, 53, 53,
55, 58, 48, 46, 47, 53, 55, 56, 58, 61, 48, 45, 46, 53, 57, 59, 61, 63,
49, 45, 46, 53, 58, 62, 64, 66, 50, 46, 46, 54, 59, 64, 66, 69, 52, 48,
47, 54, 61, 66, 70, 73, 54, 50, 49, 55, 62, 68, 72, 76, 57, 52, 50, 56,
64, 70, 75, 79, 60, 54, 52, 58, 65, 72, 77, 82, 63, 57, 55, 60, 67, 75,
80, 86,
/* Size 16x8 */
32, 31, 30, 33, 37, 42, 47, 48, 48, 49, 50, 52, 54, 57, 60, 63, 31, 31,
32, 36, 40, 43, 46, 46, 45, 45, 46, 48, 50, 52, 54, 57, 37, 38, 40, 43,
47, 47, 48, 47, 46, 46, 46, 47, 49, 50, 52, 55, 48, 47, 46, 47, 47, 50,
52, 53, 53, 53, 54, 54, 55, 56, 58, 60, 49, 47, 45, 46, 45, 49, 53, 55,
57, 58, 59, 61, 62, 64, 65, 67, 52, 50, 48, 47, 47, 50, 53, 56, 59, 62,
64, 66, 68, 70, 72, 75, 56, 53, 51, 50, 49, 53, 55, 58, 61, 64, 66, 70,
72, 75, 77, 80, 61, 57, 55, 54, 52, 56, 58, 61, 63, 66, 69, 73, 76, 79,
82, 86,

```

```
/* Size 16x32 */
```

```
32, 31, 31, 35, 37, 42, 48, 48, 49, 49, 52, 52, 56, 57, 61, 63, 31, 31,
31, 36, 38, 42, 47, 47, 47, 47, 50, 50, 54, 54, 58, 60, 31, 31, 31, 36,
38, 42, 47, 47, 47, 47, 50, 50, 53, 54, 57, 60, 30, 32, 32, 37, 39, 42,
46, 46, 46, 46, 48, 48, 52, 52, 56, 58, 30, 32, 32, 37, 40, 42, 46, 46,
45, 45, 48, 48, 51, 52, 55, 57, 32, 33, 34, 39, 41, 44, 46, 46, 45, 45,
48, 48, 51, 51, 54, 57, 33, 35, 36, 40, 43, 45, 47, 46, 46, 46, 47, 47,
50, 51, 54, 56, 34, 37, 37, 42, 44, 45, 47, 47, 45, 46, 47, 47, 50, 51,
53, 55, 37, 40, 40, 45, 47, 47, 47, 47, 45, 46, 47, 47, 49, 50, 52, 54,
37, 40, 40, 45, 47, 47, 48, 47, 46, 46, 47, 47, 49, 50, 53, 55, 42, 43,
43, 46, 47, 48, 50, 50, 49, 49, 50, 50, 53, 53, 56, 57, 42, 43, 43, 46,
47, 48, 50, 50, 49, 49, 50, 50, 53, 53, 56, 57, 47, 46, 46, 47, 48, 50,
52, 52, 53, 53, 53, 53, 55, 56, 58, 60, 49, 47, 46, 47, 48, 50, 53, 53,
53, 54, 54, 54, 56, 57, 59, 60, 48, 46, 46, 47, 47, 50, 53, 53, 55, 55,
56, 56, 58, 58, 61, 62, 48, 46, 46, 46, 47, 50, 53, 54, 56, 56, 57, 57,
59, 60, 62, 64, 48, 46, 45, 46, 46, 49, 53, 54, 57, 57, 59, 59, 61, 61,
63, 65, 49, 45, 45, 45, 46, 49, 53, 55, 58, 59, 61, 61, 63, 64, 66, 67,
49, 46, 45, 46, 46, 49, 53, 55, 58, 59, 62, 62, 64, 64, 66, 68, 50, 47,
46, 46, 46, 50, 54, 55, 59, 60, 64, 64, 66, 67, 69, 71, 50, 47, 46, 46,
46, 50, 54, 55, 59, 60, 64, 64, 66, 67, 69, 71, 52, 48, 48, 47, 47, 50,
54, 56, 61, 61, 66, 66, 69, 70, 72, 74, 52, 48, 48, 47, 47, 50, 54, 56,
61, 61, 66, 66, 70, 71, 73, 75, 53, 50, 49, 48, 48, 51, 55, 57, 62, 62,
68, 68, 71, 72, 75, 77, 54, 50, 50, 49, 49, 52, 55, 57, 62, 63, 68, 68,
72, 73, 76, 78, 55, 51, 51, 50, 49, 52, 56, 58, 63, 63, 69, 69, 74, 75,
78, 80, 57, 52, 52, 51, 50, 53, 56, 58, 64, 64, 70, 70, 75, 76, 79, 82,
58, 53, 53, 51, 51, 54, 57, 59, 64, 65, 71, 71, 76, 77, 80, 83, 60, 55,
54, 53, 52, 55, 58, 60, 65, 66, 72, 72, 77, 79, 82, 85, 60, 55, 55, 53,
53, 55, 59, 60, 65, 66, 73, 73, 78, 79, 83, 85, 63, 58, 57, 56, 55, 58,
60, 62, 67, 68, 75, 75, 80, 82, 86, 89, 63, 58, 57, 56, 55, 58, 60, 62,
67, 68, 75, 75, 80, 82, 86, 89,
```

```
/* Size 32x16 */
```

```
32, 31, 31, 30, 30, 32, 33, 34, 37, 37, 42, 42, 47, 49, 48, 48, 48, 49,
49, 50, 50, 52, 52, 53, 54, 55, 57, 58, 60, 60, 63, 63, 31, 31, 31, 32,
32, 33, 35, 37, 40, 40, 43, 43, 46, 47, 46, 46, 46, 45, 46, 47, 47, 48,
48, 50, 50, 51, 52, 53, 55, 55, 58, 58, 31, 31, 31, 32, 32, 34, 36, 37,
40, 40, 43, 43, 46, 46, 46, 46, 45, 45, 45, 46, 46, 48, 48, 49, 50, 51,
52, 53, 54, 55, 57, 57, 35, 36, 36, 37, 37, 39, 40, 42, 45, 45, 46, 46,
47, 47, 47, 46, 46, 45, 46, 46, 46, 47, 47, 48, 49, 50, 51, 51, 53, 53,
56, 56, 37, 38, 38, 39, 40, 41, 43, 44, 47, 47, 47, 47, 48, 48, 47, 47,
46, 46, 46, 46, 46, 47, 47, 48, 49, 49, 50, 51, 52, 53, 55, 55, 42, 42,
42, 42, 42, 44, 45, 45, 47, 47, 48, 48, 50, 50, 50, 50, 49, 49, 49, 50,
50, 50, 50, 51, 52, 52, 53, 54, 55, 55, 58, 58, 48, 47, 47, 46, 46, 46,
47, 47, 47, 48, 50, 50, 52, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55,
55, 56, 56, 57, 58, 59, 60, 60, 48, 47, 47, 46, 46, 46, 46, 47, 47, 47,
50, 50, 52, 53, 53, 54, 54, 55, 55, 55, 55, 56, 56, 57, 57, 58, 58, 59,
60, 60, 62, 62, 49, 47, 47, 46, 45, 45, 46, 45, 45, 46, 49, 49, 53, 53,
55, 56, 57, 58, 58, 59, 59, 61, 61, 62, 62, 63, 64, 64, 65, 65, 67, 67,
49, 47, 47, 46, 45, 45, 46, 46, 46, 46, 49, 49, 53, 54, 55, 56, 57, 59,
59, 60, 60, 61, 61, 62, 63, 63, 64, 65, 66, 66, 68, 68, 52, 50, 50, 48,
48, 48, 47, 47, 47, 47, 50, 50, 53, 54, 56, 57, 59, 61, 62, 64, 64, 66,
```

```

66, 68, 68, 69, 70, 71, 72, 73, 75, 75, 52, 50, 50, 48, 48, 48, 47, 47,
47, 47, 50, 50, 53, 54, 56, 57, 59, 61, 62, 64, 64, 66, 66, 68, 68, 69,
70, 71, 72, 73, 75, 75, 56, 54, 53, 52, 51, 51, 50, 50, 49, 49, 53, 53,
55, 56, 58, 59, 61, 63, 64, 66, 66, 69, 70, 71, 72, 74, 75, 76, 77, 78,
80, 80, 57, 54, 54, 52, 52, 51, 51, 51, 50, 50, 53, 53, 56, 57, 58, 60,
61, 64, 64, 67, 67, 70, 71, 72, 73, 75, 76, 77, 79, 79, 82, 82, 61, 58,
57, 56, 55, 54, 54, 53, 52, 53, 56, 56, 58, 59, 61, 62, 63, 66, 66, 69,
69, 72, 73, 75, 76, 78, 79, 80, 82, 83, 86, 86, 63, 60, 60, 58, 57, 57,
56, 55, 54, 55, 57, 57, 60, 60, 62, 64, 65, 67, 68, 71, 71, 74, 75, 77,
78, 80, 82, 83, 85, 85, 89, 89,
/* Size 4x16 */
31, 42, 49, 57, 31, 42, 47, 54, 32, 42, 45, 52, 35, 45, 46, 51, 40, 47,
46, 50, 43, 48, 49, 53, 46, 50, 53, 56, 46, 50, 55, 58, 46, 49, 57, 61,
46, 49, 59, 64, 47, 50, 60, 67, 48, 50, 61, 71, 50, 52, 63, 73, 52, 53,
64, 76, 55, 55, 66, 79, 58, 58, 68, 82,
/* Size 16x4 */
31, 31, 32, 35, 40, 43, 46, 46, 46, 46, 47, 48, 50, 52, 55, 58, 42, 42,
42, 45, 47, 48, 50, 50, 49, 49, 50, 50, 52, 53, 55, 58, 49, 47, 45, 46,
46, 49, 53, 55, 57, 59, 60, 61, 63, 64, 66, 68, 57, 54, 52, 51, 50, 53,
56, 58, 61, 64, 67, 71, 73, 76, 79, 82,
/* Size 8x32 */
32, 31, 37, 48, 49, 52, 56, 61, 31, 31, 38, 47, 47, 50, 54, 58, 31, 31,
38, 47, 47, 50, 53, 57, 30, 32, 39, 46, 46, 48, 52, 56, 30, 32, 40, 46,
45, 48, 51, 55, 32, 34, 41, 46, 45, 48, 51, 54, 33, 36, 43, 47, 46, 47,
50, 54, 34, 37, 44, 47, 45, 47, 50, 53, 37, 40, 47, 47, 45, 47, 49, 52,
37, 40, 47, 48, 46, 47, 49, 53, 42, 43, 47, 50, 49, 50, 53, 56, 42, 43,
47, 50, 49, 50, 53, 56, 47, 46, 48, 52, 53, 53, 55, 58, 49, 46, 48, 53,
53, 54, 56, 59, 48, 46, 47, 53, 55, 56, 58, 61, 48, 46, 47, 53, 56, 57,
59, 62, 48, 45, 46, 53, 57, 59, 61, 63, 49, 45, 46, 53, 58, 61, 63, 66,
49, 45, 46, 53, 58, 62, 64, 66, 50, 46, 46, 54, 59, 64, 66, 69, 50, 46,
46, 54, 59, 64, 66, 69, 52, 48, 47, 54, 61, 66, 69, 72, 52, 48, 47, 54,
61, 66, 70, 73, 53, 49, 48, 55, 62, 68, 71, 75, 54, 50, 49, 55, 62, 68,
72, 76, 55, 51, 49, 56, 63, 69, 74, 78, 57, 52, 50, 56, 64, 70, 75, 79,
58, 53, 51, 57, 64, 71, 76, 80, 60, 54, 52, 58, 65, 72, 77, 82, 60, 55,
53, 59, 65, 73, 78, 83, 63, 57, 55, 60, 67, 75, 80, 86, 63, 57, 55, 60,
67, 75, 80, 86,
/* Size 32x8 */
32, 31, 31, 30, 30, 32, 33, 34, 37, 37, 42, 42, 47, 49, 48, 48, 48, 49,
49, 50, 50, 52, 52, 53, 54, 55, 57, 58, 60, 60, 63, 63, 31, 31, 31, 32,
32, 34, 36, 37, 40, 40, 43, 43, 46, 46, 46, 46, 45, 45, 45, 46, 46, 48,
48, 49, 50, 51, 52, 53, 54, 55, 57, 57, 37, 38, 38, 39, 40, 41, 43, 44,
47, 47, 47, 47, 48, 48, 47, 47, 46, 46, 46, 46, 46, 47, 47, 48, 49, 49,
50, 51, 52, 53, 55, 55, 48, 47, 47, 46, 46, 46, 47, 47, 47, 48, 50, 50,
52, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 57, 58, 59,
60, 60, 49, 47, 47, 46, 45, 45, 46, 45, 45, 46, 49, 49, 53, 53, 55, 56,
57, 58, 58, 59, 59, 61, 61, 62, 62, 63, 64, 64, 65, 65, 67, 67, 52, 50,
50, 48, 48, 48, 47, 47, 47, 47, 50, 50, 53, 54, 56, 57, 59, 61, 62, 64,
64, 66, 66, 68, 68, 69, 70, 71, 72, 73, 75, 75, 56, 54, 53, 52, 51, 51,
50, 50, 49, 49, 53, 53, 55, 56, 58, 59, 61, 63, 64, 66, 66, 69, 70, 71,
72, 74, 75, 76, 77, 78, 80, 80, 61, 58, 57, 56, 55, 54, 54, 53, 52, 53,

```

```

56, 56, 58, 59, 61, 62, 63, 66, 66, 69, 69, 72, 73, 75, 76, 78, 79, 80,
82, 83, 86, 86 },
},
{
  /* Luma */
  /* Size 4x4 */
  32, 33, 42, 55, 33, 38, 46, 57, 42, 46, 63, 75, 55, 57, 75, 92,
  /* Size 8x8 */
  31, 32, 32, 34, 38, 46, 52, 63, 32, 32, 32, 34, 37, 44, 49, 59, 32, 32,
  35, 37, 40, 45, 49, 58, 34, 34, 37, 42, 47, 52, 56, 65, 38, 37, 40, 47,
  54, 60, 65, 73, 46, 44, 45, 52, 60, 69, 75, 84, 52, 49, 49, 56, 65, 75,
  82, 92, 63, 59, 58, 65, 73, 84, 92, 105,
  /* Size 16x16 */
  32, 31, 31, 31, 32, 32, 34, 36, 38, 41, 44, 48, 54, 58, 61, 65, 31, 32,
  32, 32, 32, 32, 34, 35, 38, 40, 42, 46, 51, 55, 58, 62, 31, 32, 32, 32,
  32, 32, 33, 34, 37, 38, 41, 44, 49, 53, 56, 59, 31, 32, 32, 33, 33, 33,
  35, 36, 38, 40, 42, 45, 49, 53, 56, 59, 32, 32, 32, 33, 34, 34, 36, 37,
  39, 40, 42, 45, 49, 53, 55, 59, 32, 32, 32, 33, 34, 35, 37, 38, 40, 41,
  42, 46, 49, 52, 55, 58, 34, 34, 33, 35, 36, 37, 39, 42, 44, 46, 47, 51,
  54, 57, 60, 63, 36, 35, 34, 36, 37, 38, 42, 48, 50, 52, 54, 57, 60, 63,
  65, 68, 38, 38, 37, 38, 39, 40, 44, 50, 52, 54, 57, 60, 64, 67, 69, 72,
  41, 40, 38, 40, 40, 41, 46, 52, 54, 57, 60, 63, 67, 70, 73, 75, 44, 42,
  41, 42, 42, 42, 47, 54, 57, 60, 63, 67, 71, 74, 77, 79, 48, 46, 44, 45,
  45, 46, 51, 57, 60, 63, 67, 71, 76, 79, 82, 85, 54, 51, 49, 49, 49, 49,
  54, 60, 64, 67, 71, 76, 82, 86, 89, 92, 58, 55, 53, 53, 53, 52, 57, 63,
  67, 70, 74, 79, 86, 90, 93, 97, 61, 58, 56, 56, 55, 55, 60, 65, 69, 73,
  77, 82, 89, 93, 97, 101, 65, 62, 59, 59, 59, 58, 63, 68, 72, 75, 79, 85,
  92, 97, 101, 105,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 34, 36, 36, 38, 39,
  41, 44, 44, 47, 48, 50, 54, 54, 58, 59, 61, 65, 65, 70, 31, 31, 31, 32,
  32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 38, 38, 40, 42, 42, 46,
  47, 49, 52, 52, 56, 57, 59, 63, 63, 67, 31, 31, 32, 32, 32, 32, 32, 32,
  32, 32, 33, 34, 34, 35, 35, 38, 38, 40, 42, 42, 45, 46, 48, 51, 51,
  55, 56, 58, 62, 62, 67, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
  34, 34, 35, 35, 37, 38, 39, 42, 42, 45, 45, 47, 50, 50, 54, 55, 57, 61,
  61, 65, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34,
  37, 37, 38, 41, 41, 44, 44, 46, 49, 49, 53, 54, 56, 59, 59, 64, 31, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34, 37, 37, 38, 41,
  41, 44, 44, 46, 49, 49, 53, 54, 56, 59, 59, 64, 31, 32, 32, 32, 32, 32,
  33, 33, 33, 33, 33, 34, 35, 35, 36, 36, 38, 39, 40, 42, 42, 44, 45, 47,
  49, 49, 53, 54, 56, 59, 59, 63, 31, 32, 32, 32, 32, 32, 33, 33, 33, 34,
  34, 35, 35, 36, 36, 36, 38, 39, 40, 42, 42, 45, 45, 47, 50, 50, 53, 54,
  56, 59, 59, 63, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34, 35, 36, 36,
  37, 37, 39, 39, 40, 42, 42, 45, 45, 47, 49, 49, 53, 54, 55, 59, 59, 63,
  32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 36, 37, 37, 38, 38, 40, 40,
  41, 42, 42, 45, 46, 47, 49, 49, 52, 53, 55, 58, 58, 62, 32, 32, 32, 32,
  32, 32, 33, 34, 34, 35, 35, 36, 37, 37, 38, 38, 40, 40, 41, 42, 42, 45,
  46, 47, 49, 49, 52, 53, 55, 58, 58, 62, 33, 33, 33, 33, 33, 33, 34, 35,
  35, 36, 36, 38, 39, 40, 42, 42, 43, 44, 45, 46, 46, 49, 50, 51, 53, 53,

```

```

56, 57, 59, 62, 62, 66, 34, 34, 34, 34, 33, 33, 35, 35, 36, 37, 37, 39,
39, 41, 42, 42, 44, 45, 46, 47, 47, 50, 51, 52, 54, 54, 57, 58, 60, 63,
63, 67, 34, 34, 34, 34, 34, 34, 35, 36, 36, 37, 37, 40, 41, 42, 45, 45,
46, 47, 48, 50, 50, 52, 53, 54, 56, 56, 59, 60, 62, 65, 65, 69, 36, 35,
35, 35, 34, 34, 36, 36, 37, 38, 38, 42, 42, 45, 48, 48, 50, 50, 52, 54,
54, 56, 57, 58, 60, 60, 63, 64, 65, 68, 68, 72, 36, 35, 35, 35, 34, 34,
36, 36, 37, 38, 38, 42, 42, 45, 48, 48, 50, 50, 52, 54, 54, 56, 57, 58,
60, 60, 63, 64, 65, 68, 68, 72, 38, 38, 38, 37, 37, 37, 37, 38, 38, 39, 40,
40, 43, 44, 46, 50, 50, 52, 53, 54, 57, 57, 59, 60, 61, 64, 64, 67, 68,
69, 72, 72, 76, 39, 38, 38, 38, 37, 37, 39, 39, 39, 40, 40, 44, 45, 47,
50, 50, 53, 54, 55, 58, 58, 60, 61, 62, 65, 65, 68, 69, 70, 73, 73, 77,
41, 40, 40, 39, 38, 38, 40, 40, 40, 41, 41, 45, 46, 48, 52, 52, 54, 55,
57, 60, 60, 62, 63, 65, 67, 67, 70, 71, 73, 75, 75, 79, 44, 42, 42, 42,
41, 41, 42, 42, 42, 42, 42, 46, 47, 50, 54, 54, 57, 58, 60, 63, 63, 66,
67, 68, 71, 71, 74, 75, 77, 79, 79, 83, 44, 42, 42, 42, 41, 41, 42, 42,
42, 42, 42, 46, 47, 50, 54, 54, 57, 58, 60, 63, 63, 66, 67, 68, 71, 71,
74, 75, 77, 79, 79, 83, 47, 46, 45, 45, 44, 44, 44, 45, 45, 45, 45, 49,
50, 52, 56, 56, 59, 60, 62, 66, 66, 69, 70, 72, 75, 75, 78, 79, 81, 84,
84, 88, 48, 47, 46, 45, 44, 44, 45, 45, 45, 46, 46, 50, 51, 53, 57, 57,
60, 61, 63, 67, 67, 70, 71, 73, 76, 76, 79, 80, 82, 85, 85, 89, 50, 49,
48, 47, 46, 46, 47, 47, 47, 47, 47, 51, 52, 54, 58, 58, 61, 62, 65, 68,
68, 72, 73, 75, 78, 78, 82, 83, 85, 88, 88, 92, 54, 52, 51, 50, 49, 49,
49, 50, 49, 49, 49, 53, 54, 56, 60, 60, 64, 65, 67, 71, 71, 75, 76, 78,
82, 82, 86, 87, 89, 92, 92, 96, 54, 52, 51, 50, 49, 49, 49, 50, 49, 49,
49, 53, 54, 56, 60, 60, 64, 65, 67, 71, 71, 75, 76, 78, 82, 82, 86, 87,
89, 92, 92, 96, 58, 56, 55, 54, 53, 53, 53, 53, 53, 52, 52, 56, 57, 59,
63, 63, 67, 68, 70, 74, 74, 78, 79, 82, 86, 86, 90, 91, 93, 97, 97, 101,
59, 57, 56, 55, 54, 54, 54, 54, 54, 53, 53, 57, 58, 60, 64, 64, 68, 69,
71, 75, 75, 79, 80, 83, 87, 87, 91, 92, 94, 98, 98, 102, 61, 59, 58, 57,
56, 56, 56, 56, 55, 55, 55, 59, 60, 62, 65, 65, 69, 70, 73, 77, 77, 81,
82, 85, 89, 89, 93, 94, 97, 101, 101, 105, 65, 63, 62, 61, 59, 59, 59,
59, 59, 58, 58, 62, 63, 65, 68, 68, 72, 73, 75, 79, 79, 84, 85, 88, 92,
92, 97, 98, 101, 105, 105, 109, 65, 63, 62, 61, 59, 59, 59, 59, 59, 58,
58, 62, 63, 65, 68, 68, 72, 73, 75, 79, 79, 84, 85, 88, 92, 92, 97, 98,
101, 105, 105, 109, 70, 67, 67, 65, 64, 64, 63, 63, 63, 62, 62, 66, 67,
69, 72, 72, 76, 77, 79, 83, 83, 88, 89, 92, 96, 96, 101, 102, 105, 109,
109, 114,
/* Size 4x8 */
32, 32, 42, 56, 32, 33, 41, 53, 32, 35, 42, 52, 34, 37, 50, 59, 38, 40,
58, 68, 44, 45, 66, 78, 50, 50, 71, 86, 61, 58, 79, 97,
/* Size 8x4 */
32, 32, 32, 34, 38, 44, 50, 61, 32, 33, 35, 37, 40, 45, 50, 58, 42, 41,
42, 50, 58, 66, 71, 79, 56, 53, 52, 59, 68, 78, 86, 97,
/* Size 8x16 */
32, 31, 32, 35, 39, 44, 53, 65, 31, 32, 32, 35, 38, 42, 51, 62, 31, 32,
33, 34, 37, 41, 49, 59, 31, 32, 34, 35, 38, 42, 49, 59, 32, 32, 34, 36,
39, 42, 49, 58, 32, 33, 35, 37, 40, 42, 49, 58, 34, 34, 37, 41, 44, 48,
54, 63, 36, 34, 38, 46, 50, 54, 60, 68, 38, 37, 40, 47, 52, 57, 64, 72,
41, 39, 41, 49, 54, 60, 67, 76, 44, 41, 43, 51, 57, 63, 71, 79, 48, 45,
46, 54, 60, 67, 76, 85, 53, 49, 50, 57, 64, 71, 82, 92, 57, 53, 53, 60,

```

67, 74, 86, 97, 61, 56, 56, 63, 69, 77, 89, 100, 65, 60, 58, 66, 72, 79,  
92, 105,

*/\* Size 16x8 \*/*

32, 31, 31, 31, 32, 32, 34, 36, 38, 41, 44, 48, 53, 57, 61, 65, 31, 32,  
32, 32, 32, 33, 34, 34, 37, 39, 41, 45, 49, 53, 56, 60, 32, 32, 33, 34,  
34, 35, 37, 38, 40, 41, 43, 46, 50, 53, 56, 58, 35, 35, 34, 35, 36, 37,  
41, 46, 47, 49, 51, 54, 57, 60, 63, 66, 39, 38, 37, 38, 39, 40, 44, 50,  
52, 54, 57, 60, 64, 67, 69, 72, 44, 42, 41, 42, 42, 42, 48, 54, 57, 60,  
63, 67, 71, 74, 77, 79, 53, 51, 49, 49, 49, 49, 54, 60, 64, 67, 71, 76,  
82, 86, 89, 92, 65, 62, 59, 59, 58, 58, 63, 68, 72, 76, 79, 85, 92, 97,  
100, 105,

*/\* Size 16x32 \*/*

32, 31, 31, 31, 32, 32, 35, 36, 39, 44, 44, 51, 53, 58, 65, 65, 31, 32,  
32, 32, 32, 32, 35, 35, 38, 42, 42, 49, 52, 56, 63, 63, 31, 32, 32, 32,  
32, 32, 35, 35, 38, 42, 42, 49, 51, 55, 62, 62, 31, 32, 32, 32, 32, 32,  
34, 35, 37, 41, 41, 48, 50, 54, 61, 61, 31, 32, 32, 32, 33, 33, 34, 34,  
37, 41, 41, 47, 49, 53, 59, 59, 31, 32, 32, 32, 33, 33, 34, 34, 37, 41,  
41, 47, 49, 53, 59, 59, 31, 32, 32, 33, 34, 34, 35, 36, 38, 42, 42, 48,  
49, 53, 59, 59, 32, 32, 32, 33, 34, 34, 36, 36, 38, 42, 42, 48, 50, 53,  
59, 59, 32, 32, 32, 33, 34, 34, 36, 37, 39, 42, 42, 48, 49, 53, 58, 58,  
32, 32, 33, 34, 35, 35, 37, 38, 40, 42, 42, 48, 49, 52, 58, 58, 32, 32,  
33, 34, 35, 35, 37, 38, 40, 42, 42, 48, 49, 52, 58, 58, 33, 33, 33, 35,  
36, 36, 40, 41, 43, 46, 46, 52, 53, 56, 62, 62, 34, 34, 34, 35, 37, 37,  
41, 42, 44, 48, 48, 53, 54, 57, 63, 63, 34, 34, 34, 35, 37, 37, 43, 44,  
46, 50, 50, 55, 56, 59, 65, 65, 36, 35, 34, 36, 38, 38, 46, 48, 50, 54,  
54, 58, 60, 63, 68, 68, 36, 35, 34, 36, 38, 38, 46, 48, 50, 54, 54, 58,  
60, 63, 68, 68, 38, 37, 37, 38, 40, 40, 47, 50, 52, 57, 57, 62, 64, 67,  
72, 72, 39, 38, 37, 39, 40, 40, 48, 50, 53, 58, 58, 63, 65, 68, 73, 73,  
41, 39, 39, 40, 41, 41, 49, 51, 54, 60, 60, 66, 67, 70, 76, 76, 44, 41,  
41, 42, 43, 43, 51, 53, 57, 63, 63, 69, 71, 74, 79, 79, 44, 41, 41, 42,  
43, 43, 51, 53, 57, 63, 63, 69, 71, 74, 79, 79, 47, 44, 44, 44, 45, 45,  
53, 56, 59, 66, 66, 73, 75, 78, 84, 84, 48, 45, 45, 45, 46, 46, 54, 56,  
60, 67, 67, 74, 76, 79, 85, 85, 50, 47, 46, 47, 47, 47, 55, 58, 61, 68,  
68, 76, 78, 82, 88, 88, 53, 50, 49, 50, 50, 50, 57, 60, 64, 71, 71, 79,  
82, 86, 92, 92, 53, 50, 49, 50, 50, 50, 57, 60, 64, 71, 71, 79, 82, 86,  
92, 92, 57, 54, 53, 53, 53, 53, 60, 63, 67, 74, 74, 83, 86, 90, 97, 97,  
58, 55, 54, 54, 54, 54, 61, 63, 68, 75, 75, 84, 87, 91, 98, 98, 61, 57,  
56, 56, 56, 56, 63, 65, 69, 77, 77, 86, 89, 93, 100, 100, 65, 61, 60,  
59, 58, 58, 66, 68, 72, 79, 79, 89, 92, 97, 105, 105, 65, 61, 60, 59,  
58, 58, 66, 68, 72, 79, 79, 89, 92, 97, 105, 105, 70, 65, 64, 63, 62,  
62, 70, 72, 76, 83, 83, 93, 96, 101, 109, 109,

*/\* Size 32x16 \*/*

32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 33, 34, 34, 36, 36, 38, 39,  
41, 44, 44, 47, 48, 50, 53, 53, 57, 58, 61, 65, 65, 70, 31, 32, 32, 32,  
32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 37, 38, 39, 41, 41, 44,  
45, 47, 50, 50, 54, 55, 57, 61, 61, 65, 31, 32, 32, 32, 32, 32, 32,  
32, 33, 33, 33, 34, 34, 34, 34, 37, 37, 39, 41, 41, 44, 45, 46, 49, 49,  
53, 54, 56, 60, 60, 64, 31, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 35,  
35, 35, 36, 36, 38, 39, 40, 42, 42, 44, 45, 47, 50, 50, 53, 54, 56, 59,  
59, 63, 32, 32, 32, 32, 33, 33, 34, 34, 34, 35, 35, 36, 37, 37, 38, 38,

```

40, 40, 41, 43, 43, 45, 46, 47, 50, 50, 53, 54, 56, 58, 58, 62, 32, 32,
32, 32, 33, 33, 34, 34, 34, 35, 35, 36, 37, 37, 38, 38, 40, 40, 41, 43,
43, 45, 46, 47, 50, 50, 53, 54, 56, 58, 58, 62, 35, 35, 35, 34, 34, 34,
35, 36, 36, 37, 37, 40, 41, 43, 46, 46, 47, 48, 49, 51, 51, 53, 54, 55,
57, 57, 60, 61, 63, 66, 66, 70, 36, 35, 35, 35, 34, 34, 36, 36, 37, 38,
38, 41, 42, 44, 48, 48, 50, 50, 51, 53, 53, 56, 56, 58, 60, 60, 63, 63,
65, 68, 68, 72, 39, 38, 38, 37, 37, 37, 38, 38, 39, 40, 40, 43, 44, 46,
50, 50, 52, 53, 54, 57, 57, 59, 60, 61, 64, 64, 67, 68, 69, 72, 72, 76,
44, 42, 42, 41, 41, 41, 42, 42, 42, 42, 42, 46, 48, 50, 54, 54, 57, 58,
60, 63, 63, 66, 67, 68, 71, 71, 74, 75, 77, 79, 79, 83, 44, 42, 42, 41,
41, 41, 42, 42, 42, 42, 42, 46, 48, 50, 54, 54, 57, 58, 60, 63, 63, 66,
67, 68, 71, 71, 74, 75, 77, 79, 79, 83, 51, 49, 49, 48, 47, 47, 48, 48,
48, 48, 48, 52, 53, 55, 58, 58, 62, 63, 66, 69, 69, 73, 74, 76, 79, 79,
83, 84, 86, 89, 89, 93, 53, 52, 51, 50, 49, 49, 49, 50, 49, 49, 49, 53,
54, 56, 60, 60, 64, 65, 67, 71, 71, 75, 76, 78, 82, 82, 86, 87, 89, 92,
92, 96, 58, 56, 55, 54, 53, 53, 53, 53, 53, 52, 52, 56, 57, 59, 63, 63,
67, 68, 70, 74, 74, 78, 79, 82, 86, 86, 90, 91, 93, 97, 97, 101, 65, 63,
62, 61, 59, 59, 59, 59, 58, 58, 58, 62, 63, 65, 68, 68, 72, 73, 76, 79,
79, 84, 85, 88, 92, 92, 97, 98, 100, 105, 105, 109, 65, 63, 62, 61, 59,
59, 59, 59, 58, 58, 58, 62, 63, 65, 68, 68, 72, 73, 76, 79, 79, 84, 85,
88, 92, 92, 97, 98, 100, 105, 105, 109,
/* Size 4x16 */
31, 32, 44, 58, 32, 32, 42, 55, 32, 33, 41, 53, 32, 34, 42, 53, 32, 34,
42, 53, 32, 35, 42, 52, 34, 37, 48, 57, 35, 38, 54, 63, 37, 40, 57, 67,
39, 41, 60, 70, 41, 43, 63, 74, 45, 46, 67, 79, 50, 50, 71, 86, 54, 53,
74, 90, 57, 56, 77, 93, 61, 58, 79, 97,
/* Size 16x4 */
31, 32, 32, 32, 32, 32, 34, 35, 37, 39, 41, 45, 50, 54, 57, 61, 32, 32,
33, 34, 34, 35, 37, 38, 40, 41, 43, 46, 50, 53, 56, 58, 44, 42, 41, 42,
42, 42, 48, 54, 57, 60, 63, 67, 71, 74, 77, 79, 58, 55, 53, 53, 53, 52,
57, 63, 67, 70, 74, 79, 86, 90, 93, 97,
/* Size 8x32 */
32, 31, 32, 35, 39, 44, 53, 65, 31, 32, 32, 35, 38, 42, 52, 63, 31, 32,
32, 35, 38, 42, 51, 62, 31, 32, 32, 34, 37, 41, 50, 61, 31, 32, 33, 34,
37, 41, 49, 59, 31, 32, 33, 34, 37, 41, 49, 59, 31, 32, 34, 35, 38, 42,
49, 59, 32, 32, 34, 36, 38, 42, 50, 59, 32, 32, 34, 36, 39, 42, 49, 58,
32, 33, 35, 37, 40, 42, 49, 58, 32, 33, 35, 37, 40, 42, 49, 58, 33, 33,
36, 40, 43, 46, 53, 62, 34, 34, 37, 41, 44, 48, 54, 63, 34, 34, 37, 43,
46, 50, 56, 65, 36, 34, 38, 46, 50, 54, 60, 68, 36, 34, 38, 46, 50, 54,
60, 68, 38, 37, 40, 47, 52, 57, 64, 72, 39, 37, 40, 48, 53, 58, 65, 73,
41, 39, 41, 49, 54, 60, 67, 76, 44, 41, 43, 51, 57, 63, 71, 79, 44, 41,
43, 51, 57, 63, 71, 79, 47, 44, 45, 53, 59, 66, 75, 84, 48, 45, 46, 54,
60, 67, 76, 85, 50, 46, 47, 55, 61, 68, 78, 88, 53, 49, 50, 57, 64, 71,
82, 92, 53, 49, 50, 57, 64, 71, 82, 92, 57, 53, 53, 60, 67, 74, 86, 97,
58, 54, 54, 61, 68, 75, 87, 98, 61, 56, 56, 63, 69, 77, 89, 100, 65, 60,
58, 66, 72, 79, 92, 105, 65, 60, 58, 66, 72, 79, 92, 105, 70, 64, 62,
70, 76, 83, 96, 109,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 33, 34, 34, 36, 36, 38, 39,
41, 44, 44, 47, 48, 50, 53, 53, 57, 58, 61, 65, 65, 70, 31, 32, 32, 32,

```

```

32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 37, 37, 39, 41, 41, 44,
45, 46, 49, 49, 53, 54, 56, 60, 60, 64, 32, 32, 32, 32, 33, 33, 34, 34,
34, 35, 35, 36, 37, 37, 38, 38, 40, 40, 41, 43, 43, 45, 46, 47, 50, 50,
53, 54, 56, 58, 58, 62, 35, 35, 35, 34, 34, 34, 35, 36, 36, 37, 37, 40,
41, 43, 46, 46, 47, 48, 49, 51, 51, 53, 54, 55, 57, 57, 60, 61, 63, 66,
66, 70, 39, 38, 38, 37, 37, 37, 38, 38, 39, 40, 40, 43, 44, 46, 50, 50,
52, 53, 54, 57, 57, 59, 60, 61, 64, 64, 67, 68, 69, 72, 72, 76, 44, 42,
42, 41, 41, 41, 42, 42, 42, 42, 42, 46, 48, 50, 54, 54, 57, 58, 60, 63,
63, 66, 67, 68, 71, 71, 74, 75, 77, 79, 79, 83, 53, 52, 51, 50, 49, 49,
49, 50, 49, 49, 49, 53, 54, 56, 60, 60, 64, 65, 67, 71, 71, 75, 76, 78,
82, 82, 86, 87, 89, 92, 92, 96, 65, 63, 62, 61, 59, 59, 59, 59, 58, 58,
58, 62, 63, 65, 68, 68, 72, 73, 76, 79, 79, 84, 85, 88, 92, 92, 97, 98,
100, 105, 105, 109 },
{ /* Chroma */
  /* Size 4x4 */
  31, 41, 46, 51, 41, 48, 48, 51, 46, 48, 58, 62, 51, 51, 62, 71,
  /* Size 8x8 */
  31, 31, 38, 44, 47, 48, 50, 55, 31, 32, 40, 44, 45, 46, 47, 52, 38, 40,
  47, 47, 46, 46, 47, 50, 44, 44, 47, 50, 51, 51, 52, 54, 47, 45, 46, 51,
  54, 56, 57, 60, 48, 46, 46, 51, 56, 61, 63, 66, 50, 47, 47, 52, 57, 63,
  66, 70, 55, 52, 50, 54, 60, 66, 70, 76,
  /* Size 16x16 */
  32, 31, 30, 33, 34, 36, 41, 49, 48, 49, 49, 50, 52, 54, 55, 57, 31, 31,
  31, 34, 36, 38, 42, 47, 47, 47, 47, 48, 50, 51, 53, 54, 30, 31, 32, 34,
  37, 40, 42, 46, 45, 45, 45, 46, 47, 49, 50, 52, 33, 34, 34, 37, 40, 42,
  44, 47, 46, 46, 45, 46, 47, 49, 50, 51, 34, 36, 37, 40, 42, 45, 46, 47,
  46, 46, 45, 46, 47, 48, 49, 50, 36, 38, 40, 42, 45, 47, 47, 48, 47, 46,
  45, 46, 47, 48, 49, 50, 41, 42, 42, 44, 46, 47, 48, 50, 50, 49, 49, 50,
  50, 51, 52, 53, 49, 47, 46, 47, 47, 48, 50, 53, 53, 53, 53, 54, 54, 55,
  56, 56, 48, 47, 45, 46, 46, 47, 50, 53, 54, 54, 55, 56, 57, 58, 58, 59,
  49, 47, 45, 46, 46, 46, 49, 53, 54, 55, 57, 58, 59, 60, 60, 61, 49, 47,
  45, 45, 45, 45, 49, 53, 55, 57, 58, 60, 61, 62, 63, 63, 50, 48, 46, 46,
  46, 46, 50, 54, 56, 58, 60, 61, 63, 65, 66, 67, 52, 50, 47, 47, 47, 47,
  50, 54, 57, 59, 61, 63, 66, 68, 69, 70, 54, 51, 49, 49, 48, 48, 51, 55,
  58, 60, 62, 65, 68, 70, 71, 73, 55, 53, 50, 50, 49, 49, 52, 56, 58, 60,
  63, 66, 69, 71, 73, 74, 57, 54, 52, 51, 50, 50, 53, 56, 59, 61, 63, 67,
  70, 73, 74, 76,
  /* Size 32x32 */
  32, 31, 31, 31, 30, 30, 33, 33, 34, 36, 36, 40, 41, 44, 49, 49, 48, 48,
  49, 49, 49, 50, 50, 51, 52, 52, 54, 54, 55, 57, 57, 59, 31, 31, 31, 31,
  31, 31, 33, 34, 36, 38, 38, 41, 42, 44, 48, 48, 47, 47, 47, 47, 47, 48,
  49, 49, 50, 50, 52, 52, 53, 55, 55, 57, 31, 31, 31, 31, 31, 31, 34, 34,
  36, 38, 38, 41, 42, 44, 47, 47, 47, 47, 47, 47, 47, 48, 48, 49, 50, 50,
  51, 52, 53, 54, 54, 56, 31, 31, 31, 31, 31, 31, 34, 35, 36, 39, 39, 41,
  42, 44, 47, 47, 46, 46, 46, 46, 46, 47, 47, 48, 49, 49, 50, 51, 52, 53,
  53, 55, 30, 31, 31, 31, 32, 32, 34, 35, 37, 40, 40, 42, 42, 44, 46, 46,
  45, 45, 45, 45, 45, 46, 46, 47, 47, 47, 49, 49, 50, 52, 52, 54, 30, 31,
  31, 31, 32, 32, 34, 35, 37, 40, 40, 42, 42, 44, 46, 46, 45, 45, 45, 45,
  45, 46, 46, 47, 47, 49, 49, 50, 52, 52, 54, 33, 33, 34, 34, 34, 34,
  37, 38, 40, 42, 42, 44, 44, 45, 47, 47, 46, 46, 46, 45, 45, 46, 46, 47,

```



```

47, 47, 49, 49, 50, 51, 51, 53, 33, 34, 34, 35, 35, 35, 38, 39, 40, 43,
43, 44, 45, 46, 47, 47, 46, 46, 46, 45, 45, 46, 46, 47, 47, 47, 49, 49,
50, 51, 51, 53, 34, 36, 36, 36, 37, 37, 40, 40, 42, 45, 45, 45, 46, 46,
47, 47, 46, 46, 46, 45, 45, 46, 46, 47, 47, 47, 48, 49, 49, 50, 50, 52,
36, 38, 38, 39, 40, 40, 42, 43, 45, 47, 47, 47, 47, 47, 48, 48, 47, 46,
46, 45, 45, 46, 46, 46, 47, 47, 48, 48, 49, 50, 50, 51, 36, 38, 38, 39,
40, 40, 42, 43, 45, 47, 47, 47, 47, 47, 48, 48, 47, 46, 46, 45, 45, 46,
46, 46, 47, 47, 48, 48, 49, 50, 50, 51, 40, 41, 41, 41, 42, 42, 44, 44,
45, 47, 47, 48, 48, 49, 50, 50, 49, 49, 49, 48, 48, 49, 49, 49, 49, 49,
51, 51, 51, 52, 52, 54, 41, 42, 42, 42, 42, 42, 44, 45, 46, 47, 47, 48,
48, 49, 50, 50, 50, 49, 49, 49, 49, 50, 50, 50, 50, 50, 51, 52, 52, 53,
53, 55, 44, 44, 44, 44, 44, 44, 45, 46, 46, 47, 47, 49, 49, 50, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 52, 52, 53, 53, 54, 54, 54, 56, 49, 48,
47, 47, 46, 46, 47, 47, 47, 48, 48, 50, 50, 51, 53, 53, 53, 53, 53, 53,
53, 54, 54, 54, 54, 54, 55, 55, 56, 56, 56, 58, 49, 48, 47, 47, 46, 46,
47, 47, 47, 48, 48, 50, 50, 51, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54,
54, 54, 55, 55, 56, 56, 56, 58, 48, 47, 47, 46, 45, 45, 46, 46, 46, 47,
47, 49, 50, 51, 53, 53, 54, 54, 54, 55, 55, 56, 56, 56, 57, 57, 58, 58,
58, 59, 59, 60, 48, 47, 47, 46, 45, 45, 46, 46, 46, 46, 46, 49, 49, 51,
53, 53, 54, 54, 55, 55, 55, 56, 56, 57, 57, 57, 58, 58, 59, 60, 60, 61,
49, 47, 47, 46, 45, 45, 46, 46, 46, 46, 46, 49, 49, 51, 53, 53, 54, 55,
55, 57, 57, 57, 58, 58, 59, 59, 60, 60, 60, 61, 61, 63, 49, 47, 47, 46,
45, 45, 45, 45, 45, 45, 45, 48, 49, 51, 53, 53, 55, 55, 57, 58, 58, 59,
60, 60, 61, 61, 62, 62, 63, 63, 63, 65, 49, 47, 47, 46, 45, 45, 45, 45,
45, 45, 45, 48, 49, 51, 53, 53, 55, 55, 57, 58, 58, 59, 60, 60, 61, 61,
62, 62, 63, 63, 63, 65, 50, 48, 48, 47, 46, 46, 46, 46, 46, 46, 46, 49,
50, 51, 54, 54, 56, 56, 57, 59, 59, 61, 61, 62, 63, 63, 64, 64, 65, 66,
66, 67, 50, 49, 48, 47, 46, 46, 46, 46, 46, 46, 46, 49, 50, 51, 54, 54,
56, 56, 58, 60, 60, 61, 61, 62, 63, 63, 65, 65, 66, 67, 67, 68, 51, 49,
49, 48, 47, 47, 47, 47, 47, 46, 46, 49, 50, 51, 54, 54, 56, 57, 58, 60,
60, 62, 62, 63, 65, 65, 66, 66, 67, 68, 68, 70, 52, 50, 50, 49, 47, 47,
47, 47, 47, 47, 47, 49, 50, 52, 54, 54, 57, 57, 59, 61, 61, 63, 63, 65,
66, 66, 68, 68, 69, 70, 70, 72, 52, 50, 50, 49, 47, 47, 47, 47, 47, 47,
47, 49, 50, 52, 54, 54, 57, 57, 59, 61, 61, 63, 63, 65, 66, 66, 68, 68,
69, 70, 70, 72, 54, 52, 51, 50, 49, 49, 49, 49, 48, 48, 48, 51, 51, 53,
55, 55, 58, 58, 60, 62, 62, 64, 65, 66, 68, 68, 70, 70, 71, 73, 73, 74,
54, 52, 52, 51, 49, 49, 49, 49, 49, 48, 48, 51, 52, 53, 55, 55, 58, 58,
60, 62, 62, 64, 65, 66, 68, 68, 70, 71, 72, 73, 73, 75, 55, 53, 53, 52,
50, 50, 50, 50, 49, 49, 49, 51, 52, 54, 56, 56, 58, 59, 60, 63, 63, 65,
66, 67, 69, 69, 71, 72, 73, 74, 74, 76, 57, 55, 54, 53, 52, 52, 51, 51,
50, 50, 50, 52, 53, 54, 56, 56, 59, 60, 61, 63, 63, 66, 67, 68, 70, 70,
73, 73, 74, 76, 76, 78, 57, 55, 54, 53, 52, 52, 51, 51, 50, 50, 50, 52,
53, 54, 56, 56, 59, 60, 61, 63, 63, 66, 67, 68, 70, 70, 73, 73, 74, 76,
76, 78, 59, 57, 56, 55, 54, 54, 53, 53, 52, 51, 51, 54, 55, 56, 58, 58,
60, 61, 63, 65, 65, 67, 68, 70, 72, 72, 74, 75, 76, 78, 78, 80,
/* Size 4x8 */
31, 38, 47, 52, 32, 40, 45, 49, 39, 47, 45, 48, 44, 47, 51, 53, 46, 47,
56, 58, 47, 46, 59, 64, 48, 47, 61, 68, 53, 50, 64, 73,
/* Size 8x4 */
31, 32, 39, 44, 46, 47, 48, 53, 38, 40, 47, 47, 47, 46, 47, 50, 47, 45,

```

```

45, 51, 56, 59, 61, 64, 52, 49, 48, 53, 58, 64, 68, 73,
/* Size 8x16 */
32, 31, 37, 45, 48, 49, 52, 57, 31, 31, 38, 45, 47, 47, 50, 54, 30, 32,
40, 44, 45, 45, 48, 52, 33, 35, 42, 46, 46, 45, 47, 51, 35, 37, 44, 46,
46, 45, 47, 51, 37, 40, 47, 47, 47, 45, 47, 50, 42, 43, 47, 49, 50, 49,
50, 53, 49, 46, 48, 52, 53, 53, 54, 57, 48, 46, 47, 51, 54, 55, 57, 59,
48, 45, 46, 51, 54, 57, 59, 61, 49, 45, 46, 51, 55, 58, 61, 64, 50, 46,
46, 52, 56, 59, 64, 67, 52, 48, 47, 53, 57, 61, 66, 71, 54, 49, 48, 54,
58, 62, 68, 73, 55, 51, 49, 54, 58, 63, 69, 74, 57, 52, 50, 55, 59, 64,
70, 76,
/* Size 16x8 */
32, 31, 30, 33, 35, 37, 42, 49, 48, 48, 49, 50, 52, 54, 55, 57, 31, 31,
32, 35, 37, 40, 43, 46, 46, 45, 45, 46, 48, 49, 51, 52, 37, 38, 40, 42,
44, 47, 47, 48, 47, 46, 46, 46, 47, 48, 49, 50, 45, 45, 44, 46, 46, 47,
49, 52, 51, 51, 51, 52, 53, 54, 54, 55, 48, 47, 45, 46, 46, 47, 50, 53,
54, 54, 55, 56, 57, 58, 58, 59, 49, 47, 45, 45, 45, 45, 49, 53, 55, 57,
58, 59, 61, 62, 63, 64, 52, 50, 48, 47, 47, 47, 50, 54, 57, 59, 61, 64,
66, 68, 69, 70, 57, 54, 52, 51, 51, 50, 53, 57, 59, 61, 64, 67, 71, 73,
74, 76,
/* Size 16x32 */
32, 31, 31, 33, 37, 37, 45, 48, 48, 49, 49, 51, 52, 54, 57, 57, 31, 31,
31, 34, 38, 38, 45, 47, 47, 47, 47, 50, 50, 52, 55, 55, 31, 31, 31, 34,
38, 38, 45, 47, 47, 47, 47, 49, 50, 51, 54, 54, 31, 31, 32, 34, 39, 39,
45, 46, 46, 46, 46, 48, 49, 51, 53, 53, 30, 32, 32, 35, 40, 40, 44, 46,
45, 45, 45, 47, 48, 49, 52, 52, 30, 32, 32, 35, 40, 40, 44, 46, 45, 45,
45, 47, 48, 49, 52, 52, 33, 34, 35, 37, 42, 42, 46, 47, 46, 45, 45, 47,
47, 49, 51, 51, 33, 35, 36, 38, 43, 43, 46, 47, 46, 46, 46, 47, 47, 49,
51, 51, 35, 37, 37, 40, 44, 44, 46, 47, 46, 45, 45, 47, 47, 48, 51, 51,
37, 39, 40, 43, 47, 47, 47, 47, 47, 45, 45, 46, 47, 48, 50, 50, 37, 39,
40, 43, 47, 47, 47, 47, 47, 45, 45, 46, 47, 48, 50, 50, 41, 42, 42, 44,
47, 47, 49, 49, 49, 48, 48, 49, 50, 51, 52, 52, 42, 42, 43, 44, 47, 47,
49, 50, 50, 49, 49, 50, 50, 51, 53, 53, 44, 44, 44, 45, 47, 47, 50, 51,
51, 51, 51, 52, 52, 53, 54, 54, 49, 47, 46, 47, 48, 48, 52, 53, 53, 53, 54,
53, 54, 54, 55, 57, 57, 49, 47, 46, 47, 48, 48, 52, 53, 53, 53, 54,
54, 55, 57, 57, 48, 46, 46, 46, 47, 47, 51, 53, 54, 55, 55, 56, 57, 58,
59, 59, 48, 46, 46, 46, 47, 47, 51, 53, 54, 56, 56, 57, 57, 58, 60, 60,
48, 46, 45, 46, 46, 46, 51, 53, 54, 57, 57, 58, 59, 60, 61, 61, 49, 46,
45, 45, 46, 46, 51, 53, 55, 58, 58, 61, 61, 62, 64, 64, 49, 46, 45, 45,
46, 46, 51, 53, 55, 58, 58, 61, 61, 62, 64, 64, 50, 47, 46, 46, 46, 46,
52, 54, 56, 59, 59, 62, 63, 64, 66, 66, 50, 47, 46, 46, 46, 46, 52, 54,
56, 59, 59, 63, 64, 65, 67, 67, 51, 48, 47, 47, 47, 47, 52, 54, 56, 60,
60, 64, 65, 66, 68, 68, 52, 48, 48, 47, 47, 47, 53, 54, 57, 61, 61, 65,
66, 68, 71, 71, 52, 48, 48, 47, 47, 47, 53, 54, 57, 61, 61, 65, 66, 68,
71, 71, 54, 50, 49, 49, 48, 48, 54, 55, 58, 62, 62, 67, 68, 70, 73, 73,
54, 51, 50, 49, 49, 49, 54, 55, 58, 62, 62, 67, 68, 70, 73, 73, 55, 51,
51, 50, 49, 49, 54, 56, 58, 63, 63, 68, 69, 71, 74, 74, 57, 53, 52, 51,
50, 50, 55, 56, 59, 64, 64, 69, 70, 73, 76, 76, 57, 53, 52, 51, 50, 50,
55, 56, 59, 64, 64, 69, 70, 73, 76, 76, 59, 55, 54, 53, 52, 52, 57, 58,
61, 65, 65, 70, 72, 74, 78, 78,
/* Size 32x16 */

```

```

32, 31, 31, 31, 30, 30, 33, 33, 35, 37, 37, 41, 42, 44, 49, 49, 48, 48,
48, 49, 49, 50, 50, 51, 52, 52, 54, 54, 55, 57, 57, 59, 31, 31, 31, 31,
32, 32, 34, 35, 37, 39, 39, 42, 42, 44, 47, 47, 46, 46, 46, 46, 46, 47,
47, 48, 48, 48, 50, 51, 51, 53, 53, 55, 31, 31, 31, 32, 32, 32, 35, 36,
37, 40, 40, 42, 43, 44, 46, 46, 46, 46, 45, 45, 45, 46, 46, 47, 48, 48,
49, 50, 51, 52, 52, 54, 33, 34, 34, 34, 35, 35, 37, 38, 40, 43, 43, 44,
44, 45, 47, 47, 46, 46, 46, 45, 45, 46, 46, 47, 47, 47, 49, 49, 50, 51,
51, 53, 37, 38, 38, 39, 40, 40, 42, 43, 44, 47, 47, 47, 47, 47, 47, 48, 48,
47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 48, 49, 49, 50, 50, 52, 37, 38,
38, 39, 40, 40, 42, 43, 44, 47, 47, 47, 47, 47, 48, 48, 47, 47, 46, 46,
46, 46, 46, 47, 47, 47, 48, 49, 49, 50, 50, 52, 45, 45, 45, 45, 44, 44,
46, 46, 46, 47, 47, 49, 49, 50, 52, 52, 51, 51, 51, 51, 51, 52, 52, 52,
53, 53, 54, 54, 54, 55, 55, 57, 48, 47, 47, 46, 46, 46, 47, 47, 47, 47,
47, 49, 50, 51, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 55, 55,
56, 56, 56, 58, 48, 47, 47, 46, 45, 45, 46, 46, 46, 47, 47, 49, 50, 51,
53, 53, 54, 54, 54, 55, 55, 56, 56, 56, 57, 57, 58, 58, 58, 59, 59, 61,
49, 47, 47, 46, 45, 45, 45, 46, 45, 45, 45, 48, 49, 51, 53, 53, 55, 56,
57, 58, 58, 59, 59, 60, 61, 61, 62, 62, 63, 64, 64, 65, 49, 47, 47, 46,
45, 45, 45, 46, 45, 45, 45, 48, 49, 51, 53, 53, 55, 56, 57, 58, 58, 59,
59, 60, 61, 61, 62, 62, 63, 64, 64, 65, 51, 50, 49, 48, 47, 47, 47, 47,
47, 46, 46, 49, 50, 52, 54, 54, 56, 57, 58, 61, 61, 62, 63, 64, 65, 65,
67, 67, 68, 69, 69, 70, 52, 50, 50, 49, 48, 48, 47, 47, 47, 47, 47, 50,
50, 52, 54, 54, 57, 57, 59, 61, 61, 63, 64, 65, 66, 66, 68, 68, 69, 70,
70, 72, 54, 52, 51, 51, 49, 49, 49, 49, 48, 48, 48, 51, 51, 53, 55, 55,
58, 58, 60, 62, 62, 64, 65, 66, 68, 68, 70, 70, 71, 73, 73, 74, 57, 55,
54, 53, 52, 52, 51, 51, 51, 50, 50, 52, 53, 54, 57, 57, 59, 60, 61, 64,
64, 66, 67, 68, 71, 71, 73, 73, 74, 76, 76, 78, 57, 55, 54, 53, 52, 52,
51, 51, 51, 50, 50, 52, 53, 54, 57, 57, 59, 60, 61, 64, 64, 66, 67, 68,
71, 71, 73, 73, 74, 76, 76, 78,
/* Size 4x16 */
31, 37, 49, 54, 31, 38, 47, 51, 32, 40, 45, 49, 34, 42, 45, 49, 37, 44,
45, 48, 39, 47, 45, 48, 42, 47, 49, 51, 47, 48, 53, 55, 46, 47, 55, 58,
46, 46, 57, 60, 46, 46, 58, 62, 47, 46, 59, 65, 48, 47, 61, 68, 50, 48,
62, 70, 51, 49, 63, 71, 53, 50, 64, 73,
/* Size 16x4 */
31, 31, 32, 34, 37, 39, 42, 47, 46, 46, 46, 47, 48, 50, 51, 53, 37, 38,
40, 42, 44, 47, 47, 48, 47, 46, 46, 46, 47, 48, 49, 50, 49, 47, 45, 45,
45, 45, 49, 53, 55, 57, 58, 59, 61, 62, 63, 64, 54, 51, 49, 49, 48, 48,
51, 55, 58, 60, 62, 65, 68, 70, 71, 73,
/* Size 8x32 */
32, 31, 37, 45, 48, 49, 52, 57, 31, 31, 38, 45, 47, 47, 50, 55, 31, 31,
38, 45, 47, 47, 50, 54, 31, 32, 39, 45, 46, 46, 49, 53, 30, 32, 40, 44,
45, 45, 48, 52, 30, 32, 40, 44, 45, 45, 48, 52, 33, 35, 42, 46, 46, 45,
47, 51, 33, 36, 43, 46, 46, 46, 47, 51, 35, 37, 44, 46, 46, 45, 47, 51,
37, 40, 47, 47, 47, 45, 47, 50, 37, 40, 47, 47, 47, 45, 47, 50, 41, 42,
47, 49, 49, 48, 50, 52, 42, 43, 47, 49, 50, 49, 50, 53, 44, 44, 47, 50,
51, 51, 52, 54, 49, 46, 48, 52, 53, 53, 54, 57, 49, 46, 48, 52, 53, 53,
54, 57, 48, 46, 47, 51, 54, 55, 57, 59, 48, 46, 47, 51, 54, 56, 57, 60,
48, 45, 46, 51, 54, 57, 59, 61, 49, 45, 46, 51, 55, 58, 61, 64, 49, 45,
46, 51, 55, 58, 61, 64, 50, 46, 46, 52, 56, 59, 63, 66, 50, 46, 46, 52,

```

```

56, 59, 64, 67, 51, 47, 47, 52, 56, 60, 65, 68, 52, 48, 47, 53, 57, 61,
66, 71, 52, 48, 47, 53, 57, 61, 66, 71, 54, 49, 48, 54, 58, 62, 68, 73,
54, 50, 49, 54, 58, 62, 68, 73, 55, 51, 49, 54, 58, 63, 69, 74, 57, 52,
50, 55, 59, 64, 70, 76, 57, 52, 50, 55, 59, 64, 70, 76, 59, 54, 52, 57,
61, 65, 72, 78,
/* Size 32x8 */
32, 31, 31, 31, 30, 30, 33, 33, 35, 37, 37, 41, 42, 44, 49, 49, 48, 48,
48, 49, 49, 50, 50, 51, 52, 52, 54, 54, 55, 57, 57, 59, 31, 31, 31, 32,
32, 32, 35, 36, 37, 40, 40, 42, 43, 44, 46, 46, 46, 46, 45, 45, 45, 46,
46, 47, 48, 48, 49, 50, 51, 52, 52, 54, 37, 38, 38, 39, 40, 40, 42, 43,
44, 47, 47, 47, 47, 47, 48, 48, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47,
48, 49, 49, 50, 50, 52, 45, 45, 45, 45, 44, 44, 46, 46, 46, 47, 47, 49,
49, 50, 52, 52, 51, 51, 51, 51, 51, 52, 52, 52, 53, 53, 54, 54, 54, 55,
55, 57, 48, 47, 47, 46, 45, 45, 46, 46, 46, 47, 47, 49, 50, 51, 53, 53,
54, 54, 54, 55, 55, 56, 56, 56, 57, 57, 58, 58, 58, 59, 59, 61, 49, 47,
47, 46, 45, 45, 45, 46, 45, 45, 45, 48, 49, 51, 53, 53, 55, 56, 57, 58,
58, 59, 59, 60, 61, 61, 62, 62, 63, 64, 64, 65, 52, 50, 50, 49, 48, 48,
47, 47, 47, 47, 47, 50, 50, 52, 54, 54, 57, 57, 59, 61, 61, 63, 64, 65,
66, 66, 68, 68, 69, 70, 70, 72, 57, 55, 54, 53, 52, 52, 51, 51, 51, 50,
50, 52, 53, 54, 57, 57, 59, 60, 61, 64, 64, 66, 67, 68, 71, 71, 73, 73,
74, 76, 76, 78 },
},
{
/* Luma */
/* Size 4x4 */
32, 32, 38, 51, 32, 35, 40, 49, 38, 40, 54, 64, 51, 49, 64, 81,
/* Size 8x8 */
31, 32, 32, 34, 35, 41, 47, 53, 32, 32, 32, 33, 34, 40, 44, 50, 32, 32,
34, 35, 37, 41, 45, 51, 34, 33, 35, 39, 42, 47, 51, 55, 35, 34, 37, 42,
48, 53, 57, 61, 41, 40, 41, 47, 53, 60, 65, 70, 47, 44, 45, 51, 57, 65,
71, 77, 53, 50, 51, 55, 61, 70, 77, 85,
/* Size 16x16 */
32, 31, 31, 31, 31, 32, 32, 34, 36, 38, 39, 44, 47, 49, 54, 59, 31, 32,
32, 32, 32, 32, 33, 34, 35, 37, 38, 42, 45, 47, 51, 56, 31, 32, 32, 32,
32, 32, 33, 33, 34, 36, 37, 41, 44, 46, 50, 54, 31, 32, 32, 32, 32, 33,
33, 34, 35, 36, 38, 41, 44, 45, 49, 54, 31, 32, 32, 32, 33, 34, 34, 35,
36, 38, 39, 42, 45, 46, 50, 54, 32, 32, 32, 33, 34, 35, 36, 37, 38, 39,
40, 42, 45, 46, 49, 53, 32, 33, 33, 33, 34, 36, 36, 38, 40, 41, 42, 44,
47, 48, 51, 55, 34, 34, 33, 34, 35, 37, 38, 39, 42, 44, 45, 47, 50, 51,
54, 58, 36, 35, 34, 35, 36, 38, 40, 42, 48, 50, 50, 54, 56, 57, 60, 64,
38, 37, 36, 36, 38, 39, 41, 44, 50, 51, 52, 56, 58, 60, 63, 67, 39, 38,
37, 38, 39, 40, 42, 45, 50, 52, 54, 58, 60, 62, 65, 69, 44, 42, 41, 41,
42, 42, 44, 47, 54, 56, 58, 63, 66, 68, 71, 75, 47, 45, 44, 44, 45, 45,
47, 50, 56, 58, 60, 66, 69, 71, 75, 79, 49, 47, 46, 45, 46, 46, 48, 51,
57, 60, 62, 68, 71, 73, 77, 81, 54, 51, 50, 49, 50, 49, 51, 54, 60, 63,
65, 71, 75, 77, 82, 87, 59, 56, 54, 54, 54, 53, 55, 58, 64, 67, 69, 75,
79, 81, 87, 92,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 35, 36, 36,
38, 39, 39, 42, 44, 44, 47, 48, 49, 53, 54, 55, 59, 59, 31, 31, 31, 31,

```

32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 37, 39, 39, 41,  
 43, 43, 46, 47, 48, 51, 52, 53, 57, 57, 31, 31, 32, 32, 32, 32, 32, 32,  
 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 37, 38, 38, 41, 42, 43, 45, 46,  
 47, 51, 51, 53, 56, 56, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
 33, 34, 34, 34, 35, 35, 37, 38, 38, 41, 42, 42, 45, 46, 47, 51, 51, 52,  
 56, 56, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34,  
 34, 34, 36, 37, 37, 40, 41, 41, 44, 45, 46, 49, 50, 51, 54, 54, 31, 32,  
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 36, 37,  
 37, 40, 41, 41, 44, 44, 45, 49, 49, 50, 54, 54, 31, 32, 32, 32, 32, 32,  
 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 35, 36, 38, 38, 40, 41, 41,  
 44, 45, 45, 49, 49, 50, 54, 54, 31, 32, 32, 32, 32, 32, 32, 33, 33, 33,  
 34, 34, 34, 35, 35, 35, 36, 36, 38, 39, 39, 41, 42, 42, 44, 45, 46, 49,  
 50, 51, 54, 54, 31, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 35,  
 35, 36, 36, 36, 38, 39, 39, 41, 42, 42, 45, 45, 46, 49, 50, 51, 54, 54,  
 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 37, 37,  
 38, 39, 39, 41, 42, 42, 45, 45, 46, 49, 49, 51, 54, 54, 32, 32, 32, 32,  
 32, 32, 33, 34, 34, 34, 35, 35, 36, 37, 37, 37, 38, 38, 39, 40, 40, 42,  
 42, 43, 45, 46, 46, 49, 49, 50, 53, 53, 32, 32, 32, 32, 32, 32, 33, 34,  
 34, 34, 35, 35, 36, 37, 37, 37, 38, 38, 39, 40, 40, 42, 42, 43, 45, 46,  
 46, 49, 49, 50, 53, 53, 32, 33, 33, 33, 33, 33, 33, 34, 34, 35, 36, 36,  
 36, 38, 38, 39, 40, 40, 41, 42, 42, 44, 44, 45, 47, 47, 48, 51, 51, 52,  
 55, 55, 34, 34, 34, 34, 33, 33, 34, 35, 35, 35, 37, 37, 38, 39, 39, 41,  
 42, 42, 44, 45, 45, 47, 47, 48, 50, 51, 51, 54, 54, 55, 58, 58, 34, 34,  
 34, 34, 33, 33, 34, 35, 35, 35, 37, 37, 38, 39, 39, 41, 42, 42, 44, 45,  
 45, 47, 47, 48, 50, 51, 51, 54, 54, 55, 58, 58, 35, 34, 34, 34, 34, 34,  
 34, 35, 36, 36, 37, 37, 39, 41, 41, 43, 45, 45, 47, 47, 47, 49, 50, 51,  
 53, 53, 54, 57, 57, 58, 61, 61, 36, 35, 35, 35, 34, 34, 35, 36, 36, 37,  
 38, 38, 40, 42, 42, 45, 48, 48, 50, 50, 50, 53, 54, 54, 56, 57, 57, 59,  
 60, 61, 64, 64, 36, 35, 35, 35, 34, 34, 35, 36, 36, 37, 38, 38, 40, 42,  
 42, 45, 48, 48, 50, 50, 50, 53, 54, 54, 56, 57, 57, 59, 60, 61, 64, 64,  
 38, 37, 37, 37, 36, 36, 36, 38, 38, 38, 39, 39, 41, 44, 44, 47, 50, 50,  
 51, 52, 52, 55, 56, 56, 58, 59, 60, 62, 63, 64, 67, 67, 39, 39, 38, 38,  
 37, 37, 38, 39, 39, 39, 40, 40, 42, 45, 45, 47, 50, 50, 52, 54, 54, 56,  
 58, 58, 60, 61, 62, 64, 65, 66, 69, 69, 39, 39, 38, 38, 37, 37, 38, 39,  
 39, 39, 40, 40, 42, 45, 45, 47, 50, 50, 52, 54, 54, 56, 58, 58, 60, 61,  
 62, 64, 65, 66, 69, 69, 42, 41, 41, 41, 40, 40, 40, 41, 41, 41, 42, 42,  
 44, 47, 47, 49, 53, 53, 55, 56, 56, 60, 61, 62, 64, 65, 66, 69, 69, 70,  
 73, 73, 44, 43, 42, 42, 41, 41, 41, 42, 42, 42, 42, 42, 44, 47, 47, 50,  
 54, 54, 56, 58, 58, 61, 63, 64, 66, 67, 68, 71, 71, 72, 75, 75, 44, 43,  
 43, 42, 41, 41, 41, 42, 42, 42, 43, 43, 45, 48, 48, 51, 54, 54, 56, 58,  
 58, 62, 64, 64, 66, 67, 68, 71, 72, 73, 76, 76, 47, 46, 45, 45, 44, 44,  
 44, 44, 45, 45, 45, 45, 47, 50, 50, 53, 56, 56, 58, 60, 60, 64, 66, 66,  
 69, 70, 71, 74, 75, 76, 79, 79, 48, 47, 46, 46, 45, 44, 45, 45, 45, 45,  
 46, 46, 47, 51, 51, 53, 57, 57, 59, 61, 61, 65, 67, 67, 70, 71, 72, 75,  
 76, 77, 80, 80, 49, 48, 47, 47, 46, 45, 45, 46, 46, 46, 46, 46, 48, 51,  
 51, 54, 57, 57, 60, 62, 62, 66, 68, 68, 71, 72, 73, 77, 77, 78, 81, 81,  
 53, 51, 51, 51, 49, 49, 49, 49, 49, 49, 49, 49, 51, 54, 54, 57, 59, 59,  
 62, 64, 64, 69, 71, 71, 74, 75, 77, 81, 81, 83, 86, 86, 54, 52, 51, 51,  
 50, 49, 49, 50, 50, 49, 49, 49, 51, 54, 54, 57, 60, 60, 63, 65, 65, 69,  
 71, 72, 75, 76, 77, 81, 82, 83, 87, 87, 55, 53, 53, 52, 51, 50, 50, 51,

```

51, 51, 50, 50, 52, 55, 55, 58, 61, 61, 64, 66, 66, 70, 72, 73, 76, 77,
78, 83, 83, 85, 88, 88, 59, 57, 56, 56, 54, 54, 54, 54, 54, 54, 53, 53,
55, 58, 58, 61, 64, 64, 67, 69, 69, 73, 75, 76, 79, 80, 81, 86, 87, 88,
92, 92, 59, 57, 56, 56, 54, 54, 54, 54, 54, 54, 53, 53, 55, 58, 58, 61,
64, 64, 67, 69, 69, 73, 75, 76, 79, 80, 81, 86, 87, 88, 92, 92,
/* Size 4x8 */
32, 32, 37, 52, 32, 33, 36, 49, 32, 34, 38, 49, 34, 37, 44, 54, 35, 38,
49, 60, 40, 42, 55, 69, 46, 46, 59, 76, 52, 51, 64, 83,
/* Size 8x4 */
32, 32, 32, 34, 35, 40, 46, 52, 32, 33, 34, 37, 38, 42, 46, 51, 37, 36,
38, 44, 49, 55, 59, 64, 52, 49, 49, 54, 60, 69, 76, 83,
/* Size 8x16 */
32, 31, 32, 32, 36, 44, 47, 53, 31, 32, 32, 33, 35, 42, 45, 51, 31, 32,
32, 33, 35, 41, 44, 49, 31, 32, 33, 33, 35, 41, 44, 49, 32, 32, 34, 34,
36, 42, 45, 50, 32, 33, 35, 36, 38, 42, 45, 49, 32, 33, 35, 36, 40, 44,
47, 51, 34, 34, 36, 38, 42, 48, 50, 54, 36, 34, 37, 40, 48, 54, 56, 60,
38, 36, 39, 41, 49, 56, 58, 63, 39, 37, 40, 42, 50, 58, 60, 65, 44, 41,
42, 45, 53, 63, 66, 71, 47, 44, 45, 47, 56, 66, 69, 75, 49, 46, 47, 48,
57, 67, 71, 77, 53, 49, 50, 51, 60, 71, 75, 82, 58, 54, 54, 55, 63, 75,
79, 87,
/* Size 16x8 */
32, 31, 31, 31, 32, 32, 32, 34, 36, 38, 39, 44, 47, 49, 53, 58, 31, 32,
32, 32, 32, 33, 33, 34, 34, 36, 37, 41, 44, 46, 49, 54, 32, 32, 32, 33,
34, 35, 35, 36, 37, 39, 40, 42, 45, 47, 50, 54, 32, 33, 33, 33, 34, 36,
36, 38, 40, 41, 42, 45, 47, 48, 51, 55, 36, 35, 35, 35, 36, 38, 40, 42,
48, 49, 50, 53, 56, 57, 60, 63, 44, 42, 41, 41, 42, 42, 44, 48, 54, 56,
58, 63, 66, 67, 71, 75, 47, 45, 44, 44, 45, 45, 47, 50, 56, 58, 60, 66,
69, 71, 75, 79, 53, 51, 49, 49, 50, 49, 51, 54, 60, 63, 65, 71, 75, 77,
82, 87,
/* Size 16x32 */
32, 31, 31, 31, 32, 32, 32, 35, 36, 38, 44, 44, 47, 53, 53, 59, 31, 32,
32, 32, 32, 32, 33, 35, 35, 37, 43, 43, 46, 52, 52, 57, 31, 32, 32, 32,
32, 32, 33, 35, 35, 37, 42, 42, 45, 51, 51, 56, 31, 32, 32, 32, 32, 32,
33, 35, 35, 37, 42, 42, 45, 51, 51, 56, 31, 32, 32, 32, 32, 32, 33, 34,
35, 36, 41, 41, 44, 49, 49, 54, 31, 32, 32, 32, 32, 33, 33, 34, 34, 36,
41, 41, 44, 49, 49, 54, 31, 32, 32, 32, 33, 33, 33, 35, 35, 36, 41, 41,
44, 49, 49, 54, 32, 32, 32, 32, 33, 34, 34, 36, 36, 38, 42, 42, 45, 49,
49, 54, 32, 32, 32, 33, 34, 34, 34, 36, 36, 38, 42, 42, 45, 50, 50, 54,
32, 32, 32, 33, 34, 34, 35, 37, 37, 38, 42, 42, 45, 49, 49, 54, 32, 32,
33, 33, 35, 35, 36, 38, 38, 39, 42, 42, 45, 49, 49, 53, 32, 33, 33, 33,
35, 36,
36, 39, 40, 41, 44, 44, 47, 51, 51, 55, 34, 34, 34, 34, 36, 37, 38, 42,
42, 44, 48, 48, 50, 54, 54, 58, 34, 34, 34, 34, 36, 37, 38, 42, 42, 44,
48, 48, 50, 54, 54, 58, 35, 34, 34, 34, 37, 37, 39, 44, 45, 46, 50, 50,
53, 57, 57, 61, 36, 35, 34, 35, 37, 38, 40, 47, 48, 49, 54, 54, 56, 60,
60, 64, 36, 35, 34, 35, 37, 38, 40, 47, 48, 49, 54, 54, 56, 60, 60, 64,
38, 37, 36, 37, 39, 40, 41, 48, 49, 51, 56, 56, 58, 63, 63, 67, 39, 38,
37, 38, 40, 40, 42, 49, 50, 52, 58, 58, 60, 65, 65, 69, 39, 38, 37, 38,
40, 40, 42, 49, 50, 52, 58, 58, 60, 65, 65, 69, 42, 40, 40, 40, 42, 42,
44, 51, 52, 55, 61, 61, 64, 69, 69, 73, 44, 42, 41, 41, 42, 43, 45, 52,

```

```

53, 56, 63, 63, 66, 71, 71, 75, 44, 42, 41, 41, 43, 43, 45, 52, 54, 56,
63, 63, 66, 72, 72, 76, 47, 45, 44, 44, 45, 45, 47, 54, 56, 58, 66, 66,
69, 75, 75, 79, 48, 46, 45, 45, 46, 46, 48, 55, 56, 59, 67, 67, 70, 76,
76, 80, 49, 47, 46, 46, 47, 47, 48, 56, 57, 60, 67, 67, 71, 77, 77, 81,
53, 50, 49, 49, 49, 49, 51, 58, 59, 62, 71, 71, 74, 81, 81, 86, 53, 51,
49, 49, 50, 50, 51, 59, 60, 63, 71, 71, 75, 82, 82, 87, 55, 52, 51, 51,
51, 51, 53, 60, 61, 64, 72, 72, 76, 83, 83, 88, 58, 55, 54, 54, 54, 54,
55, 62, 63, 67, 75, 75, 79, 87, 87, 92, 58, 55, 54, 54, 54, 54, 55, 62,
63, 67, 75, 75, 79, 87, 87, 92,

```

```
/* Size 32x16 */
```

```

32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 34, 35, 36, 36,
38, 39, 39, 42, 44, 44, 47, 48, 49, 53, 53, 55, 58, 58, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 37, 38, 38, 40,
42, 42, 45, 46, 47, 50, 51, 52, 55, 55, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 36, 37, 37, 40, 41, 41, 44, 45,
46, 49, 49, 51, 54, 54, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33,
33, 34, 34, 34, 35, 35, 37, 38, 38, 40, 41, 41, 44, 45, 46, 49, 49, 51,
54, 54, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 35, 35, 35, 36, 36, 37,
37, 37, 39, 40, 40, 42, 42, 43, 45, 46, 47, 49, 50, 51, 54, 54, 32, 32,
32, 32, 32, 33, 33, 34, 34, 34, 35, 35, 36, 37, 37, 37, 38, 38, 40, 40,
40, 42, 43, 43, 45, 46, 47, 49, 50, 51, 54, 54, 32, 33, 33, 33, 33, 33,
33, 34, 34, 35, 36, 36, 36, 38, 38, 39, 40, 40, 41, 42, 42, 44, 45, 45,
47, 48, 48, 51, 51, 53, 55, 55, 35, 35, 35, 35, 34, 34, 35, 36, 36, 37,
38, 38, 39, 42, 42, 44, 47, 47, 48, 49, 49, 51, 52, 52, 54, 55, 56, 58,
59, 60, 62, 62, 36, 35, 35, 35, 35, 34, 35, 36, 36, 37, 38, 38, 40, 42,
42, 45, 48, 48, 49, 50, 50, 52, 53, 54, 56, 56, 57, 59, 60, 61, 63, 63,
38, 37, 37, 37, 36, 36, 36, 38, 38, 38, 39, 39, 41, 44, 44, 46, 49, 49,
51, 52, 52, 55, 56, 56, 58, 59, 60, 62, 63, 64, 67, 67, 44, 43, 42, 42,
41, 41, 41, 42, 42, 42, 42, 42, 44, 48, 48, 50, 54, 54, 56, 58, 58, 61,
63, 63, 66, 67, 67, 71, 71, 72, 75, 75, 44, 43, 42, 42, 41, 41, 41, 42,
42, 42, 42, 42, 44, 48, 48, 50, 54, 54, 56, 58, 58, 61, 63, 63, 66, 67,
67, 71, 71, 72, 75, 75, 47, 46, 45, 45, 44, 44, 44, 45, 45, 45, 45, 45,
47, 50, 50, 53, 56, 56, 58, 60, 60, 64, 66, 66, 69, 70, 71, 74, 75, 76,
79, 79, 53, 52, 51, 51, 49, 49, 49, 49, 50, 49, 49, 49, 51, 54, 54, 57,
60, 60, 63, 65, 65, 69, 71, 72, 75, 76, 77, 81, 82, 83, 87, 87, 53, 52,
51, 51, 49, 49, 49, 49, 50, 49, 49, 49, 51, 54, 54, 57, 60, 60, 63, 65,
65, 69, 71, 72, 75, 76, 77, 81, 82, 83, 87, 87, 59, 57, 56, 56, 54, 54,
54, 54, 54, 54, 53, 53, 55, 58, 58, 61, 64, 64, 67, 69, 69, 73, 75, 76,
79, 80, 81, 86, 87, 88, 92, 92,

```

```
/* Size 4x16 */
```

```

31, 32, 38, 53, 32, 32, 37, 51, 32, 32, 36, 49, 32, 33, 36, 49, 32, 34,
38, 50, 32, 35, 39, 49, 33, 36, 41, 51, 34, 37, 44, 54, 35, 38, 49, 60,
37, 40, 51, 63, 38, 40, 52, 65, 42, 43, 56, 71, 45, 45, 58, 75, 47, 47,
60, 77, 51, 50, 63, 82, 55, 54, 67, 87,

```

```
/* Size 16x4 */
```

```

31, 32, 32, 32, 32, 32, 33, 34, 35, 37, 38, 42, 45, 47, 51, 55, 32, 32,
32, 33, 34, 35, 36, 37, 38, 40, 40, 43, 45, 47, 50, 54, 38, 37, 36, 36,
38, 39, 41, 44, 49, 51, 52, 56, 58, 60, 63, 67, 53, 51, 49, 49, 50, 49,
51, 54, 60, 63, 65, 71, 75, 77, 82, 87,

```

```
/* Size 8x32 */
```

```

32, 31, 32, 32, 36, 44, 47, 53, 31, 32, 32, 33, 35, 43, 46, 52, 31, 32,
32, 33, 35, 42, 45, 51, 31, 32, 32, 33, 35, 42, 45, 51, 31, 32, 32, 33,
35, 41, 44, 49, 31, 32, 32, 33, 34, 41, 44, 49, 31, 32, 33, 33, 35, 41,
44, 49, 32, 32, 33, 34, 36, 42, 45, 49, 32, 32, 34, 34, 36, 42, 45, 50,
32, 32, 34, 35, 37, 42, 45, 49, 32, 33, 35, 36, 38, 42, 45, 49, 32, 33,
35, 36, 38, 42, 45, 49, 32, 33, 35, 36, 40, 44, 47, 51, 34, 34, 36, 38,
42, 48, 50, 54, 34, 34, 36, 38, 42, 48, 50, 54, 35, 34, 37, 39, 45, 50,
53, 57, 36, 34, 37, 40, 48, 54, 56, 60, 36, 34, 37, 40, 48, 54, 56, 60,
38, 36, 39, 41, 49, 56, 58, 63, 39, 37, 40, 42, 50, 58, 60, 65, 39, 37,
40, 42, 50, 58, 60, 65, 42, 40, 42, 44, 52, 61, 64, 69, 44, 41, 42, 45,
53, 63, 66, 71, 44, 41, 43, 45, 54, 63, 66, 72, 47, 44, 45, 47, 56, 66,
69, 75, 48, 45, 46, 48, 56, 67, 70, 76, 49, 46, 47, 48, 57, 67, 71, 77,
53, 49, 49, 51, 59, 71, 74, 81, 53, 49, 50, 51, 60, 71, 75, 82, 55, 51,
51, 53, 61, 72, 76, 83, 58, 54, 54, 55, 63, 75, 79, 87, 58, 54, 54, 55,
63, 75, 79, 87,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 34, 35, 36, 36,
38, 39, 39, 42, 44, 44, 47, 48, 49, 53, 53, 55, 58, 58, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 36, 37, 37, 40,
41, 41, 44, 45, 46, 49, 49, 51, 54, 54, 32, 32, 32, 32, 32, 32, 33, 33,
34, 34, 35, 35, 35, 36, 36, 37, 37, 37, 39, 40, 40, 42, 42, 43, 45, 46,
47, 49, 50, 51, 54, 54, 32, 33, 33, 33, 33, 33, 33, 34, 34, 35, 36, 36,
36, 38, 38, 39, 40, 40, 41, 42, 42, 44, 45, 45, 47, 48, 48, 51, 51, 53,
55, 55, 36, 35, 35, 35, 35, 34, 35, 36, 36, 37, 38, 38, 40, 42, 42, 45,
48, 48, 49, 50, 50, 52, 53, 54, 56, 56, 57, 59, 60, 61, 63, 63, 44, 43,
42, 42, 41, 41, 41, 42, 42, 42, 42, 42, 44, 48, 48, 50, 54, 54, 56, 58,
58, 61, 63, 63, 66, 67, 67, 71, 71, 72, 75, 75, 47, 46, 45, 45, 44, 44,
44, 45, 45, 45, 45, 45, 47, 50, 50, 53, 56, 56, 58, 60, 60, 64, 66, 66,
69, 70, 71, 74, 75, 76, 79, 79, 53, 52, 51, 51, 49, 49, 49, 49, 50, 49,
49, 49, 51, 54, 54, 57, 60, 60, 63, 65, 65, 69, 71, 72, 75, 76, 77, 81,
82, 83, 87, 87 },
{ /* Chroma */
/* Size 4x4 */
31, 38, 47, 49, 38, 47, 46, 46, 47, 46, 54, 57, 49, 46, 57, 66,
/* Size 8x8 */
31, 31, 35, 42, 48, 47, 49, 51, 31, 32, 36, 42, 46, 45, 46, 48, 35, 36,
41, 45, 47, 45, 46, 48, 42, 42, 45, 48, 50, 49, 50, 51, 48, 46, 47, 50,
53, 53, 54, 54, 47, 45, 45, 49, 53, 57, 59, 60, 49, 46, 46, 50, 54, 59,
61, 64, 51, 48, 48, 51, 54, 60, 64, 68,
/* Size 16x16 */
32, 31, 30, 31, 33, 36, 38, 41, 49, 49, 48, 49, 50, 51, 52, 54, 31, 31,
31, 32, 34, 38, 40, 42, 47, 47, 47, 47, 48, 48, 50, 52, 30, 31, 31, 32,
35, 39, 41, 42, 46, 46, 46, 45, 46, 47, 48, 50, 31, 32, 32, 33, 36, 40,
41, 43, 46, 46, 45, 45, 46, 46, 47, 49, 33, 34, 35, 36, 39, 43, 44, 45,
47, 46, 46, 45, 46, 47, 47, 49, 36, 38, 39, 40, 43, 47, 47, 47, 48, 47,
46, 45, 46, 46, 47, 48, 38, 40, 41, 41, 44, 47, 47, 48, 49, 48, 48, 47,
47, 47, 48, 49, 41, 42, 42, 43, 45, 47, 48, 48, 50, 50, 49, 49, 50, 50,
50, 52, 49, 47, 46, 46, 47, 48, 49, 50, 53, 53, 53, 53, 54, 54, 54, 55,
49, 47, 46, 46, 46, 47, 48, 50, 53, 53, 54, 55, 55, 55, 56, 57, 48, 47,
46, 45, 46, 46, 48, 49, 53, 54, 54, 55, 56, 56, 57, 58, 49, 47, 45, 45,

```



```

45, 45, 47, 49, 53, 55, 55, 58, 59, 60, 61, 62, 50, 48, 46, 46, 46, 46,
47, 50, 54, 55, 56, 59, 61, 61, 63, 64, 51, 48, 47, 46, 47, 46, 47, 50,
54, 55, 56, 60, 61, 62, 64, 66, 52, 50, 48, 47, 47, 47, 48, 50, 54, 56,
57, 61, 63, 64, 66, 68, 54, 52, 50, 49, 49, 48, 49, 52, 55, 57, 58, 62,
64, 66, 68, 71,
/* Size 32x32 */
32, 31, 31, 31, 30, 30, 31, 33, 33, 34, 36, 36, 38, 41, 41, 45, 49, 49,
49, 48, 48, 49, 49, 49, 50, 50, 51, 52, 52, 53, 54, 54, 31, 31, 31, 31,
31, 31, 31, 34, 34, 35, 38, 38, 39, 42, 42, 45, 48, 48, 47, 47, 47, 47,
47, 47, 49, 49, 49, 50, 50, 51, 53, 53, 31, 31, 31, 31, 31, 31, 32, 34,
34, 35, 38, 38, 40, 42, 42, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48,
48, 49, 50, 50, 52, 52, 31, 31, 31, 31, 31, 31, 32, 34, 34, 36, 38, 38,
40, 42, 42, 45, 47, 47, 47, 47, 47, 47, 46, 47, 48, 48, 48, 49, 49, 50,
52, 52, 30, 31, 31, 31, 31, 31, 32, 35, 35, 36, 39, 39, 41, 42, 42, 44,
46, 46, 46, 46, 46, 45, 45, 45, 46, 47, 47, 48, 48, 48, 50, 50, 30, 31,
31, 31, 31, 32, 32, 35, 35, 36, 40, 40, 41, 42, 42, 44, 46, 46, 46, 45,
45, 45, 45, 45, 46, 46, 46, 47, 47, 48, 49, 49, 31, 31, 32, 32, 32, 32,
33, 35, 36, 37, 40, 40, 41, 43, 43, 44, 46, 46, 46, 45, 45, 45, 45, 45,
46, 46, 46, 47, 47, 48, 49, 49, 33, 34, 34, 34, 35, 35, 35, 38, 38, 40,
43, 43, 43, 44, 44, 46, 47, 47, 46, 46, 46, 45, 45, 45, 46, 46, 47, 47,
47, 48, 49, 49, 33, 34, 34, 34, 35, 35, 36, 38, 39, 40, 43, 43, 44, 45,
45, 46, 47, 47, 46, 46, 46, 45, 45, 45, 46, 46, 47, 47, 47, 48, 49, 49,
34, 35, 35, 36, 36, 36, 37, 40, 40, 41, 44, 44, 45, 45, 45, 46, 47, 47,
47, 46, 46, 45, 45, 45, 46, 46, 46, 47, 47, 48, 49, 49, 36, 38, 38, 38,
39, 40, 40, 43, 43, 44, 47, 47, 47, 47, 47, 47, 48, 48, 47, 46, 46, 45,
45, 45, 46, 46, 46, 46, 47, 47, 48, 48, 36, 38, 38, 38, 39, 40, 40, 43,
43, 44, 47, 47, 47, 47, 47, 47, 48, 48, 47, 46, 46, 45, 45, 45, 46, 46,
46, 46, 47, 47, 48, 48, 38, 39, 40, 40, 41, 41, 41, 43, 44, 45, 47, 47,
47, 48, 48, 48, 49, 49, 48, 48, 48, 47, 47, 47, 47, 47, 47, 48, 48, 48,
49, 49, 41, 42, 42, 42, 42, 42, 43, 44, 45, 45, 47, 47, 48, 48, 48, 49,
50, 50, 50, 49, 49, 49, 49, 49, 50, 50, 50, 50, 50, 51, 52, 52, 41, 42,
42, 42, 42, 42, 43, 44, 45, 45, 47, 47, 48, 48, 48, 49, 50, 50, 50, 49,
49, 49, 49, 50, 50, 50, 50, 50, 51, 52, 52, 45, 45, 45, 45, 44, 44,
44, 46, 46, 46, 47, 47, 48, 49, 49, 50, 51, 51, 51, 51, 51, 51, 51, 51,
52, 52, 52, 52, 52, 52, 53, 53, 49, 48, 47, 47, 46, 46, 46, 47, 47, 47,
48, 48, 49, 50, 50, 51, 53, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54,
54, 54, 55, 55, 49, 48, 47, 47, 46, 46, 46, 47, 47, 47, 48, 48, 49, 50,
50, 51, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 54, 55, 55,
49, 47, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 48, 50, 50, 51, 53, 53,
53, 54, 54, 54, 55, 55, 55, 55, 55, 56, 56, 56, 57, 57, 48, 47, 47, 47,
46, 45, 45, 46, 46, 46, 46, 46, 48, 49, 49, 51, 53, 53, 54, 54, 54, 55,
55, 56, 56, 56, 56, 57, 57, 58, 58, 58, 48, 47, 47, 47, 46, 45, 45, 46,
46, 46, 46, 46, 48, 49, 49, 51, 53, 53, 54, 54, 54, 55, 55, 56, 56, 56,
56, 57, 57, 58, 58, 58, 49, 47, 47, 47, 45, 45, 45, 45, 45, 45, 45,
47, 49, 49, 51, 53, 53, 54, 55, 55, 57, 57, 58, 58, 59, 59, 60, 60, 60,
61, 61, 49, 47, 47, 46, 45, 45, 45, 45, 45, 45, 45, 45, 47, 49, 49, 51,
53, 53, 55, 55, 55, 57, 58, 58, 59, 60, 60, 61, 61, 61, 62, 62, 49, 47,
47, 47, 45, 45, 45, 45, 45, 45, 45, 45, 47, 49, 49, 51, 53, 53, 55, 56,
56, 58, 58, 59, 59, 60, 60, 61, 61, 62, 63, 63, 50, 49, 48, 48, 46, 46,
46, 46, 46, 46, 46, 46, 47, 50, 50, 52, 54, 54, 55, 56, 56, 58, 59, 59,

```

```

61, 61, 61, 63, 63, 63, 64, 64, 50, 49, 48, 48, 47, 46, 46, 46, 46, 46,
46, 46, 47, 50, 50, 52, 54, 54, 55, 56, 56, 59, 60, 60, 61, 61, 62, 63,
63, 64, 65, 65, 51, 49, 48, 48, 47, 46, 46, 47, 47, 46, 46, 46, 47, 50,
50, 52, 54, 54, 55, 56, 56, 59, 60, 60, 61, 62, 62, 64, 64, 64, 66, 66,
52, 50, 49, 49, 48, 47, 47, 47, 47, 47, 46, 46, 48, 50, 50, 52, 54, 54,
56, 57, 57, 60, 61, 61, 63, 63, 64, 66, 66, 67, 68, 68, 52, 50, 50, 49,
48, 47, 47, 47, 47, 47, 47, 47, 48, 50, 50, 52, 54, 54, 56, 57, 57, 60,
61, 61, 63, 63, 64, 66, 66, 67, 68, 68, 53, 51, 50, 50, 48, 48, 48, 48,
48, 48, 47, 47, 48, 51, 51, 52, 54, 54, 56, 58, 58, 60, 61, 62, 63, 64,
64, 67, 67, 68, 69, 69, 54, 53, 52, 52, 50, 49, 49, 49, 49, 49, 48, 48,
49, 52, 52, 53, 55, 55, 57, 58, 58, 61, 62, 63, 64, 65, 66, 68, 68, 69,
71, 71, 54, 53, 52, 52, 50, 49, 49, 49, 49, 49, 48, 48, 49, 52, 52, 53,
55, 55, 57, 58, 58, 61, 62, 63, 64, 65, 66, 68, 68, 69, 71, 71,
/* Size 4x8 */
31, 38, 47, 50, 31, 40, 46, 48, 36, 44, 47, 47, 42, 47, 50, 50, 47, 48,
53, 54, 46, 46, 54, 60, 48, 46, 55, 64, 50, 48, 56, 67,
/* Size 8x4 */
31, 31, 36, 42, 47, 46, 48, 50, 38, 40, 44, 47, 48, 46, 46, 48, 47, 46,
47, 50, 53, 54, 55, 56, 50, 48, 47, 50, 54, 60, 64, 67,
/* Size 8x16 */
32, 31, 35, 38, 48, 49, 50, 52, 31, 31, 37, 40, 47, 47, 48, 50, 30, 32,
38, 40, 46, 45, 46, 48, 31, 33, 38, 41, 46, 45, 46, 48, 33, 36, 41, 44,
47, 46, 46, 47, 37, 40, 45, 47, 47, 45, 46, 47, 39, 41, 46, 47, 48, 47,
47, 48, 42, 43, 46, 48, 50, 49, 50, 50, 49, 46, 48, 49, 53, 53, 54, 54,
48, 46, 47, 48, 53, 55, 55, 56, 48, 46, 46, 48, 53, 56, 56, 57, 49, 45,
45, 47, 53, 58, 59, 61, 50, 46, 46, 48, 54, 59, 61, 63, 51, 47, 47, 48,
54, 60, 61, 64, 52, 48, 47, 48, 54, 61, 63, 66, 54, 50, 49, 50, 55, 62,
65, 68,
/* Size 16x8 */
32, 31, 30, 31, 33, 37, 39, 42, 49, 48, 48, 49, 50, 51, 52, 54, 31, 31,
32, 33, 36, 40, 41, 43, 46, 46, 46, 45, 46, 47, 48, 50, 35, 37, 38, 38,
41, 45, 46, 46, 48, 47, 46, 45, 46, 47, 47, 49, 38, 40, 40, 41, 44, 47,
47, 48, 49, 48, 48, 47, 48, 48, 48, 50, 48, 47, 46, 46, 47, 47, 48, 50,
53, 53, 53, 53, 54, 54, 54, 55, 49, 47, 45, 45, 46, 45, 47, 49, 53, 55,
56, 58, 59, 60, 61, 62, 50, 48, 46, 46, 46, 46, 47, 50, 54, 55, 56, 59,
61, 61, 63, 65, 52, 50, 48, 48, 47, 47, 48, 50, 54, 56, 57, 61, 63, 64,
66, 68,
/* Size 16x32 */
32, 31, 31, 31, 35, 37, 38, 47, 48, 48, 49, 49, 50, 52, 52, 54, 31, 31,
31, 32, 36, 38, 39, 46, 47, 47, 48, 48, 49, 50, 50, 53, 31, 31, 31, 32,
37, 38, 40, 46, 47, 47, 47, 47, 48, 50, 50, 52, 31, 31, 31, 32, 37, 38,
40, 46, 47, 47, 47, 47, 48, 50, 50, 52, 30, 31, 32, 32, 38, 39, 40, 45,
46, 46, 45, 45, 46, 48, 48, 50, 30, 31, 32, 33, 38, 40, 41, 45, 46, 46,
45, 45, 46, 48, 48, 50, 31, 32, 33, 33, 38, 40, 41, 45, 46, 46, 45, 45,
46, 48, 48, 50, 33, 35, 35, 36, 41, 43, 43, 46, 47, 46, 45, 45, 46, 47,
47, 49, 33, 35, 36, 36, 41, 43, 44, 46, 47, 46, 46, 46, 46, 47, 47, 49,
34, 36, 37, 37, 42, 44, 45, 47, 47, 47, 45, 45, 46, 47, 47, 49, 37, 39,
40, 41, 45, 47, 47, 47, 47, 47, 45, 45, 46, 47, 47, 48, 37, 39, 40, 41,
45, 47, 47, 47, 47, 47, 45, 45, 46, 47, 47, 48, 39, 40, 41, 42, 46, 47,
47, 48, 48, 48, 47, 47, 47, 48, 48, 50, 42, 42, 43, 43, 46, 47, 48, 50,

```

```

50, 50, 49, 49, 50, 50, 50, 52, 42, 42, 43, 43, 46, 47, 48, 50, 50, 50,
49, 49, 50, 50, 50, 52, 45, 45, 44, 45, 47, 47, 48, 51, 51, 51, 51, 51,
52, 52, 52, 54, 49, 47, 46, 47, 48, 48, 49, 52, 53, 53, 53, 53, 54, 54,
54, 55, 49, 47, 46, 47, 48, 48, 49, 52, 53, 53, 53, 53, 54, 54, 54, 55,
48, 47, 46, 46, 47, 47, 48, 52, 53, 53, 55, 55, 55, 56, 56, 57, 48, 46,
46, 46, 46, 47, 48, 52, 53, 54, 56, 56, 56, 57, 57, 59, 48, 46, 46, 46,
46, 47, 48, 52, 53, 54, 56, 56, 56, 57, 57, 59, 49, 46, 45, 45, 46, 46,
47, 52, 53, 54, 57, 57, 58, 60, 60, 61, 49, 46, 45, 45, 45, 46, 47, 52,
53, 55, 58, 58, 59, 61, 61, 62, 49, 46, 45, 45, 46, 46, 47, 52, 53, 55,
58, 58, 60, 61, 61, 63, 50, 47, 46, 46, 46, 46, 48, 53, 54, 55, 59, 59,
61, 63, 63, 65, 50, 48, 46, 46, 46, 46, 48, 53, 54, 55, 59, 59, 61, 64,
64, 65, 51, 48, 47, 47, 47, 47, 48, 53, 54, 55, 60, 60, 61, 64, 64, 66,
52, 49, 48, 48, 47, 47, 48, 53, 54, 56, 61, 61, 63, 66, 66, 68, 52, 49,
48, 48, 47, 47, 48, 53, 54, 56, 61, 61, 63, 66, 66, 68, 53, 50, 48, 48,
48, 48, 49, 54, 54, 56, 61, 61, 63, 67, 67, 69, 54, 51, 50, 50, 49, 49,
50, 55, 55, 57, 62, 62, 65, 68, 68, 71, 54, 51, 50, 50, 49, 49, 50, 55,
55, 57, 62, 62, 65, 68, 68, 71,

```

```

/* Size 32x16 */

```

```

32, 31, 31, 31, 30, 30, 31, 33, 33, 34, 37, 37, 39, 42, 42, 45, 49, 49,
48, 48, 48, 49, 49, 49, 50, 50, 51, 52, 52, 53, 54, 54, 31, 31, 31, 31,
31, 31, 32, 35, 35, 36, 39, 39, 40, 42, 42, 45, 47, 47, 47, 46, 46, 46,
46, 46, 47, 48, 48, 49, 49, 50, 51, 51, 31, 31, 31, 31, 32, 32, 33, 35,
36, 37, 40, 40, 41, 43, 43, 44, 46, 46, 46, 46, 46, 45, 45, 45, 46, 46,
47, 48, 48, 48, 50, 50, 31, 32, 32, 32, 32, 33, 33, 36, 36, 37, 41, 41,
42, 43, 43, 45, 47, 47, 46, 46, 46, 45, 45, 45, 46, 46, 47, 48, 48, 48,
50, 50, 35, 36, 37, 37, 38, 38, 38, 41, 41, 42, 45, 45, 46, 46, 46, 47,
48, 48, 47, 46, 46, 46, 45, 46, 46, 46, 47, 47, 47, 48, 49, 49, 37, 38,
38, 38, 39, 40, 40, 43, 43, 44, 47, 47, 47, 47, 47, 47, 48, 48, 47, 47,
47, 46, 46, 46, 46, 46, 47, 47, 47, 48, 49, 49, 38, 39, 40, 40, 40, 41,
41, 43, 44, 45, 47, 47, 47, 48, 48, 48, 49, 49, 48, 48, 48, 47, 47, 47,
48, 48, 48, 48, 48, 49, 50, 50, 47, 46, 46, 46, 45, 45, 45, 46, 46, 47,
47, 47, 48, 50, 50, 51, 52, 52, 52, 52, 52, 52, 52, 53, 53, 53, 53,
53, 54, 55, 55, 48, 47, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 48, 50,
50, 51, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 54, 55, 55,
48, 47, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 48, 50, 50, 51, 53, 53,
53, 54, 54, 54, 55, 55, 55, 55, 55, 56, 56, 56, 57, 57, 49, 48, 47, 47,
45, 45, 45, 45, 46, 45, 45, 45, 47, 49, 49, 51, 53, 53, 55, 56, 56, 57,
58, 58, 59, 59, 60, 61, 61, 61, 62, 62, 49, 48, 47, 47, 45, 45, 45, 45,
46, 45, 45, 45, 47, 49, 49, 51, 53, 53, 55, 56, 56, 57, 58, 58, 59, 59,
60, 61, 61, 61, 62, 62, 50, 49, 48, 48, 46, 46, 46, 46, 46, 46, 46, 46,
47, 50, 50, 52, 54, 54, 55, 56, 56, 58, 59, 60, 61, 61, 61, 63, 63, 63,
65, 65, 52, 50, 50, 50, 48, 48, 48, 47, 47, 47, 47, 47, 48, 50, 50, 52,
54, 54, 56, 57, 57, 60, 61, 61, 63, 64, 64, 66, 66, 67, 68, 68, 52, 50,
50, 50, 48, 48, 48, 47, 47, 47, 47, 47, 48, 50, 50, 52, 54, 54, 56, 57,
57, 60, 61, 61, 63, 64, 64, 66, 66, 67, 68, 68, 54, 53, 52, 52, 50, 50,
50, 49, 49, 49, 48, 48, 50, 52, 52, 54, 55, 55, 57, 59, 59, 61, 62, 63,
65, 65, 66, 68, 68, 69, 71, 71,

```

```

/* Size 4x16 */

```

```

31, 37, 48, 52, 31, 38, 47, 50, 31, 39, 46, 48, 32, 40, 46, 48, 35, 43,
46, 47, 39, 47, 47, 47, 40, 47, 48, 48, 42, 47, 50, 50, 47, 48, 53, 54,

```

```

47, 47, 53, 56, 46, 47, 54, 57, 46, 46, 55, 61, 47, 46, 55, 63, 48, 47,
55, 64, 49, 47, 56, 66, 51, 49, 57, 68,
/* Size 16x4 */
31, 31, 31, 32, 35, 39, 40, 42, 47, 47, 46, 46, 47, 48, 49, 51, 37, 38,
39, 40, 43, 47, 47, 47, 48, 47, 47, 46, 46, 47, 47, 49, 48, 47, 46, 46,
46, 47, 48, 50, 53, 53, 54, 55, 55, 55, 56, 57, 52, 50, 48, 48, 47, 47,
48, 50, 54, 56, 57, 61, 63, 64, 66, 68,
/* Size 8x32 */
32, 31, 35, 38, 48, 49, 50, 52, 31, 31, 36, 39, 47, 48, 49, 50, 31, 31,
37, 40, 47, 47, 48, 50, 31, 31, 37, 40, 47, 47, 48, 50, 30, 32, 38, 40,
46, 45, 46, 48, 30, 32, 38, 41, 46, 45, 46, 48, 31, 33, 38, 41, 46, 45,
46, 48, 33, 35, 41, 43, 47, 45, 46, 47, 33, 36, 41, 44, 47, 46, 46, 47,
34, 37, 42, 45, 47, 45, 46, 47, 37, 40, 45, 47, 47, 45, 46, 47, 37, 40,
45, 47, 47, 45, 46, 47, 39, 41, 46, 47, 48, 47, 47, 48, 42, 43, 46, 48,
50, 49, 50, 50, 42, 43, 46, 48, 50, 49, 50, 50, 45, 44, 47, 48, 51, 51,
52, 52, 49, 46, 48, 49, 53, 53, 54, 54, 49, 46, 48, 49, 53, 53, 54, 54,
48, 46, 47, 48, 53, 55, 55, 56, 48, 46, 46, 48, 53, 56, 56, 57, 48, 46,
46, 48, 53, 56, 56, 57, 49, 45, 46, 47, 53, 57, 58, 60, 49, 45, 45, 47,
53, 58, 59, 61, 49, 45, 46, 47, 53, 58, 60, 61, 50, 46, 46, 48, 54, 59,
61, 63, 50, 46, 46, 48, 54, 59, 61, 64, 51, 47, 47, 48, 54, 60, 61, 64,
52, 48, 47, 48, 54, 61, 63, 66, 52, 48, 47, 48, 54, 61, 63, 66, 53, 48,
48, 49, 54, 61, 63, 67, 54, 50, 49, 50, 55, 62, 65, 68, 54, 50, 49, 50,
55, 62, 65, 68,
/* Size 32x8 */
32, 31, 31, 31, 30, 30, 31, 33, 33, 34, 37, 37, 39, 42, 42, 45, 49, 49,
48, 48, 48, 49, 49, 49, 50, 50, 51, 52, 52, 53, 54, 54, 31, 31, 31, 31,
32, 32, 33, 35, 36, 37, 40, 40, 41, 43, 43, 44, 46, 46, 46, 46, 46, 45,
45, 45, 46, 46, 47, 48, 48, 48, 50, 50, 35, 36, 37, 37, 38, 38, 38, 41,
41, 42, 45, 45, 46, 46, 46, 47, 48, 48, 47, 46, 46, 46, 45, 46, 46, 46,
47, 47, 47, 48, 49, 49, 38, 39, 40, 40, 40, 41, 41, 43, 44, 45, 47, 47,
47, 48, 48, 48, 49, 49, 48, 48, 48, 47, 47, 47, 48, 48, 48, 48, 48, 49,
50, 50, 48, 47, 47, 47, 46, 46, 46, 47, 47, 47, 47, 47, 48, 50, 50, 51,
53, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 54, 55, 55, 49, 48,
47, 47, 45, 45, 45, 45, 46, 45, 45, 45, 47, 49, 49, 51, 53, 53, 55, 56,
56, 57, 58, 58, 59, 59, 60, 61, 61, 61, 62, 62, 50, 49, 48, 48, 46, 46,
46, 46, 46, 46, 46, 47, 50, 50, 52, 54, 54, 55, 56, 56, 58, 59, 60,
61, 61, 61, 63, 63, 63, 65, 65, 52, 50, 50, 50, 48, 48, 48, 47, 47, 47,
47, 47, 48, 50, 50, 52, 54, 54, 56, 57, 57, 60, 61, 61, 63, 64, 64, 66,
66, 67, 68, 68 },
},
{
{ /* Luma */
/* Size 4x4 */
32, 32, 35, 43, 32, 34, 37, 43, 35, 37, 48, 54, 43, 43, 54, 65,
/* Size 8x8 */
31, 31, 32, 32, 34, 37, 43, 47, 31, 32, 32, 32, 34, 36, 41, 44, 32, 32,
33, 34, 35, 38, 42, 45, 32, 32, 34, 35, 37, 39, 42, 46, 34, 34, 35, 37,
41, 45, 49, 52, 37, 36, 38, 39, 45, 51, 56, 59, 43, 41, 42, 42, 49, 56,
63, 67, 47, 44, 45, 46, 52, 59, 67, 71,
/* Size 16x16 */

```

32, 31, 31, 31, 31, 31, 32, 32, 34, 35, 36, 39, 41, 44, 47, 48, 31, 32,  
 32, 32, 32, 32, 32, 33, 34, 35, 35, 38, 40, 42, 45, 46, 31, 32, 32, 32,  
 32, 32, 32, 33, 34, 34, 35, 38, 39, 42, 45, 45, 31, 32, 32, 32, 32, 32,  
 32, 33, 33, 34, 34, 37, 38, 41, 44, 44, 31, 32, 32, 32, 33, 33, 33, 34,  
 35, 36, 36, 39, 40, 42, 44, 45, 31, 32, 32, 32, 33, 33, 34, 34, 35, 36,  
 36, 39, 40, 42, 45, 45, 32, 32, 32, 32, 33, 34, 35, 36, 37, 38, 38, 40,  
 41, 42, 45, 46, 32, 33, 33, 33, 34, 34, 36, 36, 38, 39, 40, 42, 43, 44,  
 47, 47, 34, 34, 34, 33, 35, 35, 37, 38, 39, 42, 42, 45, 46, 47, 50, 51,  
 35, 35, 34, 34, 36, 36, 38, 39, 42, 46, 47, 49, 50, 52, 55, 55, 36, 35,  
 35, 34, 36, 36, 38, 40, 42, 47, 48, 50, 52, 54, 56, 57, 39, 38, 38, 37,  
 39, 39, 40, 42, 45, 49, 50, 54, 55, 58, 60, 61, 41, 40, 39, 38, 40, 40,  
 41, 43, 46, 50, 52, 55, 57, 60, 62, 63, 44, 42, 42, 41, 42, 42, 42, 44,  
 47, 52, 54, 58, 60, 63, 66, 67, 47, 45, 45, 44, 44, 45, 45, 47, 50, 55,  
 56, 60, 62, 66, 69, 70, 48, 46, 45, 44, 45, 45, 46, 47, 51, 55, 57, 61,  
 63, 67, 70, 71,

/\* Size 32x32 \*/

32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 34,  
 35, 36, 36, 38, 39, 39, 41, 44, 44, 45, 47, 48, 48, 51, 31, 31, 31, 31,  
 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 35, 37,  
 39, 39, 40, 43, 43, 44, 46, 47, 47, 50, 31, 31, 32, 32, 32, 32, 32, 32,  
 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 35, 37, 38, 38, 40, 42,  
 42, 43, 45, 46, 46, 49, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
 32, 32, 33, 34, 34, 34, 35, 35, 35, 37, 38, 38, 40, 42, 42, 43, 45, 46,  
 46, 49, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34,  
 34, 34, 34, 35, 35, 36, 38, 38, 39, 42, 42, 42, 45, 45, 45, 48, 31, 31,  
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34,  
 34, 36, 37, 37, 38, 41, 41, 41, 44, 44, 44, 47, 31, 31, 32, 32, 32, 32,  
 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 36, 37, 37,  
 38, 41, 41, 41, 44, 44, 44, 47, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
 32, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 36, 38, 38, 39, 41, 41, 42,  
 44, 45, 45, 47, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33,  
 34, 35, 35, 35, 36, 36, 36, 37, 39, 39, 40, 42, 42, 42, 44, 45, 45, 48,  
 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 35, 35,  
 36, 36, 36, 38, 39, 39, 40, 42, 42, 42, 45, 45, 45, 48, 31, 32, 32, 32,  
 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 36, 36, 38,  
 39, 39, 40, 42, 42, 42, 45, 45, 45, 48, 32, 32, 32, 32, 32, 32, 32, 33,  
 33, 33, 33, 34, 35, 35, 35, 36, 36, 36, 37, 37, 37, 39, 40, 40, 41, 42,  
 42, 43, 45, 45, 45, 48, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 35,  
 35, 35, 36, 37, 37, 37, 38, 38, 38, 39, 40, 40, 41, 42, 42, 43, 45, 46,  
 46, 48, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 35, 35, 35, 36, 37,  
 37, 37, 38, 38, 38, 39, 40, 40, 41, 42, 42, 43, 45, 46, 46, 48, 32, 33,  
 33, 33, 33, 33, 33, 33, 34, 34, 34, 35, 36, 36, 36, 38, 38, 38, 39, 40,  
 40, 41, 42, 42, 43, 44, 44, 45, 47, 47, 47, 50, 34, 34, 34, 34, 34, 33,  
 33, 34, 35, 35, 35, 36, 37, 37, 38, 39, 39, 40, 42, 42, 42, 44, 45, 45,  
 46, 47, 47, 48, 50, 51, 51, 53, 34, 34, 34, 34, 34, 33, 33, 34, 35, 35,  
 35, 36, 37, 37, 38, 39, 39, 40, 42, 42, 42, 44, 45, 45, 46, 47, 47, 48,  
 50, 51, 51, 53, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 36, 37, 37,  
 38, 40, 40, 41, 43, 44, 44, 45, 46, 46, 47, 49, 49, 49, 51, 52, 52, 54,  
 35, 35, 35, 35, 34, 34, 34, 34, 36, 36, 36, 37, 38, 38, 39, 42, 42, 43,  
 46, 47, 47, 48, 49, 49, 50, 52, 52, 53, 55, 55, 55, 57, 36, 35, 35, 35,

```

35, 34, 34, 35, 36, 36, 36, 37, 38, 38, 40, 42, 42, 44, 47, 48, 48, 50,
50, 50, 52, 54, 54, 54, 56, 57, 57, 58, 36, 35, 35, 35, 35, 34, 34, 35,
36, 36, 36, 37, 38, 38, 40, 42, 42, 44, 47, 48, 48, 50, 50, 50, 52, 54,
54, 54, 56, 57, 57, 58, 38, 37, 37, 37, 36, 36, 36, 36, 37, 38, 38, 39,
39, 39, 41, 44, 44, 45, 48, 50, 50, 51, 52, 52, 54, 56, 56, 57, 58, 59,
59, 61, 39, 39, 38, 38, 38, 37, 37, 38, 39, 39, 39, 40, 40, 40, 42, 45,
45, 46, 49, 50, 50, 52, 54, 54, 55, 58, 58, 58, 60, 61, 61, 63, 39, 39,
38, 38, 38, 37, 37, 38, 39, 39, 39, 40, 40, 40, 42, 45, 45, 46, 49, 50,
50, 52, 54, 54, 55, 58, 58, 58, 60, 61, 61, 63, 41, 40, 40, 40, 39, 38,
38, 39, 40, 40, 40, 41, 41, 41, 43, 46, 46, 47, 50, 52, 52, 54, 55, 55,
57, 60, 60, 60, 62, 63, 63, 66, 44, 43, 42, 42, 42, 41, 41, 41, 42, 42,
42, 42, 42, 44, 47, 47, 49, 52, 54, 54, 56, 58, 58, 60, 63, 63, 64,
66, 67, 67, 69, 44, 43, 42, 42, 42, 41, 41, 41, 42, 42, 42, 42, 42, 42,
44, 47, 47, 49, 52, 54, 54, 56, 58, 58, 60, 63, 63, 64, 66, 67, 67, 69,
45, 44, 43, 43, 42, 41, 41, 42, 42, 42, 42, 43, 43, 43, 45, 48, 48, 49,
53, 54, 54, 57, 58, 58, 60, 64, 64, 65, 67, 68, 68, 70, 47, 46, 45, 45,
45, 44, 44, 44, 44, 45, 45, 45, 45, 45, 47, 50, 50, 51, 55, 56, 56, 58,
60, 60, 62, 66, 66, 67, 69, 70, 70, 73, 48, 47, 46, 46, 45, 44, 44, 45,
45, 45, 45, 46, 46, 47, 51, 51, 52, 55, 57, 57, 59, 61, 61, 63, 67,
67, 68, 70, 71, 71, 74, 48, 47, 46, 46, 45, 44, 44, 45, 45, 45, 45, 45,
46, 46, 47, 51, 51, 52, 55, 57, 57, 59, 61, 61, 63, 67, 67, 68, 70, 71,
71, 74, 51, 50, 49, 48, 47, 47, 47, 48, 48, 48, 48, 48, 48, 50, 53,
53, 54, 57, 58, 58, 61, 63, 63, 66, 69, 69, 70, 73, 74, 74, 77,
/* Size 4x8 */
31, 32, 35, 43, 32, 33, 34, 41, 32, 34, 36, 42, 32, 35, 38, 42, 34, 37,
43, 49, 37, 40, 49, 56, 42, 43, 53, 63, 46, 46, 56, 67,
/* Size 8x4 */
31, 32, 32, 32, 34, 37, 42, 46, 32, 33, 34, 35, 37, 40, 43, 46, 35, 34,
36, 38, 43, 49, 53, 56, 43, 41, 42, 42, 49, 56, 63, 67,
/* Size 8x16 */
32, 31, 31, 32, 35, 36, 44, 47, 31, 32, 32, 32, 35, 35, 42, 45, 31, 32,
32, 32, 34, 35, 41, 45, 31, 32, 32, 33, 34, 34, 41, 44, 31, 32, 33, 34,
35, 36, 42, 44, 32, 32, 33, 34, 36, 36, 42, 45, 32, 33, 34, 35, 37, 38,
42, 45, 32, 33, 34, 36, 39, 40, 44, 47, 34, 34, 35, 37, 41, 42, 48, 50,
35, 34, 36, 38, 45, 47, 52, 55, 36, 34, 36, 38, 46, 48, 54, 56, 39, 37,
39, 40, 48, 50, 58, 60, 41, 39, 40, 41, 49, 51, 60, 62, 44, 41, 42, 43,
51, 53, 63, 66, 47, 44, 44, 45, 53, 56, 66, 69, 48, 45, 45, 46, 54, 56,
67, 70,
/* Size 16x8 */
32, 31, 31, 31, 31, 32, 32, 32, 34, 35, 36, 39, 41, 44, 47, 48, 31, 32,
32, 32, 32, 32, 33, 33, 34, 34, 34, 37, 39, 41, 44, 45, 31, 32, 32, 32,
33, 33, 34, 34, 35, 36, 36, 39, 40, 42, 44, 45, 32, 32, 32, 33, 34, 34,
35, 36, 37, 38, 38, 40, 41, 43, 45, 46, 35, 35, 34, 34, 35, 36, 37, 39,
41, 45, 46, 48, 49, 51, 53, 54, 36, 35, 35, 34, 36, 36, 38, 40, 42, 47,
48, 50, 51, 53, 56, 56, 44, 42, 41, 41, 42, 42, 42, 44, 48, 52, 54, 58,
60, 63, 66, 67, 47, 45, 45, 44, 44, 45, 45, 47, 50, 55, 56, 60, 62, 66,
69, 70,
/* Size 16x32 */
32, 31, 31, 31, 31, 32, 32, 32, 35, 36, 36, 40, 44, 44, 47, 53, 31, 31,
32, 32, 32, 32, 32, 33, 35, 35, 35, 39, 43, 43, 46, 52, 31, 32, 32, 32,

```

```

32, 32, 32, 33, 35, 35, 35, 39, 42, 42, 45, 51, 31, 32, 32, 32, 32, 32,
32, 33, 35, 35, 35, 39, 42, 42, 45, 51, 31, 32, 32, 32, 32, 32, 33,
34, 35, 35, 39, 41, 41, 45, 50, 31, 32, 32, 32, 32, 33, 33, 33, 34, 34,
34, 38, 41, 41, 44, 49, 31, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 38,
41, 41, 44, 49, 31, 32, 32, 32, 32, 33, 33, 33, 34, 35, 35, 38, 41, 41,
44, 49, 31, 32, 32, 32, 33, 34, 34, 34, 35, 36, 36, 39, 42, 42, 44, 49,
32, 32, 32, 32, 33, 34, 34, 34, 36, 36, 36, 39, 42, 42, 45, 50, 32, 32,
32, 32, 33, 34, 34, 34, 36, 36, 36, 39, 42, 42, 45, 50, 32, 32, 32, 32,
33, 35, 35, 35, 37, 37, 37, 40, 42, 42, 45, 49, 32, 32, 33, 33, 34, 35,
35, 36, 37, 38, 38, 41, 42, 42, 45, 49, 32, 32, 33, 33, 34, 35, 35, 36,
37, 38, 38, 41, 42, 42, 45, 49, 32, 33, 33, 33, 34, 36, 36, 36, 39, 40,
40, 42, 44, 44, 47, 51, 34, 34, 34, 34, 35, 37, 37, 38, 41, 42, 42, 45,
48, 48, 50, 54, 34, 34, 34, 34, 35, 37, 37, 38, 41, 42, 42, 45, 48, 48,
50, 54, 34, 34, 34, 34, 35, 37, 37, 38, 42, 43, 43, 46, 49, 49, 51, 55,
35, 35, 34, 34, 36, 38, 38, 39, 45, 47, 47, 50, 52, 52, 55, 59, 36, 35,
34, 34, 36, 38, 38, 40, 46, 48, 48, 51, 54, 54, 56, 60, 36, 35, 34, 34,
36, 38, 38, 40, 46, 48, 48, 51, 54, 54, 56, 60, 38, 37, 36, 36, 37, 40,
40, 41, 47, 49, 49, 53, 56, 56, 58, 63, 39, 38, 37, 37, 39, 40, 40, 42,
48, 50, 50, 54, 58, 58, 60, 65, 39, 38, 37, 37, 39, 40, 40, 42, 48, 50,
50, 54, 58, 58, 60, 65, 41, 40, 39, 39, 40, 41, 41, 43, 49, 51, 51, 56,
60, 60, 62, 67, 44, 42, 41, 41, 42, 43, 43, 45, 51, 53, 53, 59, 63, 63,
66, 71, 44, 42, 41, 41, 42, 43, 43, 45, 51, 53, 53, 59, 63, 63, 66, 71,
44, 43, 42, 42, 42, 43, 43, 45, 51, 54, 54, 59, 64, 64, 67, 72, 47, 45,
44, 44, 44, 45, 45, 47, 53, 56, 56, 61, 66, 66, 69, 75, 48, 46, 45, 45,
45, 46, 46, 48, 54, 56, 56, 62, 67, 67, 70, 76, 48, 46, 45, 45, 45, 46,
46, 48, 54, 56, 56, 62, 67, 67, 70, 76, 51, 49, 47, 47, 48, 48, 48, 50,
56, 58, 58, 64, 69, 69, 73, 79,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 34, 34,
35, 36, 36, 38, 39, 39, 41, 44, 44, 44, 47, 48, 48, 51, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 35, 35, 37,
38, 38, 40, 42, 42, 43, 45, 46, 46, 49, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 36, 37, 37, 39, 41,
41, 42, 44, 45, 45, 47, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 36, 37, 37, 39, 41, 41, 42, 44, 45,
45, 47, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35,
35, 35, 36, 36, 36, 37, 39, 39, 40, 42, 42, 42, 44, 45, 45, 48, 32, 32,
32, 32, 32, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 38, 38,
38, 40, 40, 40, 41, 43, 43, 43, 45, 46, 46, 48, 32, 32, 32, 32, 32, 32, 33,
33, 33, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 38, 38, 38, 40, 40, 40,
41, 43, 43, 43, 45, 46, 46, 48, 32, 33, 33, 33, 33, 33, 33, 33, 34, 34,
34, 35, 36, 36, 36, 38, 38, 38, 39, 40, 40, 41, 42, 42, 43, 45, 45, 45,
47, 48, 48, 50, 35, 35, 35, 35, 34, 34, 34, 34, 35, 36, 36, 37, 37, 37,
39, 41, 41, 42, 45, 46, 46, 47, 48, 48, 49, 51, 51, 51, 53, 54, 54, 56,
36, 35, 35, 35, 35, 34, 34, 35, 36, 36, 36, 37, 38, 38, 40, 42, 42, 43,
47, 48, 48, 49, 50, 50, 51, 53, 53, 54, 56, 56, 56, 58, 36, 35, 35, 35,
35, 34, 34, 35, 36, 36, 36, 37, 38, 38, 40, 42, 42, 43, 47, 48, 48, 49,
50, 50, 51, 53, 53, 54, 56, 56, 56, 58, 40, 39, 39, 39, 39, 38, 38, 38,
39, 39, 39, 40, 41, 41, 42, 45, 45, 46, 50, 51, 51, 53, 54, 54, 56, 59,
59, 59, 61, 62, 62, 64, 44, 43, 42, 42, 41, 41, 41, 41, 42, 42, 42, 42,

```

```

42, 42, 44, 48, 48, 49, 52, 54, 54, 56, 58, 58, 60, 63, 63, 64, 66, 67,
67, 69, 44, 43, 42, 42, 41, 41, 41, 41, 42, 42, 42, 42, 42, 42, 44, 48,
48, 49, 52, 54, 54, 56, 58, 58, 60, 63, 63, 64, 66, 67, 67, 69, 47, 46,
45, 45, 45, 44, 44, 44, 44, 45, 45, 45, 45, 45, 47, 50, 50, 51, 55, 56,
56, 58, 60, 60, 62, 66, 66, 67, 69, 70, 70, 73, 53, 52, 51, 51, 50, 49,
49, 49, 49, 50, 50, 49, 49, 49, 51, 54, 54, 55, 59, 60, 60, 63, 65, 65,
67, 71, 71, 72, 75, 76, 76, 79,
/* Size 4x16 */
31, 32, 36, 44, 32, 32, 35, 42, 32, 32, 35, 41, 32, 33, 34, 41, 32, 34,
36, 42, 32, 34, 36, 42, 32, 35, 38, 42, 33, 36, 40, 44, 34, 37, 42, 48,
35, 38, 47, 52, 35, 38, 48, 54, 38, 40, 50, 58, 40, 41, 51, 60, 42, 43,
53, 63, 45, 45, 56, 66, 46, 46, 56, 67,
/* Size 16x4 */
31, 32, 32, 32, 32, 32, 32, 33, 34, 35, 35, 38, 40, 42, 45, 46, 32, 32,
32, 33, 34, 34, 35, 36, 37, 38, 38, 40, 41, 43, 45, 46, 36, 35, 35, 34,
36, 36, 38, 40, 42, 47, 48, 50, 51, 53, 56, 56, 44, 42, 41, 41, 42, 42,
42, 44, 48, 52, 54, 58, 60, 63, 66, 67,
/* Size 8x32 */
32, 31, 31, 32, 35, 36, 44, 47, 31, 32, 32, 32, 35, 35, 43, 46, 31, 32,
32, 32, 35, 35, 42, 45, 31, 32, 32, 32, 35, 35, 42, 45, 31, 32, 32, 32,
34, 35, 41, 45, 31, 32, 32, 33, 34, 34, 41, 44, 31, 32, 32, 33, 34, 34,
41, 44, 31, 32, 32, 33, 34, 35, 41, 44, 31, 32, 33, 34, 35, 36, 42, 44,
32, 32, 33, 34, 36, 36, 42, 45, 32, 32, 33, 34, 36, 36, 42, 45, 32, 32,
33, 35, 37, 37, 42, 45, 32, 33, 34, 35, 37, 38, 42, 45, 32, 33, 34, 35,
37, 38, 42, 45, 32, 33, 34, 36, 39, 40, 44, 47, 34, 34, 35, 37, 41, 42,
48, 50, 34, 34, 35, 37, 41, 42, 48, 50, 34, 34, 35, 37, 42, 43, 49, 51,
35, 34, 36, 38, 45, 47, 52, 55, 36, 34, 36, 38, 46, 48, 54, 56, 36, 34,
36, 38, 46, 48, 54, 56, 38, 36, 37, 40, 47, 49, 56, 58, 39, 37, 39, 40,
48, 50, 58, 60, 39, 37, 39, 40, 48, 50, 58, 60, 41, 39, 40, 41, 49, 51,
60, 62, 44, 41, 42, 43, 51, 53, 63, 66, 44, 41, 42, 43, 51, 53, 63, 66,
44, 42, 42, 43, 51, 54, 64, 67, 47, 44, 44, 45, 53, 56, 66, 69, 48, 45,
45, 46, 54, 56, 67, 70, 48, 45, 45, 46, 54, 56, 67, 70, 51, 47, 48, 48,
56, 58, 69, 73,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 34, 34,
35, 36, 36, 38, 39, 39, 41, 44, 44, 44, 47, 48, 48, 51, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 36,
37, 37, 39, 41, 41, 42, 44, 45, 45, 47, 31, 32, 32, 32, 32, 32, 32, 32,
33, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 36, 36, 37, 39, 39, 40, 42,
42, 42, 44, 45, 45, 48, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 35,
35, 35, 36, 37, 37, 37, 38, 38, 38, 40, 40, 40, 41, 43, 43, 43, 45, 46,
46, 48, 35, 35, 35, 35, 34, 34, 34, 34, 35, 36, 36, 37, 37, 37, 39, 41,
41, 42, 45, 46, 46, 47, 48, 48, 49, 51, 51, 51, 53, 54, 54, 56, 36, 35,
35, 35, 35, 34, 34, 35, 36, 36, 36, 37, 38, 38, 40, 42, 42, 43, 47, 48,
48, 49, 50, 50, 51, 53, 53, 54, 56, 56, 56, 58, 44, 43, 42, 42, 41, 41,
41, 41, 42, 42, 42, 42, 42, 42, 44, 48, 48, 49, 52, 54, 54, 56, 58, 58,
60, 63, 63, 64, 66, 67, 67, 69, 47, 46, 45, 45, 45, 44, 44, 44, 44, 45,
45, 45, 45, 47, 50, 50, 51, 55, 56, 56, 58, 60, 60, 62, 66, 66, 67,
69, 70, 70, 73 },
{ /* Chroma */

```



```

/* Size 4x4 */
31, 37, 47, 47, 37, 44, 47, 45, 47, 47, 53, 53, 47, 45, 53, 59,
/* Size 8x8 */
31, 31, 34, 37, 43, 48, 47, 49, 31, 32, 35, 40, 43, 46, 45, 46, 34, 35,
39, 43, 45, 46, 45, 46, 37, 40, 43, 47, 47, 47, 45, 46, 43, 43, 45, 47,
49, 50, 50, 50, 48, 46, 46, 47, 50, 53, 55, 55, 47, 45, 45, 45, 50, 55,
58, 60, 49, 46, 46, 46, 50, 55, 60, 61,
/* Size 16x16 */
32, 31, 31, 30, 33, 33, 36, 38, 41, 47, 49, 48, 49, 49, 50, 50, 31, 31,
31, 31, 34, 34, 38, 40, 42, 46, 47, 47, 47, 47, 48, 48, 31, 31, 31, 31,
34, 35, 39, 40, 42, 46, 47, 46, 46, 46, 47, 47, 30, 31, 31, 32, 34, 35,
40, 41, 42, 45, 46, 45, 45, 45, 46, 46, 33, 34, 34, 34, 37, 38, 42, 43,
44, 46, 47, 46, 46, 45, 46, 46, 33, 34, 35, 35, 38, 39, 43, 44, 45, 47,
47, 46, 46, 45, 46, 46, 36, 38, 39, 40, 42, 43, 47, 47, 47, 47, 48, 46,
46, 45, 46, 46, 38, 40, 40, 41, 43, 44, 47, 47, 48, 48, 49, 48, 47, 47,
47, 47, 41, 42, 42, 42, 44, 45, 47, 48, 48, 50, 50, 49, 49, 49, 50, 50,
47, 46, 46, 45, 46, 47, 47, 48, 50, 52, 52, 52, 52, 52, 53, 53, 49, 47,
47, 46, 47, 47, 48, 49, 50, 52, 53, 53, 53, 53, 54, 54, 48, 47, 46, 45,
46, 46, 46, 48, 49, 52, 53, 54, 55, 55, 56, 56, 49, 47, 46, 45, 46, 46,
46, 47, 49, 52, 53, 55, 55, 57, 57, 58, 49, 47, 46, 45, 45, 45, 45, 47,
49, 52, 53, 55, 57, 58, 59, 60, 50, 48, 47, 46, 46, 46, 46, 47, 50, 53,
54, 56, 57, 59, 61, 61, 50, 48, 47, 46, 46, 46, 46, 47, 50, 53, 54, 56,
58, 60, 61, 61,
/* Size 32x32 */
32, 31, 31, 31, 31, 30, 30, 31, 33, 33, 33, 35, 36, 36, 38, 41, 41, 43,
47, 49, 49, 49, 48, 48, 49, 49, 49, 49, 50, 50, 50, 51, 31, 31, 31, 31,
31, 31, 31, 31, 33, 34, 34, 36, 37, 37, 39, 42, 42, 43, 47, 48, 48, 48,
47, 47, 47, 47, 47, 48, 49, 49, 49, 50, 31, 31, 31, 31, 31, 31, 31, 32,
34, 34, 34, 37, 38, 38, 40, 42, 42, 43, 46, 47, 47, 47, 47, 47, 47, 47,
47, 47, 48, 48, 48, 49, 31, 31, 31, 31, 31, 31, 31, 32, 34, 34, 34, 37,
38, 38, 40, 42, 42, 43, 46, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48,
48, 49, 31, 31, 31, 31, 31, 31, 31, 32, 34, 35, 35, 37, 39, 39, 40, 42,
42, 43, 46, 47, 47, 46, 46, 46, 46, 46, 46, 46, 47, 47, 47, 48, 30, 31,
31, 31, 31, 32, 32, 32, 34, 35, 35, 38, 40, 40, 41, 42, 42, 43, 45, 46,
46, 46, 45, 45, 45, 45, 45, 45, 46, 46, 46, 47, 30, 31, 31, 31, 31, 32,
32, 32, 34, 35, 35, 38, 40, 40, 41, 42, 42, 43, 45, 46, 46, 46, 45, 45,
45, 45, 45, 45, 46, 46, 46, 47, 31, 31, 32, 32, 32, 32, 32, 33, 35, 36,
36, 38, 40, 40, 41, 43, 43, 43, 46, 46, 46, 46, 45, 45, 45, 45, 45, 45,
46, 46, 46, 47, 33, 33, 34, 34, 34, 34, 34, 35, 37, 38, 38, 41, 42, 42,
43, 44, 44, 45, 46, 47, 47, 46, 46, 46, 46, 45, 45, 45, 46, 46, 46, 47,
33, 34, 34, 34, 35, 35, 35, 36, 38, 39, 39, 41, 43, 43, 44, 45, 45, 45,
47, 47, 47, 46, 46, 46, 46, 45, 45, 45, 46, 46, 46, 47, 33, 34, 34, 34,
35, 35, 35, 36, 38, 39, 39, 41, 43, 43, 44, 45, 45, 45, 47, 47, 47, 46,
46, 46, 46, 45, 45, 45, 46, 46, 46, 47, 35, 36, 37, 37, 37, 38, 38, 38,
41, 41, 41, 44, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 46, 46, 46, 45,
45, 45, 46, 46, 46, 47, 36, 37, 38, 38, 39, 40, 40, 40, 42, 43, 43, 46,
47, 47, 47, 47, 47, 47, 47, 48, 48, 47, 46, 46, 46, 45, 45, 45, 46, 46,
46, 46, 36, 37, 38, 38, 39, 40, 40, 40, 42, 43, 43, 46, 47, 47, 47, 47,
47, 47, 47, 48, 48, 47, 46, 46, 46, 45, 45, 46, 46, 46, 46, 38, 39,
40, 40, 40, 41, 41, 41, 43, 44, 44, 46, 47, 47, 47, 48, 48, 48, 48, 49,

```

```

49, 48, 48, 48, 47, 47, 47, 47, 47, 47, 47, 48, 41, 42, 42, 42, 42, 42,
42, 43, 44, 45, 45, 46, 47, 47, 48, 48, 48, 49, 50, 50, 50, 50, 49, 49,
49, 49, 49, 49, 50, 50, 50, 50, 41, 42, 42, 42, 42, 42, 43, 44, 45,
45, 46, 47, 47, 48, 48, 48, 49, 50, 50, 50, 50, 49, 49, 49, 49, 49, 49,
50, 50, 50, 50, 43, 43, 43, 43, 43, 43, 43, 43, 45, 45, 45, 46, 47, 47,
48, 49, 49, 49, 50, 51, 51, 50, 50, 50, 50, 50, 50, 50, 50, 50, 51,
47, 47, 46, 46, 46, 45, 45, 46, 46, 47, 47, 47, 47, 47, 48, 50, 50, 50,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 53, 53, 53, 53, 49, 48, 47, 47,
47, 46, 46, 46, 47, 47, 47, 47, 48, 48, 49, 50, 50, 51, 52, 53, 53, 53,
53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 49, 48, 47, 47, 47, 46, 46, 46,
47, 47, 47, 47, 48, 48, 49, 50, 50, 51, 52, 53, 53, 53, 53, 53, 53, 53,
53, 53, 54, 54, 54, 54, 49, 48, 47, 47, 46, 46, 46, 46, 46, 46, 46, 47,
47, 47, 48, 50, 50, 50, 52, 53, 53, 53, 54, 54, 54, 55, 55, 55, 55, 55,
55, 56, 48, 47, 47, 47, 46, 45, 45, 45, 46, 46, 46, 46, 46, 46, 48, 49,
49, 50, 52, 53, 53, 54, 54, 54, 55, 55, 55, 56, 56, 56, 56, 57, 48, 47,
47, 47, 46, 45, 45, 45, 46, 46, 46, 46, 46, 46, 48, 49, 49, 50, 52, 53,
53, 54, 54, 54, 55, 55, 55, 56, 56, 56, 56, 57, 49, 47, 47, 47, 46, 45,
45, 45, 46, 46, 46, 46, 46, 46, 46, 47, 49, 49, 50, 52, 53, 53, 54, 55, 55,
55, 57, 57, 57, 57, 58, 58, 58, 49, 47, 47, 47, 46, 45, 45, 45, 45, 45,
45, 45, 45, 45, 47, 49, 49, 50, 52, 53, 53, 55, 55, 55, 57, 58, 58, 59, 59,
59, 60, 60, 60, 49, 47, 47, 47, 46, 45, 45, 45, 45, 45, 45, 45, 45, 45,
47, 49, 49, 50, 52, 53, 53, 55, 55, 55, 57, 58, 58, 59, 59, 60, 60, 60,
49, 48, 47, 47, 46, 45, 45, 45, 45, 45, 45, 45, 45, 45, 47, 49, 49, 50,
52, 53, 53, 55, 56, 56, 57, 59, 59, 59, 60, 60, 60, 61, 50, 49, 48, 48,
47, 46, 46, 46, 46, 46, 46, 46, 46, 46, 47, 50, 50, 50, 53, 54, 54, 55,
56, 56, 57, 59, 59, 60, 61, 61, 61, 62, 50, 49, 48, 48, 47, 46, 46, 46,
46, 46, 46, 46, 46, 46, 47, 50, 50, 50, 53, 54, 54, 55, 56, 56, 58, 60,
60, 60, 61, 61, 61, 63, 50, 49, 48, 48, 47, 46, 46, 46, 46, 46, 46, 46,
46, 46, 47, 50, 50, 50, 53, 54, 54, 55, 56, 56, 58, 60, 60, 60, 61, 61,
61, 63, 51, 50, 49, 49, 48, 47, 47, 47, 47, 47, 47, 47, 46, 46, 48, 50,
50, 51, 53, 54, 54, 56, 57, 57, 58, 60, 60, 61, 62, 63, 63, 64,
/* Size 4x8 */
31, 38, 47, 48, 31, 40, 46, 45, 35, 43, 47, 46, 39, 47, 47, 45, 43, 47,
50, 50, 47, 47, 53, 55, 46, 46, 53, 58, 48, 46, 54, 59,
/* Size 8x4 */
31, 31, 35, 39, 43, 47, 46, 48, 38, 40, 43, 47, 47, 47, 46, 46, 47, 46,
47, 47, 50, 53, 53, 54, 48, 45, 46, 45, 50, 55, 58, 59,
/* Size 8x16 */
32, 31, 33, 37, 45, 48, 49, 50, 31, 31, 34, 38, 45, 47, 47, 48, 31, 32,
34, 39, 45, 46, 46, 47, 30, 32, 35, 40, 44, 46, 45, 46, 33, 35, 37, 42,
46, 47, 45, 46, 33, 36, 38, 43, 46, 47, 46, 46, 37, 40, 43, 47, 47, 47,
45, 46, 39, 41, 43, 47, 48, 48, 47, 47, 42, 43, 44, 47, 49, 50, 49, 50,
47, 46, 46, 48, 51, 52, 53, 53, 49, 46, 47, 48, 52, 53, 53, 54, 48, 46,
46, 47, 51, 53, 56, 56, 48, 45, 46, 46, 51, 53, 57, 57, 49, 45, 45, 46,
51, 53, 58, 59, 50, 46, 46, 46, 52, 54, 59, 61, 50, 46, 46, 46, 52, 54,
59, 61,
/* Size 16x8 */
32, 31, 31, 30, 33, 33, 37, 39, 42, 47, 49, 48, 48, 49, 50, 50, 31, 31,
32, 32, 35, 36, 40, 41, 43, 46, 46, 46, 45, 45, 46, 46, 33, 34, 34, 35,
37, 38, 43, 43, 44, 46, 47, 46, 46, 45, 46, 46, 37, 38, 39, 40, 42, 43,

```

```

47, 47, 47, 48, 48, 47, 46, 46, 46, 46, 45, 45, 45, 44, 46, 46, 47, 48,
49, 51, 52, 51, 51, 51, 52, 52, 48, 47, 46, 46, 47, 47, 47, 48, 50, 52,
53, 53, 53, 53, 54, 54, 49, 47, 46, 45, 45, 46, 45, 47, 49, 53, 53, 56,
57, 58, 59, 59, 50, 48, 47, 46, 46, 46, 46, 47, 50, 53, 54, 56, 57, 59,
61, 61,
/* Size 16x32 */
32, 31, 31, 31, 33, 37, 37, 38, 45, 48, 48, 49, 49, 49, 50, 52, 31, 31,
31, 31, 33, 38, 38, 39, 45, 47, 47, 48, 48, 48, 49, 51, 31, 31, 31, 31,
34, 38, 38, 40, 45, 47, 47, 47, 47, 47, 48, 50, 31, 31, 31, 31, 34, 38,
38, 40, 45, 47, 47, 47, 47, 47, 48, 50, 31, 31, 32, 32, 34, 39, 39, 40,
45, 46, 46, 46, 46, 46, 47, 49, 30, 31, 32, 32, 35, 40, 40, 41, 44, 46,
46, 45, 45, 45, 46, 48, 30, 31, 32, 32, 35, 40, 40, 41, 44, 46, 46, 45,
45, 45, 46, 48, 31, 32, 33, 33, 35, 40, 40, 41, 45, 46, 46, 45, 45, 45,
46, 48, 33, 34, 35, 35, 37, 42, 42, 43, 46, 47, 47, 46, 45, 45, 46, 47,
33, 35, 36, 36, 38, 43, 43, 44, 46, 47, 47, 46, 46, 46, 46, 47, 35, 37, 38, 38,
36, 36, 38, 43, 43, 44, 46, 47, 47, 46, 46, 46, 46, 47, 35, 37, 38, 38,
41, 45, 45, 46, 47, 47, 47, 46, 45, 45, 46, 47, 37, 39, 40, 40, 43, 47,
47, 47, 47, 47, 47, 46, 45, 45, 46, 47, 37, 39, 40, 40, 43, 47, 47, 47,
47, 47, 47, 46, 45, 45, 46, 47, 39, 40, 41, 41, 43, 47, 47, 47, 48, 48,
48, 47, 47, 47, 47, 48, 42, 42, 43, 43, 44, 47, 47, 48, 49, 50, 50, 49,
49, 49, 50, 50, 42, 42, 43, 43, 44, 47, 47, 48, 49, 50, 50, 49, 49, 49,
50, 50, 43, 43, 43, 43, 45, 47, 47, 48, 50, 50, 50, 50, 50, 50, 50, 51,
47, 46, 46, 46, 46, 48, 48, 48, 51, 52, 52, 52, 53, 53, 53, 53, 49, 47,
46, 46, 47, 48, 48, 49, 52, 53, 53, 53, 53, 53, 54, 54, 49, 47, 46, 46,
47, 48, 48, 49, 52, 53, 53, 53, 53, 53, 54, 54, 48, 47, 46, 46, 46, 47,
47, 48, 52, 53, 53, 54, 55, 55, 55, 56, 48, 47, 46, 46, 46, 47, 47, 48,
51, 53, 53, 54, 56, 56, 56, 57, 48, 47, 46, 46, 46, 47, 47, 48, 51, 53,
53, 54, 56, 56, 56, 57, 48, 47, 45, 45, 46, 46, 46, 47, 51, 53, 53, 55,
57, 57, 57, 59, 49, 46, 45, 45, 45, 46, 46, 47, 51, 53, 53, 56, 58, 58,
59, 61, 49, 46, 45, 45, 45, 46, 46, 47, 51, 53, 53, 56, 58, 58, 59, 61,
49, 47, 45, 45, 45, 46, 46, 47, 52, 53, 53, 56, 58, 58, 60, 62, 50, 48,
46, 46, 46, 46, 46, 48, 52, 54, 54, 57, 59, 59, 61, 63, 50, 48, 46, 46,
46, 46, 46, 48, 52, 54, 54, 57, 59, 59, 61, 64, 50, 48, 46, 46, 46, 46,
46, 48, 52, 54, 54, 57, 59, 59, 61, 64, 51, 49, 47, 47, 47, 47, 47, 48,
52, 54, 54, 58, 60, 60, 62, 65,
/* Size 32x16 */
32, 31, 31, 31, 31, 30, 30, 31, 33, 33, 33, 35, 37, 37, 39, 42, 42, 43,
47, 49, 49, 48, 48, 48, 48, 49, 49, 49, 50, 50, 50, 51, 31, 31, 31, 31,
31, 31, 31, 32, 34, 35, 35, 37, 39, 39, 40, 42, 42, 43, 46, 47, 47, 47,
47, 47, 47, 46, 46, 47, 48, 48, 48, 49, 31, 31, 31, 31, 32, 32, 32, 33,
35, 36, 36, 38, 40, 40, 41, 43, 43, 43, 46, 46, 46, 46, 46, 46, 45, 45,
45, 45, 46, 46, 46, 47, 31, 31, 31, 31, 32, 32, 32, 33, 35, 36, 36, 38,
40, 40, 41, 43, 43, 43, 46, 46, 46, 46, 46, 46, 45, 45, 45, 45, 46, 46,
46, 47, 33, 33, 34, 34, 34, 35, 35, 35, 37, 38, 38, 41, 43, 43, 43, 44,
44, 45, 46, 47, 47, 46, 46, 46, 46, 45, 45, 45, 46, 46, 46, 47, 37, 38,
38, 38, 39, 40, 40, 40, 42, 43, 43, 45, 47, 47, 47, 47, 47, 47, 48, 48,
48, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 47, 37, 38, 38, 38, 39, 40,
40, 40, 42, 43, 43, 45, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47,
46, 46, 46, 46, 46, 46, 46, 47, 38, 39, 40, 40, 40, 41, 41, 41, 43, 44,
44, 46, 47, 47, 47, 48, 48, 48, 48, 49, 49, 48, 48, 48, 47, 47, 47, 47,

```

```

48, 48, 48, 48, 45, 45, 45, 45, 45, 44, 44, 45, 46, 46, 46, 47, 47, 47,
48, 49, 49, 50, 51, 52, 52, 52, 51, 51, 51, 51, 51, 52, 52, 52, 52, 52,
48, 47, 47, 47, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 50, 50, 50,
52, 53, 53, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 48, 47, 47, 47,
46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 50, 50, 50, 52, 53, 53, 53,
53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 49, 48, 47, 47, 46, 45, 45, 45,
46, 46, 46, 46, 46, 46, 47, 49, 49, 50, 52, 53, 53, 54, 54, 54, 55, 56,
56, 56, 57, 57, 57, 58, 49, 48, 47, 47, 46, 45, 45, 45, 45, 46, 46, 45,
45, 45, 47, 49, 49, 50, 53, 53, 53, 55, 56, 56, 57, 58, 58, 58, 59, 59,
59, 60, 49, 48, 47, 47, 46, 45, 45, 45, 45, 46, 46, 45, 45, 45, 47, 49,
49, 50, 53, 53, 53, 55, 56, 56, 57, 58, 58, 58, 59, 59, 59, 60, 50, 49,
48, 48, 47, 46, 46, 46, 46, 46, 46, 46, 46, 47, 50, 50, 50, 53, 54,
54, 55, 56, 56, 57, 59, 59, 60, 61, 61, 61, 62, 52, 51, 50, 50, 49, 48,
48, 48, 47, 47, 47, 47, 47, 47, 48, 50, 50, 51, 53, 54, 54, 56, 57, 57,
59, 61, 61, 62, 63, 64, 64, 65,
/* Size 4x16 */
31, 37, 48, 49, 31, 38, 47, 47, 31, 39, 46, 46, 31, 40, 46, 45, 34, 42,
47, 45, 35, 43, 47, 46, 39, 47, 47, 45, 40, 47, 48, 47, 42, 47, 50, 49,
46, 48, 52, 53, 47, 48, 53, 53, 47, 47, 53, 56, 47, 46, 53, 57, 46, 46,
53, 58, 48, 46, 54, 59, 48, 46, 54, 59,
/* Size 16x4 */
31, 31, 31, 31, 34, 35, 39, 40, 42, 46, 47, 47, 47, 46, 48, 48, 37, 38,
39, 40, 42, 43, 47, 47, 47, 48, 48, 47, 46, 46, 46, 46, 48, 47, 46, 46,
47, 47, 47, 48, 50, 52, 53, 53, 53, 53, 54, 54, 49, 47, 46, 45, 45, 46,
45, 47, 49, 53, 53, 56, 57, 58, 59, 59,
/* Size 8x32 */
32, 31, 33, 37, 45, 48, 49, 50, 31, 31, 33, 38, 45, 47, 48, 49, 31, 31,
34, 38, 45, 47, 47, 48, 31, 31, 34, 38, 45, 47, 47, 48, 31, 32, 34, 39,
45, 46, 46, 47, 30, 32, 35, 40, 44, 46, 45, 46, 30, 32, 35, 40, 44, 46,
45, 46, 31, 33, 35, 40, 45, 46, 45, 46, 33, 35, 37, 42, 46, 47, 45, 46,
33, 36, 38, 43, 46, 47, 46, 46, 33, 36, 38, 43, 46, 47, 46, 46, 35, 38,
41, 45, 47, 47, 45, 46, 37, 40, 43, 47, 47, 47, 45, 46, 37, 40, 43, 47,
47, 47, 45, 46, 39, 41, 43, 47, 48, 48, 47, 47, 42, 43, 44, 47, 49, 50,
49, 50, 42, 43, 44, 47, 49, 50, 49, 50, 43, 43, 45, 47, 50, 50, 50, 50,
47, 46, 46, 48, 51, 52, 53, 53, 49, 46, 47, 48, 52, 53, 53, 54, 49, 46,
47, 48, 52, 53, 53, 54, 48, 46, 46, 47, 52, 53, 55, 55, 48, 46, 46, 47,
51, 53, 56, 56, 48, 46, 46, 47, 51, 53, 56, 56, 48, 45, 46, 46, 51, 53,
57, 57, 49, 45, 45, 46, 51, 53, 58, 59, 49, 45, 45, 46, 51, 53, 58, 59,
49, 45, 45, 46, 52, 53, 58, 60, 50, 46, 46, 46, 52, 54, 59, 61, 50, 46,
46, 46, 52, 54, 59, 61, 50, 46, 46, 46, 52, 54, 59, 61, 51, 47, 47, 47,
52, 54, 60, 62,
/* Size 32x8 */
32, 31, 31, 31, 31, 30, 30, 31, 33, 33, 33, 35, 37, 37, 39, 42, 42, 43,
47, 49, 49, 48, 48, 48, 48, 49, 49, 49, 50, 50, 50, 51, 31, 31, 31, 31,
32, 32, 32, 33, 35, 36, 36, 38, 40, 40, 41, 43, 43, 43, 46, 46, 46, 46,
46, 46, 45, 45, 45, 45, 46, 46, 46, 47, 33, 33, 34, 34, 34, 35, 35, 35,
37, 38, 38, 41, 43, 43, 43, 44, 44, 45, 46, 47, 47, 46, 46, 46, 46, 45,
45, 45, 46, 46, 46, 47, 37, 38, 38, 38, 39, 40, 40, 40, 42, 43, 43, 45,
47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47, 46, 46, 46, 46, 46, 46,
46, 47, 45, 45, 45, 45, 45, 44, 44, 45, 46, 46, 46, 47, 47, 47, 48, 49,

```

```

49, 50, 51, 52, 52, 52, 51, 51, 51, 51, 51, 52, 52, 52, 52, 52, 48, 47,
47, 47, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 50, 50, 50, 52, 53,
53, 53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 49, 48, 47, 47, 46, 45,
45, 45, 45, 46, 46, 45, 45, 45, 47, 49, 49, 50, 53, 53, 53, 55, 56, 56,
57, 58, 58, 58, 59, 59, 59, 60, 50, 49, 48, 48, 47, 46, 46, 46, 46, 46,
46, 46, 46, 46, 47, 50, 50, 50, 53, 54, 54, 55, 56, 56, 57, 59, 59, 60,
61, 61, 61, 62 },
},
{
  /* Luma */
  /* Size 4x4 */
  32, 32, 34, 38, 32, 33, 35, 39, 34, 35, 39, 45, 38, 39, 45, 54,
  /* Size 8x8 */
  31, 31, 32, 32, 33, 34, 37, 41, 31, 32, 32, 32, 33, 34, 36, 39, 32, 32,
  32, 33, 34, 35, 37, 40, 32, 32, 33, 34, 35, 36, 38, 41, 33, 33, 34, 35,
  37, 39, 41, 44, 34, 34, 35, 36, 39, 43, 46, 49, 37, 36, 37, 38, 41, 46,
  51, 54, 41, 39, 40, 41, 44, 49, 54, 58,
  /* Size 16x16 */
  32, 31, 31, 31, 31, 31, 31, 32, 32, 34, 34, 36, 36, 39, 39, 44, 31, 32,
  32, 32, 32, 32, 32, 32, 32, 34, 34, 35, 35, 38, 38, 42, 31, 32, 32, 32,
  32, 32, 32, 32, 32, 34, 34, 35, 35, 38, 38, 42, 31, 32, 32, 32, 32, 32,
  32, 32, 32, 33, 33, 34, 34, 37, 37, 41, 31, 32, 32, 32, 32, 32, 32, 32,
  32, 33, 33, 34, 34, 37, 37, 41, 31, 32, 32, 32, 32, 33, 33, 34, 34, 35,
  35, 36, 36, 39, 39, 42, 31, 32, 32, 32, 32, 33, 33, 34, 34, 35, 35, 36,
  36, 39, 39, 42, 32, 32, 32, 32, 32, 34, 34, 35, 35, 37, 37, 38, 38, 40,
  40, 42, 32, 32, 32, 32, 32, 34, 34, 35, 35, 37, 37, 38, 38, 40, 40, 42,
  34, 34, 34, 33, 33, 35, 35, 37, 37, 39, 39, 42, 42, 45, 45, 47, 34, 34,
  34, 33, 33, 35, 35, 37, 37, 39, 39, 42, 42, 45, 45, 47, 36, 35, 35, 34,
  34, 36, 36, 38, 38, 42, 42, 48, 48, 50, 50, 54, 36, 35, 35, 34, 34, 36,
  36, 38, 38, 42, 42, 48, 48, 50, 50, 54, 39, 38, 38, 37, 37, 39, 39, 40,
  40, 45, 45, 50, 50, 54, 54, 58, 39, 38, 38, 37, 37, 39, 39, 40, 40, 45,
  45, 50, 50, 54, 54, 58, 44, 42, 42, 41, 41, 42, 42, 42, 42, 47, 47, 54,
  54, 58, 58, 63,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 33,
  34, 34, 34, 35, 36, 36, 36, 37, 39, 39, 39, 41, 44, 44, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34,
  35, 35, 35, 37, 39, 39, 39, 41, 43, 43, 31, 31, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 35, 35, 37,
  38, 38, 38, 40, 42, 42, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 35, 35, 37, 38, 38, 38, 40,
  42, 42, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 33, 34, 34, 34, 34, 35, 35, 35, 37, 38, 38, 38, 40, 42, 42, 31, 31,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34,
  34, 34, 35, 35, 35, 36, 38, 38, 38, 39, 41, 41, 31, 31, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34,
  34, 36, 37, 37, 37, 39, 41, 41, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 36, 37, 37,
  37, 39, 41, 41, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 36, 37, 37, 37, 39, 41, 41,

```

```

31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 34,
34, 34, 34, 35, 35, 35, 35, 37, 38, 38, 38, 40, 41, 41, 31, 32, 32, 32,
32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 35, 36,
36, 36, 36, 38, 39, 39, 39, 40, 42, 42, 31, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 35, 36, 36, 36, 36, 38,
39, 39, 39, 40, 42, 42, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 34, 34, 34, 34, 35, 35, 35, 36, 36, 36, 36, 38, 39, 39, 39, 40,
42, 42, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34,
34, 35, 36, 36, 36, 36, 37, 37, 37, 38, 40, 40, 40, 41, 42, 42, 32, 32,
32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 35, 35, 36, 37, 37,
37, 37, 38, 38, 38, 39, 40, 40, 40, 41, 42, 42, 32, 32, 32, 32, 32, 32,
32, 32, 32, 33, 34, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 37, 38, 38,
38, 39, 40, 40, 40, 41, 42, 42, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
34, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 37, 38, 38, 38, 39, 40, 40,
40, 41, 42, 42, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35,
36, 36, 36, 37, 38, 38, 38, 39, 40, 40, 40, 41, 42, 42, 42, 44, 45, 45,
34, 34, 34, 34, 34, 34, 33, 33, 33, 34, 35, 35, 35, 36, 37, 37, 37, 38,
39, 39, 39, 41, 42, 42, 42, 44, 45, 45, 45, 46, 47, 47, 34, 34, 34, 34,
34, 34, 33, 33, 33, 34, 35, 35, 35, 36, 37, 37, 37, 38, 39, 39, 39, 41,
42, 42, 42, 44, 45, 45, 45, 46, 47, 47, 34, 34, 34, 34, 34, 34, 33, 33,
33, 34, 35, 35, 35, 36, 37, 37, 37, 38, 39, 39, 39, 41, 42, 42, 42, 44,
45, 45, 45, 46, 47, 47, 35, 34, 34, 34, 34, 34, 34, 34, 35, 36, 36,
36, 36, 37, 37, 37, 39, 41, 41, 41, 43, 45, 45, 45, 46, 47, 47, 47, 49,
50, 50, 36, 35, 35, 35, 35, 35, 34, 34, 34, 35, 36, 36, 36, 37, 38, 38,
38, 40, 42, 42, 42, 45, 48, 48, 48, 49, 50, 50, 50, 52, 54, 54, 36, 35,
35, 35, 35, 34, 34, 34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42,
42, 45, 48, 48, 48, 49, 50, 50, 50, 52, 54, 54, 36, 35, 35, 35, 35, 35,
34, 34, 34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42, 42, 45, 48, 48,
48, 49, 50, 50, 50, 52, 54, 54, 37, 37, 37, 37, 37, 36, 36, 36, 36, 37,
38, 38, 38, 38, 39, 39, 39, 41, 44, 44, 44, 46, 49, 49, 49, 51, 52, 52,
52, 54, 56, 56, 39, 39, 38, 38, 38, 38, 37, 37, 37, 38, 39, 39, 39, 40,
40, 40, 40, 42, 45, 45, 45, 47, 50, 50, 50, 52, 54, 54, 54, 56, 58, 58,
39, 39, 38, 38, 38, 38, 37, 37, 37, 38, 39, 39, 39, 40, 40, 40, 40, 42,
45, 45, 45, 47, 50, 50, 50, 52, 54, 54, 54, 56, 58, 58, 39, 39, 38, 38,
38, 38, 37, 37, 37, 38, 39, 39, 39, 40, 40, 40, 40, 42, 45, 45, 45, 47,
50, 50, 50, 52, 54, 54, 54, 56, 58, 58, 41, 41, 40, 40, 40, 39, 39, 39,
39, 40, 40, 40, 40, 41, 41, 41, 41, 44, 46, 46, 46, 49, 52, 52, 52, 54,
56, 56, 56, 58, 60, 60, 44, 43, 42, 42, 42, 41, 41, 41, 41, 41, 42, 42,
42, 42, 42, 42, 42, 42, 42, 45, 47, 47, 47, 50, 54, 54, 54, 56, 58, 58,
58, 60, 63, 63, 44, 43, 42, 42, 42, 41, 41, 41, 41, 41, 42, 42, 42, 42, 42, 42,
42, 45, 47, 47, 47, 50, 54, 54, 54, 56, 58, 58, 58, 60, 63, 63,
/* Size 4x8 */
31, 32, 34, 39, 32, 32, 34, 38, 32, 33, 34, 38, 32, 33, 36, 40, 33, 34,
38, 42, 34, 36, 41, 47, 37, 38, 44, 52, 40, 40, 46, 56,
/* Size 8x4 */
31, 32, 32, 32, 33, 34, 37, 40, 32, 32, 33, 33, 34, 36, 38, 40, 34, 34,
34, 36, 38, 41, 44, 46, 39, 38, 38, 40, 42, 47, 52, 56,
/* Size 8x16 */
32, 31, 31, 32, 32, 36, 36, 44, 31, 32, 32, 32, 32, 35, 35, 42, 31, 32,
32, 32, 32, 35, 35, 42, 31, 32, 32, 33, 33, 34, 34, 41, 31, 32, 32, 33,

```

```

33, 34, 34, 41, 32, 32, 32, 34, 34, 36, 36, 42, 32, 32, 32, 34, 34, 36,
36, 42, 32, 33, 33, 35, 35, 38, 38, 42, 32, 33, 33, 35, 35, 38, 38, 42,
34, 34, 34, 37, 37, 42, 42, 48, 34, 34, 34, 37, 37, 42, 42, 48, 36, 34,
34, 38, 38, 48, 48, 54, 36, 34, 34, 38, 38, 48, 48, 54, 39, 37, 37, 40,
40, 50, 50, 58, 39, 37, 37, 40, 40, 50, 50, 58, 44, 41, 41, 43, 43, 53,
53, 63,
/* Size 16x8 */
32, 31, 31, 31, 31, 32, 32, 32, 32, 34, 34, 36, 36, 39, 39, 44, 31, 32,
32, 32, 32, 32, 32, 33, 33, 34, 34, 34, 34, 37, 37, 41, 31, 32, 32, 32,
32, 32, 32, 33, 33, 34, 34, 34, 34, 37, 37, 41, 32, 32, 32, 33, 33, 34,
34, 35, 35, 37, 37, 38, 38, 40, 40, 43, 32, 32, 32, 33, 33, 34, 34, 35,
35, 37, 37, 38, 38, 40, 40, 43, 36, 35, 35, 34, 34, 36, 36, 38, 38, 42,
42, 48, 48, 50, 50, 53, 36, 35, 35, 34, 34, 36, 36, 38, 38, 42, 42, 48,
48, 50, 50, 53, 44, 42, 42, 41, 41, 42, 42, 42, 42, 48, 48, 54, 54, 58,
58, 63,
/* Size 16x32 */
32, 31, 31, 31, 31, 32, 32, 32, 32, 34, 36, 36, 36, 39, 44, 44, 31, 31,
31, 31, 31, 32, 32, 32, 32, 34, 35, 35, 35, 39, 43, 43, 31, 32, 32, 32,
32, 32, 32, 32, 32, 34, 35, 35, 35, 38, 42, 42, 31, 32, 32, 32, 32, 32,
32, 32, 34, 35, 35, 35, 38, 42, 42, 31, 32, 32, 32, 32, 32, 32, 32,
32, 34, 35, 35, 35, 38, 42, 42, 31, 32, 32, 32, 32, 32, 32, 32, 32, 34,
35, 35, 35, 38, 41, 41, 31, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34,
34, 37, 41, 41, 31, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 37,
41, 41, 31, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 37, 41, 41,
31, 32, 32, 32, 32, 33, 33, 33, 33, 34, 35, 35, 35, 38, 41, 41, 32, 32,
32, 32, 32, 33, 34, 34, 34, 35, 36, 36, 36, 39, 42, 42, 32, 32, 32, 32,
32, 33, 34, 34, 34, 35, 36, 36, 36, 39, 42, 42, 32, 32, 32, 32, 32, 33,
34, 34, 34, 34, 35, 36, 36, 36, 39, 42, 42, 32, 32, 32, 32, 32, 33, 34,
34, 34, 36, 37, 37, 37, 40, 42, 42, 32, 32, 33, 33, 33, 34, 35, 35, 35,
37, 38, 38, 38, 40, 42, 42, 32, 32, 33, 33, 33, 34, 35, 35, 35, 37, 38,
38, 38, 40, 42, 42, 32, 32, 33, 33, 33, 34, 35, 35, 35, 37, 38, 38, 38,
40, 42, 42, 33, 33, 33, 33, 33, 34, 36, 36, 36, 38, 40, 40, 40, 42, 45,
45, 34, 34, 34, 34, 34, 35, 37, 37, 37, 39, 42, 42, 42, 45, 48, 48, 34,
34, 34, 34, 34, 35, 37, 37, 37, 39, 42, 42, 42, 45, 48, 48, 34, 34, 34,
34, 34, 35, 37, 37, 37, 39, 42, 42, 42, 45, 48, 48, 35, 34, 34, 34, 34,
36, 37, 37, 37, 41, 45, 45, 45, 47, 50, 50, 36, 35, 34, 34, 34, 36, 38,
38, 38, 43, 48, 48, 48, 51, 54, 54, 36, 35, 34, 34, 34, 36, 38, 38, 38,
43, 48, 48, 48, 51, 54, 54, 36, 35, 34, 34, 34, 36, 38, 38, 38, 43, 48,
48, 48, 51, 54, 54, 37, 37, 36, 36, 36, 38, 39, 39, 39, 44, 49, 49, 49,
52, 56, 56, 39, 38, 37, 37, 37, 39, 40, 40, 40, 45, 50, 50, 50, 54, 58,
58, 39, 38, 37, 37, 37, 39, 40, 40, 40, 45, 50, 50, 50, 54, 58, 58, 39,
38, 37, 37, 37, 39, 40, 40, 40, 45, 50, 50, 50, 54, 58, 58, 41, 40, 39,
39, 39, 40, 42, 42, 42, 46, 52, 52, 52, 56, 60, 60, 44, 42, 41, 41, 41,
42, 43, 43, 43, 48, 53, 53, 53, 58, 63, 63, 44, 42, 41, 41, 41, 42, 43,
43, 43, 48, 53, 53, 53, 58, 63, 63,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32,
33, 34, 34, 34, 35, 36, 36, 36, 37, 39, 39, 39, 41, 44, 44, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 34,
35, 35, 35, 37, 38, 38, 38, 40, 42, 42, 31, 31, 32, 32, 32, 32, 32, 32,

```

```

32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 36,
37, 37, 37, 39, 41, 41, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 36, 37, 37, 37, 39,
41, 41, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33,
33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 36, 37, 37, 37, 39, 41, 41, 32, 32,
32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35,
35, 36, 36, 36, 36, 38, 39, 39, 39, 40, 42, 42, 32, 32, 32, 32, 32, 32,
33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 37, 38, 38,
38, 39, 40, 40, 40, 42, 43, 43, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33,
34, 34, 34, 34, 35, 35, 35, 36, 37, 37, 37, 37, 38, 38, 38, 39, 40, 40,
40, 42, 43, 43, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34,
35, 35, 35, 36, 37, 37, 37, 37, 38, 38, 38, 39, 40, 40, 40, 42, 43, 43,
34, 34, 34, 34, 34, 34, 33, 33, 33, 34, 35, 35, 35, 36, 37, 37, 37, 38,
39, 39, 39, 41, 43, 43, 43, 44, 45, 45, 45, 46, 48, 48, 36, 35, 35, 35,
35, 35, 34, 34, 34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42, 42, 45,
48, 48, 48, 49, 50, 50, 50, 52, 53, 53, 36, 35, 35, 35, 35, 35, 34, 34,
34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42, 42, 45, 48, 48, 48, 49,
50, 50, 50, 52, 53, 53, 36, 35, 35, 35, 35, 35, 34, 34, 34, 35, 36, 36,
36, 37, 38, 38, 38, 40, 42, 42, 42, 45, 48, 48, 48, 49, 50, 50, 50, 52,
53, 53, 39, 39, 38, 38, 38, 38, 37, 37, 37, 38, 39, 39, 39, 40, 40, 40,
40, 42, 45, 45, 45, 47, 51, 51, 51, 52, 54, 54, 54, 56, 58, 58, 44, 43,
42, 42, 42, 41, 41, 41, 41, 41, 42, 42, 42, 42, 42, 42, 45, 48, 48,
48, 50, 54, 54, 54, 56, 58, 58, 58, 60, 63, 63, 44, 43, 42, 42, 42, 41,
41, 41, 41, 42, 42, 42, 42, 42, 42, 42, 45, 48, 48, 48, 50, 54, 54,
54, 56, 58, 58, 58, 60, 63, 63,
/* Size 4x16 */
31, 32, 34, 39, 32, 32, 34, 38, 32, 32, 34, 38, 32, 32, 33, 37, 32, 32,
33, 37, 32, 33, 35, 39, 32, 33, 35, 39, 32, 34, 37, 40, 32, 34, 37, 40,
34, 35, 39, 45, 34, 35, 39, 45, 35, 36, 43, 51, 35, 36, 43, 51, 38, 39,
45, 54, 38, 39, 45, 54, 42, 42, 48, 58,
/* Size 16x4 */
31, 32, 32, 32, 32, 32, 32, 32, 32, 34, 34, 35, 35, 38, 38, 42, 32, 32,
32, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 39, 39, 42, 34, 34, 34, 33,
33, 35, 35, 37, 37, 39, 39, 43, 43, 45, 45, 48, 39, 38, 38, 37, 37, 39,
39, 40, 40, 45, 45, 51, 51, 54, 54, 58,
/* Size 8x32 */
32, 31, 31, 32, 32, 36, 36, 44, 31, 31, 31, 32, 32, 35, 35, 43, 31, 32,
32, 32, 32, 35, 35, 42, 31, 32, 32, 32, 32, 35, 35, 42, 31, 32, 32, 32,
32, 35, 35, 42, 31, 32, 32, 32, 32, 35, 35, 41, 31, 32, 32, 33, 33, 34,
34, 41, 31, 32, 32, 33, 33, 34, 34, 41, 31, 32, 32, 33, 33, 34, 34, 41,
31, 32, 32, 33, 33, 35, 35, 41, 32, 32, 32, 34, 34, 36, 36, 42, 32, 32,
32, 34, 34, 36, 36, 42, 32, 32, 32, 34, 34, 36, 36, 42, 32, 32, 32, 34,
34, 37, 37, 42, 32, 33, 33, 35, 35, 38, 38, 42, 32, 33, 33, 35, 35, 38,
38, 42, 32, 33, 33, 35, 35, 38, 38, 42, 33, 33, 33, 36, 36, 40, 40, 45,
34, 34, 34, 37, 37, 42, 42, 48, 34, 34, 34, 37, 37, 42, 42, 48, 34, 34,
34, 37, 37, 42, 42, 48, 35, 34, 34, 37, 37, 45, 45, 50, 36, 34, 34, 38,
38, 48, 48, 54, 36, 34, 34, 38, 38, 48, 48, 54, 36, 34, 34, 38, 38, 48,
48, 54, 37, 36, 36, 39, 39, 49, 49, 56, 39, 37, 37, 40, 40, 50, 50, 58,
39, 37, 37, 40, 40, 50, 50, 58, 39, 37, 37, 40, 40, 50, 50, 58, 41, 39,
39, 42, 42, 52, 52, 60, 44, 41, 41, 43, 43, 53, 53, 63, 44, 41, 41, 43,

```



```

43, 53, 53, 63,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 33,
34, 34, 34, 35, 36, 36, 36, 37, 39, 39, 39, 41, 44, 44, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34,
34, 34, 34, 36, 37, 37, 37, 39, 41, 41, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 36,
37, 37, 37, 39, 41, 41, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34,
34, 34, 35, 35, 35, 36, 37, 37, 37, 37, 38, 38, 38, 39, 40, 40, 40, 42,
43, 43, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35,
35, 36, 37, 37, 37, 37, 38, 38, 38, 39, 40, 40, 40, 42, 43, 43, 36, 35,
35, 35, 35, 34, 34, 34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42,
42, 45, 48, 48, 48, 49, 50, 50, 50, 52, 53, 53, 36, 35, 35, 35, 35, 35,
34, 34, 34, 35, 36, 36, 36, 37, 38, 38, 38, 40, 42, 42, 42, 45, 48, 48,
48, 49, 50, 50, 50, 52, 53, 53, 44, 43, 42, 42, 42, 41, 41, 41, 41, 41,
42, 42, 42, 42, 42, 42, 42, 45, 48, 48, 48, 50, 54, 54, 54, 56, 58, 58,
58, 60, 63, 63 },
{ /* Chroma */
/* Size 4x4 */
31, 34, 42, 47, 34, 39, 45, 46, 42, 45, 48, 49, 47, 46, 49, 54,
/* Size 8x8 */
31, 31, 32, 35, 39, 45, 48, 48, 31, 31, 33, 37, 41, 44, 46, 46, 32, 33,
35, 39, 42, 45, 46, 45, 35, 37, 39, 43, 45, 47, 47, 46, 39, 41, 42, 45,
47, 48, 48, 47, 45, 44, 45, 47, 48, 50, 51, 51, 48, 46, 46, 47, 48, 51,
53, 54, 48, 46, 45, 46, 47, 51, 54, 56,
/* Size 16x16 */
32, 31, 31, 30, 30, 33, 33, 36, 36, 41, 41, 49, 49, 48, 48, 49, 31, 31,
31, 31, 31, 34, 34, 38, 38, 42, 42, 47, 47, 47, 47, 47, 31, 31, 31, 31,
31, 34, 34, 38, 38, 42, 42, 47, 47, 47, 47, 47, 30, 31, 31, 32, 32, 35,
35, 40, 40, 42, 42, 46, 46, 45, 45, 45, 30, 31, 31, 32, 32, 35, 35, 40,
40, 42, 42, 46, 46, 45, 45, 45, 33, 34, 34, 35, 35, 39, 39, 43, 43, 45,
45, 47, 47, 46, 46, 45, 33, 34, 34, 35, 35, 39, 39, 43, 43, 45, 45, 47,
47, 46, 46, 45, 36, 38, 38, 40, 40, 43, 43, 47, 47, 47, 47, 47, 48, 48, 46,
46, 45, 36, 38, 38, 40, 40, 43, 43, 47, 47, 47, 47, 48, 48, 46, 46, 45,
41, 42, 42, 42, 42, 45, 45, 47, 47, 48, 48, 50, 50, 49, 49, 49, 41, 42,
42, 42, 42, 45, 45, 47, 47, 48, 48, 50, 50, 49, 49, 49, 49, 47, 47, 46,
46, 47, 47, 48, 48, 50, 50, 53, 53, 53, 53, 53, 49, 47, 47, 46, 46, 47,
47, 48, 48, 50, 50, 53, 53, 53, 53, 53, 48, 47, 47, 45, 45, 46, 46, 46,
46, 49, 49, 53, 53, 54, 54, 55, 48, 47, 47, 45, 45, 46, 46, 46, 46, 49,
49, 53, 53, 54, 54, 55, 49, 47, 47, 45, 45, 45, 45, 45, 45, 49, 49, 53,
53, 55, 55, 58,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 30, 30, 30, 32, 33, 33, 33, 35, 36, 36, 36, 39,
41, 41, 41, 45, 49, 49, 49, 49, 48, 48, 48, 49, 49, 49, 31, 31, 31, 31,
31, 31, 31, 31, 31, 32, 34, 34, 34, 35, 37, 37, 37, 39, 42, 42, 42, 45,
48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 31, 31, 31, 31, 31, 31, 31,
31, 33, 34, 34, 34, 36, 38, 38, 38, 40, 42, 42, 42, 45, 47, 47, 47, 47,
47, 47, 47, 47, 47, 47, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 34,
34, 36, 38, 38, 38, 40, 42, 42, 42, 45, 47, 47, 47, 47, 47, 47, 47,
47, 47, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 34, 36, 38, 38,

```

```

38, 40, 42, 42, 42, 45, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 31, 31,
31, 31, 31, 31, 31, 31, 31, 33, 35, 35, 35, 35, 37, 39, 39, 39, 41, 42, 42,
42, 44, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 30, 31, 31, 31, 31, 31,
32, 32, 32, 33, 35, 35, 35, 37, 40, 40, 40, 41, 42, 42, 42, 44, 46, 46,
46, 46, 45, 45, 45, 45, 45, 45, 30, 31, 31, 31, 31, 31, 32, 32, 32, 33,
35, 35, 35, 37, 40, 40, 40, 41, 42, 42, 42, 44, 46, 46, 46, 46, 45, 45,
45, 45, 45, 45, 30, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 35, 35, 35, 37,
40, 40, 40, 41, 42, 42, 42, 44, 46, 46, 46, 46, 45, 45, 45, 45, 45, 45,
32, 32, 33, 33, 33, 33, 33, 33, 33, 35, 37, 37, 37, 39, 41, 41, 41, 42,
43, 43, 43, 45, 47, 47, 47, 46, 46, 46, 46, 45, 45, 45, 33, 34, 34, 34,
34, 35, 35, 35, 35, 37, 39, 39, 39, 41, 43, 43, 43, 44, 45, 45, 45, 46,
47, 47, 47, 47, 46, 46, 46, 46, 45, 45, 33, 34, 34, 34, 34, 35, 35, 35,
35, 37, 39, 39, 39, 41, 43, 43, 43, 44, 45, 45, 45, 46, 47, 47, 47, 47,
46, 46, 46, 46, 45, 45, 33, 34, 34, 34, 34, 35, 35, 35, 35, 37, 39, 39,
39, 41, 43, 43, 43, 44, 45, 45, 45, 46, 47, 47, 47, 47, 46, 46, 46, 46,
45, 45, 35, 35, 36, 36, 36, 37, 37, 37, 37, 39, 41, 41, 41, 43, 45, 45,
45, 45, 46, 46, 46, 47, 47, 47, 47, 47, 47, 46, 46, 46, 46, 45, 45, 36, 37,
38, 38, 38, 39, 40, 40, 40, 41, 43, 43, 43, 45, 47, 47, 47, 47, 47, 47,
47, 47, 48, 48, 48, 47, 46, 46, 46, 46, 45, 45, 36, 37, 38, 38, 38, 39,
40, 40, 40, 41, 43, 43, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48,
48, 47, 46, 46, 46, 46, 45, 45, 36, 37, 38, 38, 38, 39, 40, 40, 40, 41,
43, 43, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 46, 46,
46, 46, 45, 45, 39, 39, 40, 40, 40, 41, 41, 41, 41, 42, 44, 44, 44, 45,
47, 47, 47, 47, 48, 48, 48, 48, 49, 49, 49, 48, 48, 48, 48, 47, 47, 47,
41, 42, 42, 42, 42, 42, 42, 42, 42, 43, 45, 45, 45, 46, 47, 47, 47, 48,
48, 48, 48, 49, 50, 50, 50, 50, 49, 49, 49, 49, 49, 49, 41, 42, 42, 42,
42, 42, 42, 42, 42, 43, 45, 45, 45, 46, 47, 47, 47, 48, 48, 48, 48, 49,
50, 50, 50, 50, 49, 49, 49, 49, 49, 49, 41, 42, 42, 42, 42, 42, 42, 42,
42, 43, 45, 45, 45, 46, 47, 47, 47, 48, 48, 48, 48, 49, 50, 50, 50, 50,
49, 49, 49, 49, 49, 49, 45, 45, 45, 45, 45, 44, 44, 44, 44, 45, 46, 46,
46, 47, 47, 47, 47, 48, 49, 49, 49, 50, 51, 51, 51, 51, 51, 51, 51, 51,
51, 51, 49, 48, 47, 47, 47, 47, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 48,
48, 49, 50, 50, 50, 51, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 49, 48,
47, 47, 47, 47, 46, 46, 46, 47, 47, 47, 47, 47, 48, 48, 48, 49, 50, 50,
50, 51, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 49, 48, 47, 47, 47, 47,
46, 46, 46, 47, 47, 47, 47, 47, 48, 48, 48, 49, 50, 50, 50, 51, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 49, 48, 47, 47, 47, 46, 46, 46, 46, 46,
47, 47, 47, 47, 47, 47, 47, 48, 50, 50, 50, 51, 53, 53, 53, 53, 53,
53, 54, 54, 54, 48, 48, 47, 47, 47, 46, 45, 45, 45, 46, 46, 46, 46, 46,
46, 46, 46, 48, 49, 49, 49, 51, 53, 53, 53, 53, 54, 54, 54, 55, 55, 55,
48, 48, 47, 47, 47, 46, 45, 45, 45, 46, 46, 46, 46, 46, 46, 46, 46, 48,
49, 49, 49, 51, 53, 53, 53, 53, 54, 54, 54, 55, 55, 55, 48, 48, 47, 47,
47, 46, 45, 45, 45, 46, 46, 46, 46, 46, 46, 46, 46, 48, 49, 49, 49, 51,
53, 53, 53, 53, 54, 54, 54, 55, 55, 55, 49, 48, 47, 47, 47, 46, 45, 45,
45, 45, 45, 45, 45, 46, 46, 46, 46, 46, 46, 47, 49, 49, 49, 51, 53, 53,
53, 54, 55, 55, 55, 57, 58, 49, 48, 47, 47, 47, 46, 45, 45, 45, 45, 45,
45, 45, 45, 45, 45, 47, 49, 49, 49, 51, 53, 53, 53, 54, 55, 55, 55, 57,
58, 58, 49, 48, 47, 47, 47, 46, 45, 45, 45, 45, 45, 45, 45, 45, 45,
45, 47, 49, 49, 49, 51, 53, 53, 53, 54, 55, 55, 55, 57, 58, 58,

```

```
/* Size 4x8 */
```

```

31, 34, 42, 48, 31, 35, 42, 46, 33, 37, 44, 46, 36, 41, 46, 46, 40, 44,
48, 48, 45, 46, 49, 51, 47, 47, 50, 54, 47, 46, 49, 55,
/* Size 8x4 */
31, 31, 33, 36, 40, 45, 47, 47, 34, 35, 37, 41, 44, 46, 47, 46, 42, 42,
44, 46, 48, 49, 50, 49, 48, 46, 46, 46, 48, 51, 54, 55,
/* Size 8x16 */
32, 31, 31, 37, 37, 48, 48, 49, 31, 31, 31, 38, 38, 47, 47, 47, 31, 31,
31, 38, 38, 47, 47, 47, 30, 32, 32, 40, 40, 46, 46, 45, 30, 32, 32, 40,
40, 46, 46, 45, 33, 36, 36, 43, 43, 47, 47, 46, 33, 36, 36, 43, 43, 47,
47, 46, 37, 40, 40, 47, 47, 47, 47, 45, 37, 40, 40, 47, 47, 47, 47, 45,
42, 43, 43, 47, 47, 50, 50, 49, 42, 43, 43, 47, 47, 50, 50, 49, 49, 46,
46, 48, 48, 53, 53, 53, 49, 46, 46, 48, 48, 53, 53, 53, 48, 46, 46, 47,
47, 53, 53, 56, 48, 46, 46, 47, 47, 53, 53, 56, 49, 45, 45, 46, 46, 53,
53, 58,
/* Size 16x8 */
32, 31, 31, 30, 30, 33, 33, 37, 37, 42, 42, 49, 49, 48, 48, 49, 31, 31,
31, 32, 32, 36, 36, 40, 40, 43, 43, 46, 46, 46, 46, 45, 31, 31, 31, 32,
32, 36, 36, 40, 40, 43, 43, 46, 46, 46, 46, 45, 37, 38, 38, 40, 40, 43,
43, 47, 47, 47, 47, 48, 48, 47, 47, 46, 37, 38, 38, 40, 40, 43, 43, 47,
47, 47, 47, 48, 48, 47, 47, 46, 48, 47, 47, 46, 46, 47, 47, 47, 47, 50,
50, 53, 53, 53, 53, 53, 48, 47, 47, 46, 46, 47, 47, 47, 47, 50, 50, 53,
53, 53, 53, 53, 49, 47, 47, 45, 45, 46, 46, 45, 45, 49, 49, 53, 53, 56,
56, 58,
/* Size 16x32 */
32, 31, 31, 31, 31, 33, 37, 37, 37, 42, 48, 48, 48, 48, 49, 49, 31, 31,
31, 31, 31, 34, 37, 37, 37, 42, 47, 47, 47, 48, 48, 48, 31, 31, 31, 31,
31, 34, 38, 38, 38, 42, 47, 47, 47, 47, 47, 47, 31, 31, 31, 31, 31, 34,
38, 38, 38, 42, 47, 47, 47, 47, 47, 47, 31, 31, 31, 31, 31, 34, 38, 38,
38, 42, 47, 47, 47, 47, 47, 47, 31, 31, 32, 32, 32, 35, 39, 39, 39, 42,
46, 46, 46, 46, 46, 46, 30, 31, 32, 32, 32, 35, 40, 40, 40, 42, 46, 46,
46, 45, 45, 45, 30, 31, 32, 32, 32, 35, 40, 40, 40, 42, 46, 46, 46, 45,
45, 45, 30, 31, 32, 32, 32, 35, 40, 40, 40, 42, 46, 46, 46, 45, 45, 45,
32, 33, 34, 34, 34, 37, 41, 41, 41, 44, 46, 46, 46, 46, 45, 45, 33, 34,
36, 36, 36, 39, 43, 43, 43, 45, 47, 47, 47, 46, 46, 46, 33, 34, 36, 36,
36, 39, 43, 43, 43, 45, 47, 47, 47, 46, 46, 46, 33, 34, 36, 36, 36, 39,
43, 43, 43, 45, 47, 47, 47, 46, 46, 46, 35, 36, 38, 38, 38, 41, 45, 45,
45, 46, 47, 47, 47, 46, 45, 45, 37, 38, 40, 40, 40, 43, 47, 47, 47, 47,
47, 47, 47, 46, 45, 45, 37, 38, 40, 40, 40, 43, 47, 47, 47, 47, 47,
47, 46, 45, 45, 37, 38, 40, 40, 40, 43, 47, 47, 47, 47, 47, 47, 47, 46,
45, 45, 39, 40, 41, 41, 41, 44, 47, 47, 47, 48, 49, 49, 49, 48, 47, 47,
42, 42, 43, 43, 43, 45, 47, 47, 47, 48, 50, 50, 50, 50, 49, 49, 42, 42,
43, 43, 43, 45, 47, 47, 47, 48, 50, 50, 50, 50, 49, 49, 42, 42, 43, 43,
43, 45, 47, 47, 47, 48, 50, 50, 50, 50, 49, 49, 45, 45, 44, 44, 44, 46,
47, 47, 47, 49, 51, 51, 51, 51, 51, 51, 49, 48, 46, 46, 46, 47, 48, 48,
48, 50, 48, 50, 53, 53, 53, 53, 53, 53, 49, 48, 46, 46, 46, 47, 48, 48,
50, 53, 53, 53, 53, 53, 48, 47, 46, 46, 46, 47, 47, 47, 47, 50, 53, 53,
54, 54, 54, 48, 47, 46, 46, 46, 46, 47, 47, 47, 50, 53, 53, 54, 56, 56,
48, 47, 46, 46, 46, 46, 47, 47, 50, 53, 53, 53, 54, 56, 56, 48, 47,
46, 46, 46, 46, 47, 47, 50, 53, 53, 54, 56, 56, 48, 47, 45, 45,

```

```

45, 46, 46, 46, 46, 49, 53, 53, 53, 55, 57, 57, 49, 47, 45, 45, 45, 45,
46, 46, 46, 49, 53, 53, 53, 56, 58, 58, 49, 47, 45, 45, 45, 45, 46, 46,
46, 49, 53, 53, 53, 56, 58, 58,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 30, 30, 30, 32, 33, 33, 33, 35, 37, 37, 37, 39,
42, 42, 42, 45, 49, 49, 49, 48, 48, 48, 48, 48, 49, 49, 31, 31, 31, 31,
31, 31, 31, 31, 31, 33, 34, 34, 34, 36, 38, 38, 38, 40, 42, 42, 42, 45,
48, 48, 48, 47, 47, 47, 47, 47, 47, 47, 31, 31, 31, 31, 31, 32, 32, 32,
32, 34, 36, 36, 36, 38, 40, 40, 40, 41, 43, 43, 43, 44, 46, 46, 46, 46,
46, 46, 46, 45, 45, 45, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 36, 36,
36, 38, 40, 40, 40, 41, 43, 43, 43, 44, 46, 46, 46, 46, 46, 46, 46, 45,
45, 45, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 36, 36, 36, 38, 40, 40,
40, 41, 43, 43, 43, 44, 46, 46, 46, 46, 46, 46, 46, 45, 45, 45, 33, 34,
34, 34, 34, 35, 35, 35, 35, 37, 39, 39, 39, 41, 43, 43, 43, 44, 45, 45,
45, 46, 47, 47, 47, 47, 46, 46, 46, 46, 45, 45, 37, 37, 38, 38, 38, 39,
40, 40, 40, 41, 43, 43, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48,
48, 47, 47, 47, 47, 46, 46, 46, 37, 37, 38, 38, 38, 39, 40, 40, 40, 41,
43, 43, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47,
47, 46, 46, 46, 37, 37, 38, 38, 38, 39, 40, 40, 40, 41, 43, 43, 43, 45,
47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47, 47, 46, 46, 46,
42, 42, 42, 42, 42, 42, 42, 42, 42, 44, 45, 45, 45, 46, 47, 47, 47, 48,
48, 48, 48, 49, 50, 50, 50, 50, 50, 50, 50, 49, 49, 49, 48, 47, 47, 47,
47, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 49, 50, 50, 50, 51,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 48, 47, 47, 47, 47, 46, 46, 46,
46, 46, 47, 47, 47, 47, 47, 47, 47, 49, 50, 50, 50, 51, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 48, 47, 47, 47, 47, 46, 46, 46, 46, 46, 47, 47,
47, 47, 47, 47, 47, 49, 50, 50, 50, 51, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 48, 48, 47, 47, 47, 46, 45, 45, 45, 46, 46, 46, 46, 46, 46, 46,
46, 48, 50, 50, 50, 51, 53, 53, 53, 54, 54, 54, 54, 55, 56, 56, 49, 48,
47, 47, 47, 46, 45, 45, 45, 45, 46, 46, 46, 45, 45, 45, 45, 47, 49, 49,
49, 51, 53, 53, 53, 54, 56, 56, 56, 57, 58, 58, 49, 48, 47, 47, 47, 46,
45, 45, 45, 45, 46, 46, 46, 45, 45, 45, 45, 47, 49, 49, 49, 51, 53, 53,
53, 54, 56, 56, 56, 57, 58, 58,
/* Size 4x16 */
31, 33, 42, 48, 31, 34, 42, 47, 31, 34, 42, 47, 31, 35, 42, 45, 31, 35,
42, 45, 34, 39, 45, 46, 34, 39, 45, 46, 38, 43, 47, 46, 38, 43, 47, 46,
42, 45, 48, 50, 42, 45, 48, 50, 48, 47, 50, 53, 48, 47, 50, 53, 47, 46,
50, 54, 47, 46, 50, 54, 47, 45, 49, 56,
/* Size 16x4 */
31, 31, 31, 31, 31, 34, 34, 38, 38, 42, 42, 48, 48, 47, 47, 47, 33, 34,
34, 35, 35, 39, 39, 43, 43, 45, 45, 47, 47, 46, 46, 45, 42, 42, 42, 42,
42, 45, 45, 47, 47, 48, 48, 50, 50, 50, 50, 49, 48, 47, 47, 45, 45, 46,
46, 46, 46, 50, 50, 53, 53, 54, 54, 56,
/* Size 8x32 */
32, 31, 31, 37, 37, 48, 48, 49, 31, 31, 31, 37, 37, 47, 47, 48, 31, 31,
31, 38, 38, 47, 47, 47, 31, 31, 31, 38, 38, 47, 47, 47, 31, 31, 31, 38,
38, 47, 47, 47, 31, 32, 32, 39, 39, 46, 46, 46, 30, 32, 32, 40, 40, 46,
46, 45, 30, 32, 32, 40, 40, 46, 46, 45, 30, 32, 32, 40, 40, 46, 46, 45,
32, 34, 34, 41, 41, 46, 46, 45, 33, 36, 36, 43, 43, 47, 47, 46, 33, 36,
36, 43, 43, 47, 47, 46, 33, 36, 36, 43, 43, 47, 47, 46, 35, 38, 38, 45,

```

```

45, 47, 47, 45, 37, 40, 40, 47, 47, 47, 47, 45, 37, 40, 40, 47, 47, 47,
47, 45, 37, 40, 40, 47, 47, 47, 47, 45, 39, 41, 41, 47, 47, 49, 49, 47,
42, 43, 43, 47, 47, 50, 50, 49, 42, 43, 43, 47, 47, 50, 50, 49, 42, 43,
43, 47, 47, 50, 50, 49, 45, 44, 44, 47, 47, 51, 51, 51, 49, 46, 46, 48,
48, 53, 53, 53, 49, 46, 46, 48, 48, 53, 53, 53, 49, 46, 46, 48, 48, 53,
53, 53, 48, 46, 46, 47, 47, 53, 53, 54, 48, 46, 46, 47, 47, 53, 53, 56,
48, 46, 46, 47, 47, 53, 53, 56, 48, 46, 46, 47, 47, 53, 53, 56, 48, 45,
45, 46, 46, 53, 53, 57, 49, 45, 45, 46, 46, 53, 53, 58, 49, 45, 45, 46,
46, 53, 53, 58,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 30, 30, 30, 32, 33, 33, 33, 35, 37, 37, 37, 39,
42, 42, 42, 45, 49, 49, 49, 48, 48, 48, 48, 48, 49, 49, 31, 31, 31, 31,
31, 32, 32, 32, 32, 34, 36, 36, 36, 38, 40, 40, 40, 41, 43, 43, 43, 44,
46, 46, 46, 46, 46, 46, 46, 45, 45, 45, 31, 31, 31, 31, 31, 32, 32, 32,
32, 34, 36, 36, 36, 38, 40, 40, 40, 41, 43, 43, 43, 44, 46, 46, 46, 46,
46, 46, 46, 45, 45, 45, 37, 37, 38, 38, 38, 39, 40, 40, 40, 41, 43, 43,
43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47, 47, 46,
46, 46, 37, 37, 38, 38, 38, 39, 40, 40, 40, 41, 43, 43, 43, 45, 47, 47,
47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 47, 47, 46, 46, 46, 48, 47,
47, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 49, 50, 50,
50, 51, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 48, 47, 47, 47, 47, 46,
46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 49, 50, 50, 50, 51, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 49, 48, 47, 47, 47, 46, 45, 45, 45, 45,
46, 46, 46, 45, 45, 45, 45, 47, 49, 49, 49, 51, 53, 53, 53, 54, 56, 56,
56, 57, 58, 58 },
},
{
{ /* Luma */
/* Size 4x4 */
32, 32, 32, 35, 32, 32, 33, 35, 32, 33, 35, 38, 35, 35, 38, 46,
/* Size 8x8 */
31, 31, 31, 32, 32, 32, 34, 35, 31, 32, 32, 32, 32, 33, 34, 35, 31, 32,
32, 32, 32, 33, 33, 34, 32, 32, 32, 33, 34, 34, 35, 36, 32, 32, 32, 34,
35, 35, 36, 38, 32, 33, 33, 34, 35, 36, 38, 40, 34, 34, 33, 35, 36, 38,
39, 42, 35, 35, 34, 36, 38, 40, 42, 48,
/* Size 16x16 */
32, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 34, 36, 36, 31, 31,
31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 33, 33, 34, 34, 34, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 34, 34, 34, 31, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 34,
35, 35, 36, 36, 31, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 35, 35, 36,
36, 36, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34, 35, 36, 36, 37, 37,
32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 36, 37, 37, 38, 38, 32, 32,
32, 32, 32, 32, 33, 34, 34, 35, 35, 36, 37, 37, 38, 38, 33, 33, 33, 33,
33, 33, 34, 35, 35, 36, 36, 38, 39, 40, 42, 42, 34, 34, 34, 34, 34, 35, 36,
35, 35, 36, 37, 37, 39, 39, 41, 42, 42, 34, 34, 34, 34, 34, 34, 35, 36,
36, 37, 37, 40, 41, 42, 45, 45, 36, 35, 35, 35, 34, 34, 36, 36, 37, 38,
38, 42, 42, 45, 48, 48, 36, 35, 35, 35, 34, 34, 36, 36, 37, 38, 38, 42,

```

[illegible]

```

35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 35, 36, 36, 36, 36, 37, 37,
38, 38, 38, 39, 41, 42, 42, 42, 44, 46, 47, 47, 47, 48, 36, 35, 35, 35,
35, 35, 35, 34, 34, 34, 34, 35, 36, 36, 36, 36, 37, 38, 38, 38, 38, 40,
42, 42, 42, 42, 45, 47, 48, 48, 48, 49, 36, 35, 35, 35, 35, 35, 35, 34,
34, 34, 34, 35, 36, 36, 36, 36, 37, 38, 38, 38, 38, 40, 42, 42, 42, 42,
45, 47, 48, 48, 48, 49, 36, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 35,
36, 36, 36, 36, 37, 38, 38, 38, 38, 40, 42, 42, 42, 42, 45, 47, 48, 48,
48, 49, 37, 37, 36, 36, 36, 36, 36, 35, 35, 35, 35, 36, 37, 37, 37, 37,
38, 39, 39, 39, 39, 41, 42, 43, 43, 43, 45, 48, 49, 49, 49, 50,
/* Size 4x8 */
31, 31, 32, 35, 32, 32, 32, 35, 32, 32, 33, 34, 32, 32, 34, 36, 32, 33,
35, 38, 33, 33, 36, 40, 34, 34, 37, 42, 35, 34, 38, 48,
/* Size 8x4 */
31, 32, 32, 32, 32, 33, 34, 35, 31, 32, 32, 32, 33, 33, 34, 34, 32, 32,
33, 34, 35, 36, 37, 38, 35, 35, 34, 36, 38, 40, 42, 48,
/* Size 8x16 */
32, 31, 31, 31, 32, 32, 35, 36, 31, 32, 32, 32, 32, 32, 35, 35, 31, 32,
32, 32, 32, 32, 35, 35, 31, 32, 32, 32, 32, 32, 34, 35, 31, 32, 32, 32,
33, 33, 34, 34, 31, 32, 32, 32, 33, 33, 34, 34, 31, 32, 32, 33, 34, 34,
35, 36, 32, 32, 32, 33, 34, 34, 36, 36, 32, 32, 32, 33, 34, 34, 36, 37,
32, 32, 33, 34, 35, 35, 37, 38, 32, 32, 33, 34, 35, 35, 37, 38, 33, 33,
33, 35, 36, 36, 40, 41, 34, 34, 34, 35, 37, 37, 41, 42, 34, 34, 34, 35,
37, 37, 43, 44, 36, 35, 34, 36, 38, 38, 46, 48, 36, 35, 34, 36, 38, 38,
46, 48,
/* Size 16x8 */
32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 33, 34, 34, 36, 36, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 31, 32, 32, 32,
32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 31, 32, 32, 32, 32, 32,
33, 33, 33, 34, 34, 35, 35, 35, 36, 36, 32, 32, 32, 32, 33, 33, 34, 34,
34, 35, 35, 36, 37, 37, 38, 38, 32, 32, 32, 32, 33, 33, 34, 34, 34, 35,
35, 36, 37, 37, 38, 38, 35, 35, 35, 34, 34, 34, 35, 36, 36, 37, 37, 40,
41, 43, 46, 46, 36, 35, 35, 35, 34, 34, 36, 36, 37, 38, 38, 41, 42, 44,
48, 48,
/* Size 16x32 */
32, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 33, 35, 36, 36, 36, 31, 31,
31, 31, 31, 31, 32, 32, 32, 32, 32, 33, 35, 35, 35, 35, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 33, 35, 35, 35, 35, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 33, 35, 35, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 33, 35, 35, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33,
34, 35, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 35,
35, 35, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34,
31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 31, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 31, 32, 32, 32,
32, 32, 33, 33, 33, 33, 33, 34, 35, 35, 35, 35, 31, 32, 32, 32, 32, 32,
33, 33, 34, 34, 34, 34, 35, 36, 36, 36, 32, 32, 32, 32, 32, 32, 33, 34,
34, 34, 34, 35, 36, 36, 36, 36, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34,
34, 35, 36, 36, 36, 36, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 35,
36, 36, 36, 36, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 35, 36, 37,
37, 37, 32, 32, 32, 33, 33, 33, 33, 34, 35, 35, 35, 36, 37, 38, 38, 38,

```

```

32, 32, 32, 33, 33, 33, 34, 35, 35, 35, 35, 36, 37, 38, 38, 38, 32, 32,
32, 33, 33, 33, 34, 35, 35, 35, 35, 36, 37, 38, 38, 38, 32, 32, 32, 33,
33, 33, 34, 35, 35, 35, 35, 36, 37, 38, 38, 38, 32, 33, 33, 33, 33, 33,
34, 35, 36, 36, 36, 37, 39, 40, 40, 40, 33, 33, 33, 33, 33, 33, 35, 36,
36, 36, 36, 38, 40, 41, 41, 41, 34, 34, 34, 34, 34, 34, 35, 36, 37, 37,
37, 39, 41, 42, 42, 42, 34, 34, 34, 34, 34, 34, 35, 36, 37, 37, 37, 39,
41, 42, 42, 42, 34, 34, 34, 34, 34, 34, 35, 36, 37, 37, 37, 39, 41, 42,
42, 42, 34, 34, 34, 34, 34, 34, 35, 37, 37, 37, 37, 40, 43, 44, 44, 44,
35, 35, 34, 34, 34, 34, 36, 37, 38, 38, 38, 41, 45, 47, 47, 47, 36, 35,
35, 34, 34, 34, 36, 37, 38, 38, 38, 42, 46, 48, 48, 48, 36, 35, 35, 34,
34, 34, 36, 37, 38, 38, 38, 42, 46, 48, 48, 48, 36, 35, 35, 34, 34, 34,
36, 37, 38, 38, 38, 42, 46, 48, 48, 48, 37, 36, 36, 36, 36, 36, 37, 38,
39, 39, 39, 42, 46, 49, 49, 49,

```

```
/* Size 32x16 */
```

```

32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 36, 36, 36, 37, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
33, 34, 34, 34, 34, 35, 35, 35, 35, 36, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34,
34, 34, 35, 35, 35, 36, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34,
34, 36, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 36, 31, 31,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 36, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35,
35, 35, 35, 36, 36, 36, 36, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 36, 36, 36, 36, 37, 37,
37, 37, 37, 38, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34,
34, 34, 34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38, 38, 38, 39,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35,
35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38, 38, 38, 39, 32, 32, 32, 32,
32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 36,
36, 37, 37, 37, 37, 38, 38, 38, 38, 39, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 34, 34, 35, 35, 35, 35, 36, 36, 36, 36, 37, 38, 39, 39, 39,
40, 41, 42, 42, 42, 42, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 35,
35, 36, 36, 36, 36, 37, 37, 37, 37, 39, 40, 41, 41, 41, 43, 45, 46, 46,
46, 46, 36, 35, 35, 35, 35, 35, 35, 35, 34, 34, 34, 35, 36, 36, 36, 36,
37, 38, 38, 38, 38, 40, 41, 42, 42, 42, 44, 47, 48, 48, 48, 49, 36, 35,
35, 35, 35, 35, 35, 35, 34, 34, 34, 35, 36, 36, 36, 36, 37, 38, 38, 38,
38, 40, 41, 42, 42, 42, 44, 47, 48, 48, 48, 49, 36, 35, 35, 35, 35, 35,
35, 35, 34, 34, 34, 35, 36, 36, 36, 36, 37, 38, 38, 38, 38, 40, 41, 42,
42, 42, 44, 47, 48, 48, 48, 49,

```

```
/* Size 4x16 */
```

```

31, 31, 32, 36, 31, 32, 32, 35, 32, 32, 32, 35, 32, 32, 35, 32, 32,
33, 34, 32, 32, 33, 34, 32, 32, 34, 36, 32, 32, 34, 36, 32, 32, 34, 37,
32, 33, 35, 38, 32, 33, 35, 38, 33, 33, 36, 41, 34, 34, 37, 42, 34, 34,
37, 44, 35, 34, 38, 48, 35, 34, 38, 48,

```

```
/* Size 16x4 */
```

```

31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 31, 32,

```



```

32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 32, 32, 32, 32,
33, 33, 34, 34, 34, 35, 35, 36, 37, 37, 38, 38, 36, 35, 35, 35, 34, 34,
36, 36, 37, 38, 38, 41, 42, 44, 48, 48,
/* Size 8x32 */
32, 31, 31, 31, 32, 32, 35, 36, 31, 31, 31, 32, 32, 32, 35, 35, 31, 32,
32, 32, 32, 32, 35, 35, 31, 32, 32, 32, 32, 32, 35, 35, 31, 32, 32, 32,
32, 32, 35, 35, 31, 32, 32, 32, 32, 32, 35, 35, 31, 32, 32, 32, 32, 32,
34, 35, 31, 32, 32, 32, 32, 32, 34, 35, 31, 32, 32, 32, 33, 33, 34, 34,
31, 32, 32, 32, 33, 33, 34, 34, 31, 32, 32, 32, 33, 33, 34, 34, 31, 32,
32, 33, 33, 33, 35, 35, 31, 32, 32, 33, 34, 34, 35, 36, 32, 32, 32, 33,
34, 34, 36, 36, 32, 32, 32, 33, 34, 34, 36, 36, 32, 32, 32, 33, 34, 34,
36, 36, 32, 32, 32, 33, 34, 34, 36, 37, 32, 32, 33, 33, 35, 35, 37, 38,
32, 32, 33, 34, 35, 35, 37, 38, 32, 32, 33, 34, 35, 35, 37, 38, 32, 32,
33, 34, 35, 35, 37, 38, 32, 33, 33, 34, 36, 36, 39, 40, 33, 33, 33, 35,
36, 36, 40, 41, 34, 34, 34, 35, 37, 37, 41, 42, 34, 34, 34, 35, 37, 37,
41, 42, 34, 34, 34, 35, 37, 37, 41, 42, 34, 34, 34, 35, 37, 37, 43, 44,
35, 34, 34, 36, 38, 38, 45, 47, 36, 35, 34, 36, 38, 38, 46, 48, 36, 35,
34, 36, 38, 38, 46, 48, 36, 35, 34, 36, 38, 38, 46, 48, 37, 36, 36, 37,
39, 39, 46, 49,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 36, 36, 36, 37, 31, 31, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
33, 34, 34, 34, 34, 34, 35, 35, 35, 36, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34,
34, 34, 34, 34, 34, 36, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 35, 35, 35, 36, 36, 36,
36, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34,
34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 37, 38, 38, 38, 38, 39, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35, 35,
35, 36, 36, 37, 37, 37, 37, 38, 38, 38, 38, 39, 35, 35, 35, 35, 35, 35,
34, 34, 34, 34, 34, 35, 35, 36, 36, 36, 36, 37, 37, 37, 37, 39, 40, 41,
41, 41, 43, 45, 46, 46, 46, 46, 36, 35, 35, 35, 35, 35, 35, 35, 34, 34,
34, 35, 36, 36, 36, 36, 37, 38, 38, 38, 38, 40, 41, 42, 42, 42, 44, 47,
48, 48, 48, 49 },
{ /* Chroma */
/* Size 4x4 */
31, 32, 38, 46, 32, 34, 41, 46, 38, 41, 47, 47, 46, 46, 47, 52,
/* Size 8x8 */
31, 31, 30, 34, 36, 39, 42, 48, 31, 31, 31, 34, 37, 40, 42, 47, 30, 31,
32, 35, 39, 41, 42, 46, 34, 34, 35, 39, 42, 44, 45, 47, 36, 37, 39, 42,
46, 47, 47, 47, 39, 40, 41, 44, 47, 47, 48, 49, 42, 42, 42, 45, 47, 48,
48, 50, 48, 47, 46, 47, 47, 49, 50, 53,
/* Size 16x16 */
32, 31, 31, 31, 30, 30, 33, 33, 34, 36, 36, 40, 41, 44, 49, 49, 31, 31,
31, 31, 31, 31, 33, 34, 36, 38, 38, 41, 42, 44, 48, 48, 31, 31, 31, 31,
31, 31, 34, 34, 36, 38, 38, 41, 42, 44, 47, 47, 31, 31, 31, 31, 31, 31,
34, 35, 36, 39, 39, 41, 42, 44, 47, 47, 30, 31, 31, 31, 32, 32, 34, 35,
37, 40, 40, 42, 42, 44, 46, 46, 30, 31, 31, 31, 32, 32, 34, 35, 37, 40,
40, 42, 42, 44, 46, 46, 33, 33, 34, 34, 34, 34, 37, 38, 40, 42, 42, 44,

```

```

44, 45, 47, 47, 33, 34, 34, 35, 35, 35, 38, 39, 40, 43, 43, 44, 45, 46,
47, 47, 34, 36, 36, 36, 37, 37, 40, 40, 42, 45, 45, 45, 46, 46, 47, 47,
36, 38, 38, 39, 40, 40, 42, 43, 45, 47, 47, 47, 47, 47, 48, 48, 36, 38,
38, 39, 40, 40, 42, 43, 45, 47, 47, 47, 47, 47, 48, 48, 40, 41, 41, 41,
42, 42, 44, 44, 45, 47, 47, 48, 48, 49, 50, 50, 41, 42, 42, 42, 42, 42,
44, 45, 46, 47, 47, 48, 48, 49, 50, 50, 44, 44, 44, 44, 44, 44, 45, 46,
46, 47, 47, 49, 49, 50, 51, 51, 49, 48, 47, 47, 46, 46, 47, 47, 47, 48,
48, 50, 50, 51, 53, 53, 49, 48, 47, 47, 46, 46, 47, 47, 47, 48, 48, 50,
50, 51, 53, 53,
/* Size 32x32 */
32, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 33, 33, 33, 33, 34, 36,
36, 36, 36, 38, 40, 41, 41, 41, 44, 47, 49, 49, 49, 49, 31, 31, 31, 31,
31, 31, 31, 31, 30, 30, 30, 32, 33, 34, 34, 34, 35, 36, 37, 37, 37, 39,
41, 42, 42, 42, 44, 47, 48, 48, 48, 48, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 32, 33, 34, 34, 34, 36, 37, 38, 38, 38, 39, 41, 42, 42, 42,
44, 46, 48, 48, 48, 47, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32,
34, 34, 34, 34, 36, 37, 38, 38, 38, 40, 41, 42, 42, 42, 44, 46, 47, 47,
47, 47, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 34, 34, 34, 34,
36, 37, 38, 38, 38, 40, 41, 42, 42, 42, 44, 46, 47, 47, 47, 47, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 34, 34, 34, 34, 36, 37, 38, 38,
38, 40, 41, 42, 42, 42, 44, 46, 47, 47, 47, 47, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 33, 34, 35, 35, 35, 36, 38, 39, 39, 39, 40, 41, 42,
42, 42, 44, 46, 47, 47, 47, 47, 30, 31, 31, 31, 31, 31, 31, 31, 31,
31, 33, 34, 35, 35, 35, 37, 38, 39, 39, 39, 41, 42, 42, 42, 42, 44, 46,
46, 46, 46, 46, 30, 30, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 35,
35, 35, 37, 39, 40, 40, 40, 41, 42, 42, 42, 42, 44, 45, 46, 46, 46, 46,
30, 30, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 37, 39,
40, 40, 40, 41, 42, 42, 42, 42, 44, 45, 46, 46, 46, 46, 30, 30, 31, 31,
31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 37, 39, 40, 40, 40, 41,
42, 42, 42, 42, 44, 45, 46, 46, 46, 46, 31, 32, 32, 32, 32, 32, 33, 33,
33, 33, 33, 34, 36, 37, 37, 37, 38, 40, 41, 41, 41, 42, 43, 43, 43, 43,
44, 46, 46, 46, 46, 46, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 36,
37, 38, 38, 38, 40, 41, 42, 42, 42, 43, 44, 44, 44, 44, 45, 46, 47, 47,
47, 46, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 37, 38, 39, 39, 39,
40, 42, 43, 43, 43, 44, 44, 45, 45, 45, 46, 47, 47, 47, 47, 47, 33, 34,
34, 34, 34, 35, 35, 35, 35, 35, 37, 38, 39, 39, 39, 40, 42, 43, 43,
43, 44, 44, 45, 45, 45, 46, 47, 47, 47, 47, 47, 33, 34, 34, 34, 34, 34,
35, 35, 35, 35, 35, 37, 38, 39, 39, 39, 40, 42, 43, 43, 43, 44, 44, 45,
45, 45, 46, 47, 47, 47, 47, 47, 34, 35, 36, 36, 36, 36, 36, 37, 37, 37,
37, 38, 40, 40, 40, 40, 42, 44, 45, 45, 45, 45, 45, 46, 46, 46, 46, 47,
47, 47, 47, 47, 36, 36, 37, 37, 37, 37, 38, 38, 39, 39, 39, 40, 41, 42,
42, 42, 44, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
36, 37, 38, 38, 38, 38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43, 45, 46,
47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 47, 36, 37, 38, 38,
38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43, 45, 46, 47, 47, 47, 47, 47,
47, 47, 48, 48, 48, 47, 38, 39, 39, 40, 40, 40, 40, 41, 41, 41, 41, 42,
43, 44, 44, 44, 45, 47, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 49, 49,
49, 48, 40, 41, 41, 41, 41, 41, 41, 42, 42, 42, 42, 43, 44, 44, 44, 44,

```

```

45, 47, 47, 47, 47, 48, 48, 48, 48, 48, 49, 49, 50, 50, 50, 49, 41, 42,
42, 42, 42, 42, 42, 42, 42, 42, 42, 43, 44, 45, 45, 45, 46, 47, 47, 47,
47, 48, 48, 48, 48, 48, 49, 50, 50, 50, 50, 50, 41, 42, 42, 42, 42, 42,
42, 42, 42, 42, 42, 43, 44, 45, 45, 45, 46, 47, 47, 47, 47, 48, 48, 48,
48, 48, 49, 50, 50, 50, 50, 50, 41, 42, 42, 42, 42, 42, 42, 42, 42,
42, 43, 44, 45, 45, 45, 46, 47, 47, 47, 47, 48, 48, 48, 48, 48, 49, 50,
50, 50, 50, 50, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 45, 46,
46, 46, 46, 47, 47, 47, 47, 48, 49, 49, 49, 49, 50, 51, 51, 51, 51, 51,
47, 47, 46, 46, 46, 46, 46, 46, 45, 45, 45, 46, 46, 47, 47, 47, 47, 47,
47, 47, 47, 48, 49, 50, 50, 50, 51, 52, 52, 52, 52, 52, 49, 48, 48, 47,
47, 47, 47, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 48, 48, 49,
50, 50, 50, 50, 51, 52, 53, 53, 53, 53, 49, 48, 48, 47, 47, 47, 47, 46,
46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 48, 48, 48, 49, 50, 50, 50, 50,
51, 52, 53, 53, 53, 53, 49, 48, 48, 47, 47, 47, 47, 46, 46, 46, 46, 46,
47, 47, 47, 47, 47, 47, 48, 48, 48, 49, 50, 50, 50, 50, 51, 52, 53, 53,
53, 53, 49, 48, 47, 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 47, 47, 47,
47, 47, 47, 47, 47, 48, 49, 50, 50, 50, 51, 52, 53, 53, 53, 53,
/* Size 4x8 */
31, 31, 37, 48, 31, 31, 38, 47, 31, 32, 40, 46, 34, 36, 43, 47, 37, 39,
46, 47, 39, 41, 47, 48, 42, 43, 47, 50, 48, 46, 48, 53,
/* Size 8x4 */
31, 31, 31, 34, 37, 39, 42, 48, 31, 31, 32, 36, 39, 41, 43, 46, 37, 38,
40, 43, 46, 47, 47, 48, 48, 47, 46, 47, 47, 48, 50, 53,
/* Size 8x16 */
32, 31, 31, 33, 37, 37, 45, 48, 31, 31, 31, 34, 38, 38, 45, 47, 31, 31,
31, 34, 38, 38, 45, 47, 31, 31, 32, 34, 39, 39, 45, 46, 30, 32, 32, 35,
40, 40, 44, 46, 30, 32, 32, 35, 40, 40, 44, 46, 33, 34, 35, 37, 42, 42,
46, 47, 33, 35, 36, 38, 43, 43, 46, 47, 35, 37, 37, 40, 44, 44, 46, 47,
37, 39, 40, 43, 47, 47, 47, 47, 37, 39, 40, 43, 47, 47, 47, 47, 41, 42,
42, 44, 47, 47, 49, 49, 42, 42, 43, 44, 47, 47, 49, 50, 44, 44, 44, 45,
47, 47, 50, 51, 49, 47, 46, 47, 48, 48, 52, 53, 49, 47, 46, 47, 48, 48,
52, 53,
/* Size 16x8 */
32, 31, 31, 31, 30, 30, 33, 33, 35, 37, 37, 41, 42, 44, 49, 49, 31, 31,
31, 31, 32, 32, 34, 35, 37, 39, 39, 42, 42, 44, 47, 47, 31, 31, 31, 32,
32, 32, 35, 36, 37, 40, 40, 42, 43, 44, 46, 46, 33, 34, 34, 34, 35, 35,
37, 38, 40, 43, 43, 44, 44, 45, 47, 47, 37, 38, 38, 39, 40, 40, 42, 43,
44, 47, 47, 47, 47, 47, 48, 48, 37, 38, 38, 39, 40, 40, 42, 43, 44, 47,
47, 47, 47, 47, 48, 48, 45, 45, 45, 45, 44, 44, 46, 46, 46, 47, 47, 49,
49, 50, 52, 52, 48, 47, 47, 46, 46, 46, 47, 47, 47, 47, 47, 49, 50, 51,
53, 53,
/* Size 16x32 */
32, 31, 31, 31, 31, 31, 33, 35, 37, 37, 37, 40, 45, 48, 48, 48, 31, 31,
31, 31, 31, 31, 33, 36, 37, 37, 37, 41, 45, 48, 48, 48, 31, 31, 31, 31,
31, 31, 34, 36, 38, 38, 38, 41, 45, 47, 47, 47, 31, 31, 31, 31, 31, 31,
34, 37, 38, 38, 38, 41, 45, 47, 47, 47, 31, 31, 31, 31, 31, 31, 34, 37,
38, 38, 38, 41, 45, 47, 47, 47, 31, 31, 31, 31, 31, 31, 34, 37, 38, 38,
38, 41, 45, 47, 47, 47, 31, 31, 31, 32, 32, 32, 34, 37, 39, 39, 39, 41,
45, 46, 46, 46, 30, 31, 31, 32, 32, 32, 34, 38, 39, 39, 39, 42, 44, 46,
46, 46, 30, 31, 32, 32, 32, 32, 35, 38, 40, 40, 40, 42, 44, 46, 46, 46,

```

```

30, 31, 32, 32, 32, 32, 35, 38, 40, 40, 40, 42, 44, 46, 46, 46, 30, 31,
32, 32, 32, 32, 35, 38, 40, 40, 40, 42, 44, 46, 46, 46, 31, 32, 33, 33,
33, 33, 36, 39, 41, 41, 41, 43, 45, 46, 46, 46, 33, 34, 34, 35, 35, 35,
37, 40, 42, 42, 42, 44, 46, 47, 47, 47, 33, 34, 35, 36, 36, 36, 38, 41,
43, 43, 43, 44, 46, 47, 47, 47, 33, 34, 35, 36, 36, 36, 38, 41, 43, 43,
43, 44, 46, 47, 47, 47, 33, 34, 35, 36, 36, 36, 38, 41, 43, 43, 43, 44,
46, 47, 47, 47, 35, 36, 37, 37, 37, 37, 40, 43, 44, 44, 44, 45, 46, 47,
47, 47, 36, 37, 38, 39, 39, 39, 42, 44, 46, 46, 46, 47, 47, 47, 47, 47,
37, 38, 39, 40, 40, 40, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 37, 38,
39, 40, 40, 40, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 37, 38, 39, 40,
40, 40, 43, 45, 47, 47, 47, 47, 47, 47, 47, 47, 39, 39, 40, 41, 41, 41,
43, 46, 47, 47, 47, 48, 48, 48, 48, 48, 41, 41, 42, 42, 42, 42, 44, 46,
47, 47, 47, 48, 49, 49, 49, 49, 42, 42, 42, 43, 43, 43, 44, 46, 47, 47,
47, 48, 49, 50, 50, 50, 42, 42, 42, 43, 43, 43, 44, 46, 47, 47, 47, 48,
49, 50, 50, 50, 42, 42, 42, 43, 43, 43, 44, 46, 47, 47, 47, 48, 49, 50,
50, 50, 44, 44, 44, 44, 44, 44, 44, 45, 47, 47, 47, 47, 49, 50, 51, 51, 51,
47, 46, 46, 46, 46, 46, 46, 47, 48, 48, 48, 49, 51, 52, 52, 52, 49, 48,
47, 46, 46, 46, 47, 48, 48, 48, 48, 50, 52, 53, 53, 53, 49, 48, 47, 46,
46, 46, 46, 47, 48, 48, 48, 48, 50, 52, 53, 53, 53, 49, 48, 47, 46, 46,
47, 47, 47, 49, 52, 53, 53, 53,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 33, 33, 33, 33, 35, 36,
37, 37, 37, 39, 41, 42, 42, 42, 44, 47, 49, 49, 49, 49, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 32, 34, 34, 34, 34, 36, 37, 38, 38, 38, 39,
41, 42, 42, 42, 44, 46, 48, 48, 48, 48, 31, 31, 31, 31, 31, 31, 31, 31,
32, 32, 32, 33, 34, 35, 35, 35, 37, 38, 39, 39, 39, 40, 42, 42, 42, 42,
44, 46, 47, 47, 47, 47, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 33,
35, 36, 36, 36, 37, 39, 40, 40, 40, 41, 42, 43, 43, 43, 44, 46, 46, 46,
46, 46, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 33, 35, 36, 36, 36,
37, 39, 40, 40, 40, 41, 42, 43, 43, 43, 44, 46, 46, 46, 46, 46, 31, 31,
31, 31, 31, 31, 32, 32, 32, 32, 32, 33, 35, 36, 36, 36, 37, 39, 40, 40,
40, 41, 42, 43, 43, 43, 44, 46, 46, 46, 46, 46, 33, 33, 34, 34, 34, 34,
34, 34, 35, 35, 35, 36, 37, 38, 38, 38, 40, 42, 43, 43, 43, 43, 44, 44,
44, 44, 45, 46, 47, 47, 47, 47, 35, 36, 36, 37, 37, 37, 37, 38, 38, 38,
38, 39, 40, 41, 41, 41, 43, 44, 45, 45, 45, 46, 46, 46, 46, 46, 47, 47,
48, 48, 48, 47, 37, 37, 38, 38, 38, 38, 39, 39, 40, 40, 40, 41, 42, 43,
43, 43, 44, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 47,
37, 37, 38, 38, 38, 38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43, 44, 46,
47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 47, 37, 37, 38, 38,
38, 38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43, 44, 46, 47, 47, 47, 47,
47, 47, 47, 47, 47, 48, 48, 48, 48, 47, 40, 41, 41, 41, 41, 41, 41, 42,
42, 42, 42, 43, 44, 44, 44, 44, 45, 47, 47, 47, 47, 48, 48, 48, 48, 48,
49, 49, 50, 50, 50, 49, 45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 45,
46, 46, 46, 46, 46, 47, 47, 47, 47, 48, 49, 49, 49, 49, 50, 51, 52, 52,
52, 52, 48, 48, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47,
47, 47, 47, 47, 47, 48, 49, 50, 50, 50, 51, 52, 53, 53, 53, 53, 48, 48,
47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 47,
47, 48, 49, 50, 50, 50, 51, 52, 53, 53, 53, 53, 48, 48, 47, 47, 47, 47,
46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 47, 48, 49, 50,

```

```

50, 50, 51, 52, 53, 53, 53, 53,
/* Size 4x16 */
31, 31, 37, 48, 31, 31, 38, 47, 31, 31, 38, 47, 31, 32, 39, 46, 31, 32,
40, 46, 31, 32, 40, 46, 34, 35, 42, 47, 34, 36, 43, 47, 36, 37, 44, 47,
38, 40, 47, 47, 38, 40, 47, 47, 41, 42, 47, 49, 42, 43, 47, 50, 44, 44,
47, 51, 48, 46, 48, 53, 48, 46, 48, 53,
/* Size 16x4 */
31, 31, 31, 31, 31, 31, 34, 34, 36, 38, 38, 41, 42, 44, 48, 48, 31, 31,
31, 32, 32, 32, 35, 36, 37, 40, 40, 42, 43, 44, 46, 46, 37, 38, 38, 39,
40, 40, 42, 43, 44, 47, 47, 47, 47, 47, 48, 48, 48, 47, 47, 46, 46, 46,
47, 47, 47, 47, 47, 49, 50, 51, 53, 53,
/* Size 8x32 */
32, 31, 31, 33, 37, 37, 45, 48, 31, 31, 31, 33, 37, 37, 45, 48, 31, 31,
31, 34, 38, 38, 45, 47, 31, 31, 31, 34, 38, 38, 45, 47, 31, 31, 31, 34,
38, 38, 45, 47, 31, 31, 31, 34, 38, 38, 45, 47, 31, 31, 32, 34, 39, 39,
45, 46, 30, 31, 32, 34, 39, 39, 44, 46, 30, 32, 32, 35, 40, 40, 44, 46,
30, 32, 32, 35, 40, 40, 44, 46, 30, 32, 32, 35, 40, 40, 44, 46, 31, 33,
33, 36, 41, 41, 45, 46, 33, 34, 35, 37, 42, 42, 46, 47, 33, 35, 36, 38,
43, 43, 46, 47, 33, 35, 36, 38, 43, 43, 46, 47, 33, 35, 36, 38, 43, 43,
46, 47, 35, 37, 37, 40, 44, 44, 46, 47, 36, 38, 39, 42, 46, 46, 47, 47,
37, 39, 40, 43, 47, 47, 47, 47, 37, 39, 40, 43, 47, 47, 47, 47, 37, 39,
40, 43, 47, 47, 47, 47, 39, 40, 41, 43, 47, 47, 48, 48, 41, 42, 42, 44,
47, 47, 49, 49, 42, 42, 43, 44, 47, 47, 49, 50, 42, 42, 43, 44, 47, 47,
49, 50, 42, 42, 43, 44, 47, 47, 49, 50, 44, 44, 44, 45, 47, 47, 50, 51,
47, 46, 46, 46, 48, 48, 51, 52, 49, 47, 46, 47, 48, 48, 52, 53, 49, 47,
46, 47, 48, 48, 52, 53, 49, 47, 46, 47, 48, 48, 52, 53, 49, 47, 46, 47,
47, 47, 52, 53,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 33, 33, 33, 33, 35, 36,
37, 37, 37, 39, 41, 42, 42, 42, 44, 47, 49, 49, 49, 49, 31, 31, 31, 31,
31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 37, 38, 39, 39, 39, 40,
42, 42, 42, 42, 44, 46, 47, 47, 47, 47, 31, 31, 31, 31, 31, 31, 32, 32,
32, 32, 32, 33, 35, 36, 36, 36, 37, 39, 40, 40, 40, 41, 42, 43, 43, 43,
44, 46, 46, 46, 46, 46, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 36,
37, 38, 38, 38, 40, 42, 43, 43, 43, 43, 44, 44, 44, 44, 45, 46, 47, 47,
47, 47, 37, 37, 38, 38, 38, 38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43,
44, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 47, 37, 37,
38, 38, 38, 38, 39, 39, 40, 40, 40, 41, 42, 43, 43, 43, 44, 46, 47, 47,
47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 47, 45, 45, 45, 45, 45, 45,
45, 44, 44, 44, 44, 45, 46, 46, 46, 46, 46, 47, 47, 47, 47, 48, 49, 49,
49, 49, 50, 51, 52, 52, 52, 52, 48, 48, 47, 47, 47, 47, 46, 46, 46, 46,
46, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 49, 50, 50, 50, 51, 52,
53, 53, 53, 53 },
},
{
/* Luma */
/* Size 4x4 */
31, 32, 32, 32, 32, 32, 32, 33, 32, 32, 33, 34, 32, 33, 34, 35,
/* Size 8x8 */
31, 31, 31, 31, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 33, 31, 32,

```

[illegible]

```

34, 34, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35,
31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33,
33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 31, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34,
34, 34, 34, 35, 35, 35, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35,
35, 35, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33,
33, 33, 33, 33, 33, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34,
34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 36, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35,
35, 35, 35, 36, 36, 37, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35,
36, 36, 37, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
33, 33, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 37, 37,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34,
34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 32, 32, 33, 33,
33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35,
35, 36, 36, 36, 36, 36, 37, 38, 38, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35, 36, 36, 36,
36, 36, 37, 38, 38, 38, 34, 34, 34, 34, 34, 34, 34, 34, 34, 33, 33, 33,
33, 33, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38,
39, 39, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 33, 33, 33, 33, 33, 34, 34,
35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 37, 37, 38, 38, 39, 39,
/* Size 4x8 */
31, 31, 32, 32, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 32, 32,
33, 34, 32, 32, 34, 34, 32, 33, 34, 35, 33, 33, 35, 36,
/* Size 8x4 */
31, 31, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 33, 33, 32, 32,
32, 32, 33, 34, 34, 35, 32, 32, 32, 33, 34, 34, 35, 36,
/* Size 8x16 */
32, 31, 31, 31, 31, 32, 32, 32, 31, 31, 32, 32, 32, 32, 32, 33, 31, 32,
32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32,
32, 32, 32, 33, 31, 32, 32, 32, 32, 33, 33, 33, 31, 32, 32, 32, 32, 33,
33, 33, 31, 32, 32, 32, 32, 33, 33, 33, 31, 32, 32, 32, 33, 34, 34, 34,
32, 32, 32, 32, 33, 34, 34, 34, 32, 32, 32, 32, 33, 34, 34, 34, 32, 32,
32, 32, 33, 35, 35, 35, 32, 32, 33, 33, 34, 35, 35, 36, 32, 32, 33, 33,
34, 35, 35, 36, 32, 33, 33, 33, 34, 36, 36, 36, 34, 34, 34, 34, 35, 37,
37, 38,
/* Size 16x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 31, 31,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 31, 32, 32, 32, 32, 32, 32, 32,
33, 33, 33, 33, 34, 34, 34, 35, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34,
34, 35, 35, 35, 36, 37, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 35,
35, 35, 36, 37, 32, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 35, 36, 36,

```

[illegible]



```

32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34,
35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35,
35, 35, 36, 36, 37, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33,
33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36,
37, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33,
34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 32, 33,
33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34,
34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 38, 34, 34, 34, 34, 34, 34,
34, 34, 34, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36,
37, 37, 37, 37, 38, 38, 39, 39,
/* Size 4x16 */
31, 31, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32,
32, 32, 32, 32, 32, 33, 32, 32, 32, 33, 32, 32, 33, 33, 32, 32, 33, 34,
32, 32, 33, 34, 32, 32, 33, 34, 32, 32, 34, 35, 32, 33, 34, 35, 32, 33,
34, 35, 33, 33, 35, 36, 34, 34, 36, 37,
/* Size 16x4 */
31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 32, 32, 32, 32,
32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 36, 32, 32, 32, 32, 32, 33,
33, 33, 34, 34, 34, 35, 35, 35, 36, 37,
/* Size 8x32 */
32, 31, 31, 31, 31, 32, 32, 32, 31, 31, 31, 31, 32, 32, 32, 33, 31, 31,
32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32,
32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32,
32, 33, 31, 32, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 32, 33,
31, 32, 32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 33, 33, 33, 31, 32,
32, 32, 33, 33, 33, 33, 31, 32, 32, 32, 32, 33, 33, 33, 31, 32, 32, 32,
32, 33, 33, 33, 31, 32, 32, 32, 32, 33, 33, 33, 31, 32, 32, 32, 33, 33,
33, 34, 31, 32, 32, 32, 33, 34, 34, 34, 32, 32, 32, 32, 33, 34, 34, 34,
32, 32, 32, 32, 33, 34, 34, 34, 32, 32, 32, 32, 33, 34, 34, 34, 32, 32,
32, 32, 33, 34, 34, 34, 32, 32, 32, 32, 33, 34, 34, 35, 32, 32, 32, 32,
33, 35, 35, 35, 32, 32, 33, 33, 33, 35, 35, 36, 32, 32, 33, 33, 34, 35,
35, 36, 32, 32, 33, 33, 34, 35, 35, 36, 32, 32, 33, 33, 34, 35, 35, 36,
32, 32, 33, 33, 34, 35, 35, 36, 32, 33, 33, 33, 34, 36, 36, 36, 33, 33,
33, 33, 34, 36, 36, 37, 34, 34, 34, 34, 35, 37, 37, 38, 34, 34, 34, 34,
35, 37, 37, 38,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 33, 33, 34, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33,
34, 34, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34,
34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 37, 37, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35,

```

```

35, 35, 35, 35, 36, 36, 37, 37, 32, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 36, 36, 36, 36,
36, 37, 38, 38 },
{ /* Chroma */
  /* Size 4x4 */
  31, 31, 34, 38, 31, 32, 35, 40, 34, 35, 39, 43, 38, 40, 43, 47,
  /* Size 8x8 */
  31, 31, 31, 30, 34, 35, 37, 40, 31, 31, 31, 31, 34, 35, 38, 41, 31, 31,
  31, 31, 35, 36, 39, 41, 30, 31, 31, 32, 35, 36, 40, 42, 34, 34, 35, 35,
  39, 40, 43, 44, 35, 35, 36, 36, 40, 41, 44, 45, 37, 38, 39, 40, 43, 44,
  47, 47, 40, 41, 41, 42, 44, 45, 47, 48,
  /* Size 16x16 */
  32, 31, 31, 31, 31, 30, 30, 31, 33, 33, 33, 35, 36, 36, 38, 41, 31, 31,
  31, 31, 31, 31, 31, 31, 33, 34, 34, 36, 37, 37, 39, 42, 31, 31, 31, 31,
  31, 31, 31, 32, 34, 34, 34, 37, 38, 38, 40, 42, 31, 31, 31, 31, 31, 31,
  31, 32, 34, 34, 34, 37, 38, 38, 40, 42, 31, 31, 31, 31, 31, 31, 31, 32,
  34, 35, 35, 37, 39, 39, 40, 42, 30, 31, 31, 31, 31, 32, 32, 32, 34, 35,
  35, 38, 40, 40, 41, 42, 30, 31, 31, 31, 31, 32, 32, 32, 34, 35, 35, 38,
  40, 40, 41, 42, 31, 31, 32, 32, 32, 32, 32, 33, 35, 36, 36, 38, 40, 40,
  41, 43, 33, 33, 34, 34, 34, 34, 34, 35, 37, 38, 38, 41, 42, 42, 43, 44,
  33, 34, 34, 34, 35, 35, 35, 36, 38, 39, 39, 41, 43, 43, 44, 45, 33, 34,
  34, 34, 35, 35, 35, 36, 38, 39, 39, 41, 43, 43, 44, 45, 35, 36, 37, 37,
  37, 38, 38, 38, 41, 41, 41, 44, 46, 46, 46, 46, 36, 37, 38, 38, 39, 40,
  40, 40, 42, 43, 43, 46, 47, 47, 47, 47, 36, 37, 38, 38, 39, 40, 40, 40,
  42, 43, 43, 46, 47, 47, 47, 47, 38, 39, 40, 40, 40, 41, 41, 41, 43, 44,
  44, 46, 47, 47, 47, 48, 41, 42, 42, 42, 42, 42, 42, 43, 44, 45, 45, 46,
  47, 47, 48, 48,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 31, 32, 33, 33,
  33, 33, 33, 34, 35, 36, 36, 36, 36, 37, 38, 40, 41, 41, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 32, 33, 34, 34, 34, 34, 35,
  36, 37, 37, 37, 37, 37, 39, 40, 42, 42, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34, 34, 34, 35, 36, 37, 37, 37,
  37, 38, 39, 40, 42, 42, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 32, 32, 34, 34, 34, 34, 34, 35, 36, 38, 38, 38, 38, 38, 40, 41,
  42, 42, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33,
  34, 34, 34, 34, 34, 35, 37, 38, 38, 38, 38, 39, 40, 41, 42, 42, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34,
  34, 35, 37, 38, 38, 38, 38, 39, 40, 41, 42, 42, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34, 34, 34, 34, 35, 37, 38,
  38, 38, 38, 39, 40, 41, 42, 42, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 32, 33, 34, 34, 34, 34, 34, 36, 37, 38, 38, 38, 38, 39,
  40, 41, 42, 42, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  32, 33, 34, 35, 35, 35, 35, 36, 37, 38, 39, 39, 39, 39, 40, 41, 42, 42,
  30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 35,
  35, 35, 35, 36, 37, 39, 39, 39, 39, 40, 40, 41, 42, 42, 30, 30, 31, 31,
  31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 33, 34, 35, 35, 35, 35, 36,
  38, 39, 40, 40, 40, 40, 41, 42, 42, 42, 30, 30, 31, 31, 31, 31, 31, 31,
  31, 31, 32, 32, 32, 32, 32, 33, 34, 35, 35, 35, 35, 36, 38, 39, 40, 40,
  40, 40, 41, 42, 42, 42, 30, 30, 31, 31, 31, 31, 31, 31, 31, 32, 32,

```

```

32, 32, 32, 33, 34, 35, 35, 35, 35, 36, 38, 39, 40, 40, 40, 40, 41, 42,
42, 42, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 33,
34, 35, 35, 35, 35, 36, 38, 39, 40, 40, 40, 40, 41, 42, 42, 42, 31, 31,
31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 35, 36, 36, 36,
36, 37, 38, 40, 40, 40, 40, 41, 41, 42, 43, 43, 32, 32, 32, 32, 33, 33,
33, 33, 33, 33, 33, 33, 33, 33, 34, 35, 36, 37, 37, 37, 37, 38, 39, 41,
41, 41, 41, 42, 42, 43, 43, 43, 33, 33, 33, 34, 34, 34, 34, 34, 34,
34, 34, 34, 34, 35, 36, 37, 38, 38, 38, 38, 39, 41, 42, 42, 42, 42, 43,
43, 44, 44, 44, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35,
36, 37, 38, 39, 39, 39, 39, 40, 41, 43, 43, 43, 43, 43, 44, 44, 45, 45,
33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 37, 38, 39,
39, 39, 39, 40, 41, 43, 43, 43, 43, 43, 44, 44, 45, 45, 33, 34, 34, 34,
34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 36, 37, 38, 39, 39, 39, 39, 40,
41, 43, 43, 43, 43, 43, 44, 44, 45, 45, 33, 34, 34, 34, 34, 34, 34, 34,
35, 35, 35, 35, 35, 35, 36, 37, 38, 39, 39, 39, 39, 40, 41, 43, 43, 43,
43, 43, 44, 44, 45, 45, 34, 35, 35, 35, 35, 35, 35, 36, 36, 36, 36, 36,
36, 36, 37, 38, 39, 40, 40, 40, 40, 41, 42, 44, 44, 44, 44, 44, 45, 45,
45, 45, 35, 36, 36, 36, 37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 39,
41, 41, 41, 41, 41, 42, 44, 45, 46, 46, 46, 46, 46, 46, 46, 46, 36, 37,
37, 38, 38, 38, 38, 38, 38, 39, 39, 39, 39, 39, 40, 41, 42, 43, 43, 43,
43, 44, 45, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 36, 37, 37, 38, 38, 38,
38, 38, 39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43, 43, 43, 44, 46, 47,
47, 47, 47, 47, 47, 47, 47, 47, 36, 37, 37, 38, 38, 38, 38, 38, 39, 39,
40, 40, 40, 40, 40, 41, 42, 43, 43, 43, 43, 44, 46, 47, 47, 47, 47, 47,
47, 47, 47, 47, 36, 37, 37, 38, 38, 38, 38, 38, 39, 39, 40, 40, 40, 40,
40, 41, 42, 43, 43, 43, 43, 44, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47,
37, 37, 38, 38, 39, 39, 39, 39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43,
43, 43, 43, 44, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 38, 39, 39, 40,
40, 40, 40, 40, 40, 41, 41, 41, 41, 41, 42, 43, 44, 44, 44, 44, 45,
46, 47, 47, 47, 47, 47, 47, 48, 48, 48, 40, 40, 40, 41, 41, 41, 41, 41,
41, 41, 42, 42, 42, 42, 42, 42, 43, 44, 44, 44, 44, 44, 45, 46, 47, 47, 47,
47, 47, 48, 48, 48, 48, 41, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
42, 42, 43, 43, 44, 45, 45, 45, 45, 45, 46, 47, 47, 47, 47, 47, 48, 48,
48, 48, 41, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 43, 43,
44, 45, 45, 45, 45, 46, 47, 47, 47, 47, 47, 48, 48, 48, 48,
/* Size 4x8 */
31, 31, 35, 37, 31, 31, 36, 38, 31, 32, 37, 39, 31, 32, 37, 40, 34, 36,
40, 43, 35, 37, 42, 44, 38, 40, 45, 47, 41, 42, 45, 47,
/* Size 8x4 */
31, 31, 31, 31, 34, 35, 38, 41, 31, 31, 32, 32, 36, 37, 40, 42, 35, 36,
37, 37, 40, 42, 45, 45, 37, 38, 39, 40, 43, 44, 47, 47,
/* Size 8x16 */
32, 31, 31, 31, 33, 37, 37, 38, 31, 31, 31, 31, 33, 38, 38, 39, 31, 31,
31, 31, 34, 38, 38, 40, 31, 31, 31, 31, 34, 38, 38, 40, 31, 31, 32, 32,
34, 39, 39, 40, 30, 31, 32, 32, 35, 40, 40, 41, 30, 31, 32, 32, 35, 40,
40, 41, 31, 32, 33, 33, 35, 40, 40, 41, 33, 34, 35, 35, 37, 42, 42, 43,
33, 35, 36, 36, 38, 43, 43, 44, 33, 35, 36, 36, 38, 43, 43, 44, 35, 37,
38, 38, 41, 45, 45, 46, 37, 39, 40, 40, 43, 47, 47, 47, 37, 39, 40, 40,
43, 47, 47, 47, 39, 40, 41, 41, 43, 47, 47, 47, 42, 42, 43, 43, 44, 47,
47, 48,

```

```
/* Size 16x8 */
```

```
32, 31, 31, 31, 31, 30, 30, 31, 33, 33, 33, 35, 37, 37, 39, 42, 31, 31,
31, 31, 31, 31, 31, 32, 34, 35, 35, 37, 39, 39, 40, 42, 31, 31, 31, 31,
32, 32, 32, 33, 35, 36, 36, 38, 40, 40, 41, 43, 31, 31, 31, 31, 32, 32,
32, 33, 35, 36, 36, 38, 40, 40, 41, 43, 33, 33, 34, 34, 34, 35, 35, 35,
37, 38, 38, 41, 43, 43, 43, 44, 37, 38, 38, 38, 39, 40, 40, 40, 42, 43,
43, 45, 47, 47, 47, 47, 37, 38, 38, 38, 39, 40, 40, 40, 42, 43, 43, 45,
47, 47, 47, 47, 38, 39, 40, 40, 40, 41, 41, 41, 43, 44, 44, 46, 47, 47,
47, 48,
```

```
/* Size 16x32 */
```

```
32, 31, 31, 31, 31, 31, 31, 31, 33, 35, 37, 37, 37, 37, 38, 42, 31, 31,
31, 31, 31, 31, 31, 31, 33, 35, 37, 37, 37, 37, 39, 42, 31, 31, 31, 31,
31, 31, 31, 32, 33, 35, 38, 38, 38, 38, 39, 42, 31, 31, 31, 31, 31, 31,
31, 32, 34, 36, 38, 38, 38, 38, 40, 42, 31, 31, 31, 31, 31, 31, 31, 32,
34, 36, 38, 38, 38, 38, 40, 42, 31, 31, 31, 31, 31, 31, 31, 32, 34, 36,
38, 38, 38, 38, 40, 42, 31, 31, 31, 31, 31, 31, 31, 32, 34, 36, 38, 38,
38, 38, 40, 42, 31, 31, 31, 31, 31, 31, 31, 32, 34, 36, 38, 38, 38, 38,
40, 42, 31, 31, 31, 31, 32, 32, 32, 32, 34, 36, 39, 39, 39, 39, 40, 42,
30, 31, 31, 32, 32, 32, 32, 32, 34, 37, 39, 39, 39, 39, 40, 42, 30, 31,
31, 32, 32, 32, 32, 33, 35, 37, 40, 40, 40, 40, 41, 42, 30, 31, 31, 32,
32, 32, 32, 33, 35, 37, 40, 40, 40, 40, 41, 42, 30, 31, 31, 32, 32, 32,
32, 33, 35, 37, 40, 40, 40, 40, 41, 42, 30, 31, 31, 32, 32, 32, 32, 33,
35, 37, 40, 40, 40, 40, 41, 42, 31, 31, 32, 32, 33, 33, 33, 33, 35, 38,
40, 40, 40, 40, 41, 43, 32, 32, 33, 33, 34, 34, 34, 34, 36, 39, 41, 41,
41, 41, 42, 44, 33, 33, 34, 35, 35, 35, 35, 35, 37, 40, 42, 42, 42, 42,
43, 44, 33, 34, 35, 35, 36, 36, 36, 36, 38, 40, 43, 43, 43, 43, 44, 45,
33, 34, 35, 35, 36, 36, 36, 36, 38, 40, 43, 43, 43, 43, 44, 45, 33, 34,
35, 35, 36, 36, 36, 36, 38, 40, 43, 43, 43, 43, 44, 45, 33, 34, 35, 35,
36, 36, 36, 36, 38, 40, 43, 43, 43, 43, 44, 45, 34, 35, 36, 37, 37, 37,
37, 37, 39, 42, 44, 44, 44, 44, 45, 45, 35, 36, 37, 38, 38, 38, 38, 39,
41, 43, 45, 45, 45, 45, 46, 46, 36, 37, 38, 39, 39, 39, 39, 40, 42, 44,
47, 47, 47, 47, 47, 47, 37, 38, 39, 40, 40, 40, 40, 41, 43, 45, 47, 47,
47, 47, 47, 47, 37, 38, 39, 40, 40, 40, 40, 41, 43, 45, 47, 47, 47, 47,
47, 47, 37, 38, 39, 40, 40, 40, 40, 41, 43, 45, 47, 47, 47, 47, 47, 47,
37, 38, 39, 40, 40, 40, 40, 41, 43, 45, 47, 47, 47, 47, 47, 47, 39, 39,
40, 41, 41, 41, 41, 42, 43, 45, 47, 47, 47, 47, 47, 48, 40, 41, 41, 42,
42, 42, 42, 42, 44, 45, 47, 47, 47, 47, 47, 48, 42, 42, 42, 43, 43, 43,
43, 43, 44, 46, 47, 47, 47, 47, 48, 48, 42, 42, 42, 43, 43, 43, 43, 43,
44, 46, 47, 47, 47, 47, 48, 48,
```

```
/* Size 32x16 */
```

```
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 31, 32, 33, 33,
33, 33, 33, 34, 35, 36, 37, 37, 37, 37, 39, 40, 42, 42, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34, 34, 34, 35,
36, 37, 38, 38, 38, 38, 39, 41, 42, 42, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 32, 33, 34, 35, 35, 35, 35, 36, 37, 38, 39, 39,
39, 39, 40, 41, 42, 42, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
32, 32, 32, 33, 35, 35, 35, 35, 35, 37, 38, 39, 40, 40, 40, 40, 41, 42,
43, 43, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 33, 34,
35, 36, 36, 36, 36, 37, 38, 39, 40, 40, 40, 40, 41, 42, 43, 43, 31, 31,
31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 33, 34, 35, 36, 36, 36,
```

```

36, 37, 38, 39, 40, 40, 40, 40, 41, 42, 43, 43, 31, 31, 31, 31, 31, 31,
31, 31, 32, 32, 32, 32, 32, 32, 33, 34, 35, 36, 36, 36, 36, 37, 38, 39,
40, 40, 40, 40, 41, 42, 43, 43, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
33, 33, 33, 33, 33, 34, 35, 36, 36, 36, 36, 37, 39, 40, 41, 41, 41, 41,
42, 42, 43, 43, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35,
35, 36, 37, 38, 38, 38, 38, 39, 41, 42, 43, 43, 43, 43, 43, 44, 44, 44,
35, 35, 35, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37, 37, 38, 39, 40, 40,
40, 40, 40, 42, 43, 44, 45, 45, 45, 45, 45, 45, 46, 46, 37, 37, 38, 38,
38, 38, 38, 38, 39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43, 43, 43, 44,
45, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 37, 37, 38, 38, 38, 38, 38,
39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43, 43, 43, 44, 45, 47, 47, 47,
47, 47, 47, 47, 47, 47, 37, 37, 38, 38, 38, 38, 38, 38, 39, 39, 40, 40,
40, 40, 40, 41, 42, 43, 43, 43, 43, 44, 45, 47, 47, 47, 47, 47, 47, 47,
47, 47, 37, 37, 38, 38, 38, 38, 38, 38, 39, 39, 40, 40, 40, 40, 40, 41,
42, 43, 43, 43, 43, 44, 45, 47, 47, 47, 47, 47, 47, 47, 47, 38, 39,
39, 40, 40, 40, 40, 40, 40, 40, 41, 41, 41, 41, 41, 42, 43, 44, 44, 44,
44, 45, 46, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 42, 42, 42, 42, 42, 42,
42, 42, 42, 42, 42, 42, 42, 42, 43, 44, 44, 45, 45, 45, 45, 45, 46, 47,
47, 47, 47, 47, 48, 48, 48, 48,
/* Size 4x16 */
31, 31, 35, 37, 31, 31, 35, 38, 31, 31, 36, 38, 31, 31, 36, 38, 31, 32,
36, 39, 31, 32, 37, 40, 31, 32, 37, 40, 31, 33, 38, 40, 33, 35, 40, 42,
34, 36, 40, 43, 34, 36, 40, 43, 36, 38, 43, 45, 38, 40, 45, 47, 38, 40,
45, 47, 39, 41, 45, 47, 42, 43, 46, 47,
/* Size 16x4 */
31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 34, 36, 38, 38, 39, 42, 31, 31,
31, 31, 32, 32, 32, 33, 35, 36, 36, 38, 40, 40, 41, 43, 35, 35, 36, 36,
36, 37, 37, 38, 40, 40, 40, 43, 45, 45, 45, 46, 37, 38, 38, 38, 39, 40,
40, 40, 42, 43, 43, 45, 47, 47, 47, 47,
/* Size 8x32 */
32, 31, 31, 31, 33, 37, 37, 38, 31, 31, 31, 31, 33, 37, 37, 39, 31, 31,
31, 31, 33, 38, 38, 39, 31, 31, 31, 31, 34, 38, 38, 40, 31, 31, 31, 31,
34, 38, 38, 40, 31, 31, 31, 31, 34, 38, 38, 40, 31, 31, 31, 31, 34, 38,
38, 40, 31, 31, 31, 31, 34, 38, 38, 40, 31, 31, 32, 32, 34, 39, 39, 40,
30, 31, 32, 32, 34, 39, 39, 40, 30, 31, 32, 32, 35, 40, 40, 41, 30, 31,
32, 32, 35, 40, 40, 41, 30, 31, 32, 32, 35, 40, 40, 41, 30, 31, 32, 32,
35, 40, 40, 41, 31, 32, 33, 33, 35, 40, 40, 41, 32, 33, 34, 34, 36, 41,
41, 42, 33, 34, 35, 35, 37, 42, 42, 43, 33, 35, 36, 36, 38, 43, 43, 44,
33, 35, 36, 36, 38, 43, 43, 44, 33, 35, 36, 36, 38, 43, 43, 44, 33, 35,
36, 36, 38, 43, 43, 44, 34, 36, 37, 37, 39, 44, 44, 45, 35, 37, 38, 38,
41, 45, 45, 46, 36, 38, 39, 39, 42, 47, 47, 47, 37, 39, 40, 40, 43, 47,
47, 47, 37, 39, 40, 40, 43, 47, 47, 47, 37, 39, 40, 40, 43, 47, 47, 47,
37, 39, 40, 40, 43, 47, 47, 47, 39, 40, 41, 41, 43, 47, 47, 47, 40, 41,
42, 42, 44, 47, 47, 47, 42, 42, 43, 43, 44, 47, 47, 48, 42, 42, 43, 43,
44, 47, 47, 48,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 31, 32, 33, 33,
33, 33, 33, 34, 35, 36, 37, 37, 37, 37, 39, 40, 42, 42, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 35, 35, 35, 35, 36,
37, 38, 39, 39, 39, 39, 40, 41, 42, 42, 31, 31, 31, 31, 31, 31, 31,

```

```

32, 32, 32, 32, 32, 32, 33, 34, 35, 36, 36, 36, 36, 37, 38, 39, 40, 40,
40, 40, 41, 42, 43, 43, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
32, 32, 33, 34, 35, 36, 36, 36, 36, 37, 38, 39, 40, 40, 40, 40, 41, 42,
43, 43, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 36,
37, 38, 38, 38, 38, 39, 41, 42, 43, 43, 43, 43, 43, 44, 44, 44, 37, 37,
38, 38, 38, 38, 38, 38, 39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43, 43,
43, 44, 45, 47, 47, 47, 47, 47, 47, 47, 47, 47, 37, 37, 38, 38, 38, 38,
38, 38, 39, 39, 40, 40, 40, 40, 40, 41, 42, 43, 43, 43, 43, 44, 45, 47,
47, 47, 47, 47, 47, 47, 47, 47, 38, 39, 39, 40, 40, 40, 40, 40, 40,
41, 41, 41, 41, 41, 42, 43, 44, 44, 44, 44, 45, 46, 47, 47, 47, 47,
47, 47, 48, 48 },
},
{
  /* Luma */
  /* Size 4x4 */
  31, 31, 31, 32, 31, 32, 32, 32, 31, 32, 32, 32, 32, 32, 32, 33,
  /* Size 8x8 */
  31, 31, 31, 31, 31, 31, 32, 32, 31, 32, 32, 32, 32, 32, 32, 31, 32,
  32, 32, 32, 32, 32, 32, 31, 32, 32, 32, 32, 32, 32, 32, 31, 32, 32, 32,
  32, 32, 32, 32, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  33, 33, 32, 32, 32, 32, 32, 32, 33, 33,
  /* Size 16x16 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 31, 31, 31, 31,
  31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 31, 32, 32, 32, 32, 32,
  32, 32, 32, 33, 33, 33, 33, 33, 31, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 33, 33, 33, 33, 33, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,
  33, 33, 33, 33,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32,
  32, 32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31,
  31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,

```

[illegible]

[illegible]



[illegible]

```

32, 32, 33, 33, 31, 32, 32, 32, 32, 32, 33, 33, 31, 32, 32, 32, 32, 32,
33, 33, 32, 32, 32, 32, 32, 32, 33, 34, 32, 32, 32, 32, 32, 33, 34,
32, 32, 32, 32, 32, 32, 33, 34, 32, 32, 32, 32, 32, 33, 34, 32, 32,
32, 32, 32, 32, 33, 34, 32, 32, 32, 32, 32, 33, 34, 32, 32, 32, 32,
32, 32, 33, 34,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 31,
31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31,
31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33,
33, 33, 33, 33, 33, 33, 33, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34,
34, 34, 34, 34 },
{ /* Chroma */
  /* Size 4x4 */
  31, 31, 31, 34, 31, 31, 31, 35, 31, 31, 32, 35, 34, 35, 35, 39,
  /* Size 8x8 */
  31, 31, 31, 31, 30, 31, 33, 33, 31, 31, 31, 31, 31, 32, 34, 34, 31, 31,
  31, 31, 31, 32, 34, 34, 31, 31, 31, 31, 31, 32, 35, 35, 30, 31, 31, 31,
  32, 32, 35, 35, 31, 32, 32, 32, 32, 33, 36, 36, 33, 34, 34, 35, 35, 36,
  39, 39, 33, 34, 34, 35, 35, 36, 39, 39,
  /* Size 16x16 */
  32, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 33, 33, 33, 33, 31, 31,
  31, 31, 31, 31, 31, 31, 30, 30, 30, 32, 33, 34, 34, 34, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34, 34, 31, 31, 31, 31, 31, 31,
  31, 31, 31, 32, 34, 34, 34, 34, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
  31, 32, 34, 34, 34, 34, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33,
  34, 35, 35, 35, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 35,
  35, 35, 30, 30, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35,
  30, 30, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 30, 30,
  31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 31, 32, 32, 32,
  32, 32, 33, 33, 33, 33, 33, 34, 36, 37, 37, 37, 33, 33, 33, 34, 34, 34,
  34, 34, 34, 34, 34, 36, 37, 38, 38, 38, 33, 34, 34, 34, 34, 34, 35, 35,
  35, 35, 35, 37, 38, 39, 39, 39, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35,
  35, 37, 38, 39, 39, 39, 33, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 37,
  38, 39, 39, 39,
  /* Size 32x32 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30,
  30, 30, 30, 31, 31, 32, 33, 33, 33, 33, 33, 33, 33, 34, 31, 31, 31, 31,
  31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 30, 30, 31,
  31, 32, 33, 33, 33, 33, 33, 33, 33, 34, 31, 31, 31, 31, 31, 31, 31,

```

[illegible]

```

35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 37, 37, 38, 39, 39, 39, 39, 39,
39, 40, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 36, 36,
36, 36, 36, 36, 36, 37, 37, 38, 39, 40, 40, 40, 40, 40, 40, 40,
/* Size 4x8 */
31, 31, 31, 34, 31, 31, 31, 35, 31, 31, 31, 35, 31, 32, 32, 36, 31, 32,
32, 36, 31, 33, 33, 37, 34, 36, 36, 40, 34, 36, 36, 40,
/* Size 8x4 */
31, 31, 31, 31, 31, 31, 34, 34, 31, 31, 31, 32, 32, 33, 36, 36, 31, 31,
31, 32, 32, 33, 36, 36, 34, 35, 35, 36, 36, 37, 40, 40,
/* Size 8x16 */
32, 31, 31, 31, 31, 31, 33, 35, 31, 31, 31, 31, 31, 31, 33, 36, 31, 31,
31, 31, 31, 31, 34, 36, 31, 31, 31, 31, 31, 31, 34, 37, 31, 31, 31, 31,
31, 31, 34, 37, 31, 31, 31, 31, 31, 31, 31, 34, 37, 31, 31, 31, 32, 32, 32,
34, 37, 30, 31, 31, 32, 32, 32, 34, 38, 30, 31, 32, 32, 32, 32, 35, 38,
30, 31, 32, 32, 32, 32, 35, 38, 30, 31, 32, 32, 32, 32, 35, 38, 31, 32,
33, 33, 33, 33, 36, 39, 33, 34, 34, 35, 35, 35, 37, 40, 33, 34, 35, 36,
36, 36, 38, 41, 33, 34, 35, 36, 36, 36, 38, 41, 33, 34, 35, 36, 36, 36,
38, 41,
/* Size 16x8 */
32, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 31, 33, 33, 33, 33, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 34, 34, 34, 34, 31, 31, 31, 31,
31, 31, 31, 31, 32, 32, 32, 33, 34, 35, 35, 35, 31, 31, 31, 31, 31, 31,
32, 32, 32, 32, 32, 33, 35, 36, 36, 36, 31, 31, 31, 31, 31, 31, 32, 32,
32, 32, 32, 33, 35, 36, 36, 36, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
32, 33, 35, 36, 36, 36, 33, 33, 34, 34, 34, 34, 34, 34, 35, 35, 35, 36,
37, 38, 38, 38, 35, 36, 36, 37, 37, 37, 37, 38, 38, 38, 38, 39, 40, 41,
41, 41,
/* Size 16x32 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 35, 37, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 35, 37, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 36, 37, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 32, 33, 35, 36, 38, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 32, 34, 35, 36, 38, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 33, 34, 35, 37, 38, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
34, 35, 37, 38, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 35,
37, 38, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 35, 37, 38,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 35, 37, 38, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 34, 35, 37, 38, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 33, 34, 35, 37, 38, 31, 31, 31, 31, 31, 32,
32, 32, 32, 32, 32, 33, 34, 36, 37, 39, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32,
32, 32, 32, 33, 34, 36, 37, 39, 30, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32,
32, 33, 34, 36, 38, 39, 30, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33,
35, 36, 38, 40, 30, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 35, 36,
38, 40, 30, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 35, 36, 38, 40,
30, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 35, 36, 38, 40, 30, 31,
31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 35, 36, 38, 40, 30, 31, 31, 31,
32, 32, 32, 32, 32, 32, 32, 33, 35, 36, 38, 40, 31, 31, 31, 32, 32, 33,
33, 33, 33, 33, 33, 34, 35, 37, 38, 40, 31, 32, 32, 33, 33, 33, 33, 33, 33,
33, 33, 33, 35, 36, 37, 39, 41, 32, 32, 33, 33, 34, 34, 34, 34, 34, 34,
34, 35, 37, 38, 40, 41, 33, 33, 34, 34, 34, 35, 35, 35, 35, 35, 36,

```

```

37, 39, 40, 42, 33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40,
41, 43, 33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43,
33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 33, 34,
34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 33, 34, 34, 35,
35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 33, 34, 34, 35, 35, 36,
36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 34, 34, 35, 35, 36, 36, 36, 36,
36, 36, 36, 38, 39, 40, 42, 44,
/* Size 32x16 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30,
30, 30, 30, 31, 31, 32, 33, 33, 33, 33, 33, 33, 34, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
32, 32, 33, 34, 34, 34, 34, 34, 34, 34, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34,
34, 34, 34, 34, 34, 35, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
35, 35, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 33, 34, 34, 35, 35, 35, 35, 35, 35, 36, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 34, 35, 36, 36, 36, 36, 36, 36, 36, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34,
35, 36, 36, 36, 36, 36, 36, 36, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 35, 36, 36,
36, 36, 36, 36, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 35, 36, 36, 36, 36, 36, 36,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
32, 32, 32, 33, 33, 34, 35, 36, 36, 36, 36, 36, 36, 36, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33,
33, 34, 35, 36, 36, 36, 36, 36, 36, 36, 36, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 35, 35, 36, 37,
37, 37, 37, 37, 37, 38, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34,
34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 36, 37, 37, 38, 38, 38, 38, 38,
38, 39, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 36, 36,
36, 36, 36, 36, 36, 37, 37, 38, 39, 40, 40, 40, 40, 40, 40, 40, 35, 35,
36, 36, 36, 37, 37, 37, 37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38,
38, 38, 39, 40, 40, 41, 41, 41, 41, 41, 41, 42, 37, 37, 37, 38, 38, 38,
38, 38, 38, 38, 38, 38, 39, 39, 39, 40, 40, 40, 40, 40, 40, 40, 41, 41,
42, 43, 43, 43, 43, 43, 43, 44,
/* Size 4x16 */
31, 31, 31, 34, 31, 31, 31, 34, 31, 31, 31, 35, 31, 31, 31, 35, 31, 31,
31, 35, 31, 31, 31, 35, 31, 32, 32, 36, 31, 32, 32, 36, 31, 32, 32, 36,
31, 32, 32, 36, 31, 32, 32, 36, 32, 33, 33, 37, 33, 35, 35, 39, 34, 36,
36, 40, 34, 36, 36, 40, 34, 36, 36, 40,
/* Size 16x4 */
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 33, 34, 34, 34, 31, 31,
31, 31, 31, 31, 32, 32, 32, 32, 32, 33, 35, 36, 36, 36, 31, 31, 31, 31,
31, 31, 32, 32, 32, 32, 32, 33, 35, 36, 36, 36, 34, 34, 35, 35, 35, 35,
36, 36, 36, 36, 36, 37, 39, 40, 40, 40,
/* Size 8x32 */
32, 31, 31, 31, 31, 31, 33, 35, 31, 31, 31, 31, 31, 31, 31, 33, 35, 31, 31,
31, 31, 31, 31, 33, 36, 31, 31, 31, 31, 31, 31, 33, 36, 31, 31, 31, 31,

```

```

31, 31, 34, 36, 31, 31, 31, 31, 31, 31, 34, 37, 31, 31, 31, 31, 31, 31,
34, 37, 31, 31, 31, 31, 31, 31, 34, 37, 31, 31, 31, 31, 31, 31, 34, 37,
31, 31, 31, 31, 31, 31, 34, 37, 31, 31, 31, 31, 31, 31, 34, 37, 31, 31,
31, 31, 31, 31, 34, 37, 31, 31, 31, 32, 32, 32, 34, 37, 31, 31, 31, 32,
32, 32, 34, 37, 30, 31, 31, 32, 32, 32, 34, 38, 30, 31, 32, 32, 32, 32,
35, 38, 30, 31, 32, 32, 32, 32, 35, 38, 30, 31, 32, 32, 32, 32, 35, 38,
30, 31, 32, 32, 32, 32, 35, 38, 30, 31, 32, 32, 32, 32, 35, 38, 30, 31,
32, 32, 32, 32, 35, 38, 31, 31, 32, 33, 33, 33, 35, 38, 31, 32, 33, 33,
33, 33, 36, 39, 32, 33, 34, 34, 34, 34, 37, 40, 33, 34, 34, 35, 35, 35,
37, 40, 33, 34, 35, 36, 36, 36, 38, 41, 33, 34, 35, 36, 36, 36, 38, 41,
33, 34, 35, 36, 36, 36, 38, 41, 33, 34, 35, 36, 36, 36, 38, 41, 33, 34,
35, 36, 36, 36, 38, 41, 33, 34, 35, 36, 36, 36, 38, 41, 34, 35, 36, 36,
36, 36, 39, 42,
/* Size 32x8 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 30, 30, 30,
30, 30, 30, 31, 31, 32, 33, 33, 33, 33, 33, 33, 33, 34, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
32, 33, 34, 34, 34, 34, 34, 34, 34, 35, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 35,
35, 35, 35, 35, 35, 36, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 35, 36, 36, 36, 36,
36, 36, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32,
32, 32, 32, 32, 32, 33, 33, 34, 35, 36, 36, 36, 36, 36, 36, 36, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
32, 33, 33, 34, 35, 36, 36, 36, 36, 36, 36, 36, 33, 33, 33, 33, 34, 34,
34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 36, 37,
37, 38, 38, 38, 38, 38, 38, 39, 35, 35, 36, 36, 36, 37, 37, 37, 37, 37,
37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 39, 40, 40, 41, 41, 41,
41, 41, 41, 42 },

```

```

},
{

```

```

{ /* Luma */
/* Size 4x4 */
31, 31, 31, 31, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32,
/* Size 8x8 */
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 32, 32, 32, 32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 31, 31, 32, 32,
32, 32, 32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 31, 31, 32, 32, 32, 32,
32, 32, 31, 31, 32, 32, 32, 32, 32, 32, 32,
/* Size 16x16 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31,
31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31,

```

[illegible]

[illegible]



[illegible]

```
/* Size 32x16 */
```

[illegible]

```
/* Size 4x16 */
```

31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 31, 32,  
32, 32, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32,

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
/* Size 4x16 */
```

31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,  
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,  
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 31, 31,  
32, 32, 31, 31, 32, 32, 30, 31, 32, 32,

```
/* Size 16x4 */
```

31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 30, 31, 31,  
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,  
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 31, 31, 31, 31, 31, 31,  
31, 31, 31, 31, 31, 31, 32, 32, 32, 32,

```
/* Size 8x32 */
```

[illegible]

```
/* Size 32x8 */
```

[illegible] $\},$ 

{

```
{ /* Luma */
```

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32 } ,

```

```

}

```

```

}

```

# Annex A: Profiles and levels

## A.1. General

Profiles and levels specify restrictions on the capabilities needed to decode the bitstreams.

The profile specifies the bit depth and subsampling formats supported, while the level defines resolution and performance characteristics.

## A.2. Profiles

There are three named profiles:

- “Main” compliant decoders must be able to decode streams with seq\_profile equal to 0.
- “High” compliant decoders must be able to decode streams with seq\_profile less than or equal to 1.
- “Professional” compliant decoders must be able to decode streams with seq\_profile less than or equal to 2.

**Note:** The Main profile supports YUV 4:2:0 or monochrome bitstreams with bit depth equal to 8 or 10. The High profile further adds support for 4:4:4 bitstreams with the same bit depth constraints. Finally, the Professional profile extends support over the High profile to also bitstreams with bit depth equal to 12, and also adds support for the 4:2:2 video format.

## A.3. Levels

Each operating point contains a syntax element seq\_level\_idx.

The following table defines the mapping from the syntax element (which takes integer values) to the defined levels:

Value of seq_level_idx	Level
0	2.0
1	2.1
2	2.2
3	2.3
4	3.0
5	3.1
6	3.2
7	3.3
8	4.0
9	4.1



Value of seq_level_idx	Level
10	4.2
11	4.3
12	5.0
13	5.1
14	5.2
15	5.3
16	6.0
17	6.1
18	6.2
19	6.3
20	7.0
21	7.1
22	7.2
23	7.3
24-30	Reserved
31	Maximum parameters

**Note:** The level uses a X.Y format. X is equal to  $2 + (\text{seq\_level\_idx} \gg 2)$ . Y is given by  $(\text{seq\_level\_idx} \& 3)$ .

The level defines variables as specified in the following tables:

Level	MaxPicSize (Samples)	MaxHSize (Samples)	MaxVSize (Samples)	MaxDisplayRate (Samples/sec)	MaxDecodeRate (Samples/sec)
2.0	147456	2048	1152	4,423,680	5,529,600
2.1	278784	2816	1584	8,363,520	10,454,400
3.0	665856	4352	2448	19,975,680	24,969,600
3.1	1065024	5504	3096	31,950,720	39,938,400
4.0	2359296	6144	3456	70,778,880	77,856,768
4.1	2359296	6144	3456	141,557,760	155,713,536
5.0	8912896	8192	4352	267,386,880	273,715,200

Level	MaxPicSize	MaxHSize	MaxVSize	MaxDisplayRate	MaxDecodeRate
	(Samples)	(Samples)	(Samples)	(Samples/sec)	(Samples/sec)
5.1	8912896	8192	4352	534,773,760	547,430,400
5.2	8912896	8192	4352	1,069,547,520	1,094,860,800
5.3	8912896	8192	4352	1,069,547,520	1,176,502,272
6.0	35651584	16384	8704	1,069,547,520	1,176,502,272
6.1	35651584	16384	8704	2,139,095,040	2,189,721,600
6.2	35651584	16384	8704	4,278,190,080	4,379,443,200
6.3	35651584	16384	8704	4,278,190,080	4,706,009,088

**Note:** The missing entries in these tables (for example level 2.2 and 7.0) represent levels that are not yet defined.

Level	MaxHeaderRate	MainMbps	HighMbps	MinCompBasis	MaxTiles	MaxTileCols	Example
	(/sec)	(MBits/sec)	(MBits/sec)				
2.0	150	1.5	-	2	8	4	426x240@30fps
2.1	150	3.0	-	2	8	4	640x360@30fps
3.0	150	6.0	-	2	16	6	854x480@30fps
3.1	150	10.0	-	2	16	6	1280x720@30fps
4.0	300	12.0	30.0	4	32	8	1920x1080@30fps
4.1	300	20.0	50.0	4	32	8	1920x1080@60fps
5.0	300	30.0	100.0	6	64	8	3840x2160@30fps
5.1	300	40.0	160.0	8	64	8	3840x2160@60fps
5.2	300	60.0	240.0	8	64	8	3840x2160@120fps
5.3	300	60.0	240.0	8	64	8	3840x2160@120fps
6.0	300	60.0	240.0	8	128	16	7680x4320@30fps
6.1	300	100.0	480.0	8	128	16	7680x4320@60fps
6.2	300	160.0	800.0	8	128	16	7680x4320@120fps
6.3	300	160.0	800.0	8	128	16	7680x4320@120fps

**Note:** Examples are given for non-scalable cases, but the constraints also apply to each operating point of a scalable stream. For example, consider a 60fps spatial scalable stream with a base layer at 960x540, and an enhancement layer at 1920x1080. The operating point containing just the base layer may be labelled as level 3.0, while the operating point containing both the base and enhancement layer may be labelled as level 4.1.

**Note:** HighMbps values are not defined for levels below level 4.0. seq\_tier equal to 1 can only be signaled for level 4.0 and above.

The bitstream constraints depend on the variables in the table, and additional variables derived as follows:

- TileWidth is defined as  $(\text{MiColEnd} - \text{MiColStart}) * \text{MI\_SIZE}$
- TileHeight is defined as  $(\text{MiRowEnd} - \text{MiRowStart}) * \text{MI\_SIZE}$
- CroppedTileWidth is defined as  $\text{FrameWidth} - \text{MiColStart} * \text{MI\_SIZE}$
- CroppedTileHeight is defined as  $\text{FrameHeight} - \text{MiRowStart} * \text{MI\_SIZE}$
- MaxTileSizeInLumaSamples is defined as the largest product of TileWidth \* TileHeight for all tiles within the coded video sequence
- showCount for a particular frame is defined as the number of times the frame is shown (either due to show\_frame equal to 1, or via the show\_existing\_frame mechanism)
- TotalDisplayLumaSampleRate is defined as the sum of the UpscaledWidth \* FrameHeight \* showCount of all frames that belong to the temporal unit that belongs to the operating point, divided by the time difference between the display time of the first frame of the current temporal unit and the display time of the first frame of the next temporal unit (if present). For the last temporal unit in the bitstream, the time difference from the previous temporal unit is used. In particular, for spatial and quality scalability, this limit applies to output pictures that belong to this particular layer. For temporal scalability, this restriction applies to the output pictures that belong to the indicated scalability layer and to the layers below.
- TotalDecodedLumaSampleRate is defined as the sum of the UpscaledWidth \* FrameHeight of all frames with show\_existing\_frame equal to 0 that belong to the temporal unit that belongs to the operating point, divided by the time difference between the decoding time of the first frame of the current temporal unit and the decoding time of the first frame of the next temporal unit (if present). For the last temporal unit in the bitstream, the time difference from the previous temporal unit is used.
- NumFrameHeadersSec is defined as the number of OBU\_FRAME and OBU\_FRAME\_HEADER OBUs received per second (this number does not include duplicate frame headers or frame headers with show\_existing\_frame equal to 1)
- CompressedSize is defined for each frame as the total bytes in the OBUs related to this frame (OBU\_FRAME, OBU\_FRAME\_HEADER, OBU\_METADATA, OBU\_TILE\_GROUP), minus 128 (to allow for overhead of metadata and header data)

- If `mono_chrome` is equal to 1, `ChromaOverhead` (representing the number of chroma samples for every 4 luma samples) is set equal to 0, otherwise `ChromaOverhead` is set equal to 8 >> (`subsampling_x` + `subsampling_y`)
- `UnCompressedSize` is defined as  $\text{MaxPicSize} * \text{BitDepth} * (4 + \text{ChromaOverhead}) / 32$
- `SpeedAdj` is defined as  $\text{TotalDecodedLumaSampleRate} \div \text{MaxDisplayRate}$
- `MinPicCompressRatio` defined as  $\text{Max}(0.8, \text{MinCompBasis} * \text{SpeedAdj} * \text{!still\_picture})$
- `CompressedRatio` is defined as  $\text{UnCompressedSize} \div \text{CompressedSize}$

**Note:** The  $\div$  operator represents standard mathematical division (in contrast to the  $/$  operator which represents integer division).

If `scalability_mode_idc` is not present or equal to a reserved value, then `TemporalParallelNum` and `TemporalParallelDen` are defined to be equal to 1.

Otherwise, if `scalability_mode_idc` is not equal to `SCALABILITY_SS`, `TemporalParallelDen` is defined as 1 and `TemporalParallelNum` is defined as  $(1 << (\text{TemporalLayers} - 1))$  where `TemporalLayers` is the number of temporal layers as defined in the semantics for `scalability_mode_idc`.

Otherwise (`scalability_mode_idc` is equal to `SCALABILITY_SS`), `TemporalParallelNum` and `TemporalParallelDen` are defined as follows:

```
NumIndependent = 0
for ( i = 0; i < temporal_group_size; i++ ) {
    if ( temporal_group_temporal_id[ i ] ) {
        independent = 1
        for ( j = 0; j < temporal_group_ref_cnt[ i ]; j++ ) {
            ref = (i - temporal_group_ref_pic_diff[ i ][ j ] + temporal_group_size) % temporal_group_size
            if ( temporal_group_temporal_id[ ref ] == temporal_group_temporal_id[ i ] )
                independent = 0
        }
        NumIndependent += independent
    }
}
TemporalParallelNum = temporal_group_size
TemporalParallelDen = temporal_group_size - NumIndependent
```

When the mapped level is contained in the tables above, it is a requirement of bitstream conformance that the following constraints hold:

- `UpscaledWidth * FrameHeight` is less than or equal to `MaxPicSize`
- `UpscaledWidth` is less than or equal to `MaxHSize`
- `FrameHeight` is less than or equal to `MaxVSize`

- TotalDisplayLumaSampleRate is less than or equal to MaxDisplayRate
- TotalDecodedLumaSampleRate is less than or equal to MaxDecodeRate
- NumFrameHeadersSec is less than or equal to MaxHeaderRate
- The number of tiles per second is less than or equal to MaxTiles \* 120
- NumTiles is less than or equal to MaxTiles
- TileCols is less than or equal to MaxTileCols
- CompressedRatio is greater than or equal to MinPicCompressRatio
- TileWidth is less than or equal to 4096 for each tile
- TileWidth \* TileHeight is less than or equal to 4096 \* 2304 for each tile
- FrameWidth is greater than or equal to 16
- FrameHeight is greater than or equal to 16
- CroppedTileWidth is greater than or equal to 8 for each tile
- CroppedTileHeight is greater than or equal to 8 for each tile
- MaxTileSizeInLumaSamples \* NumFrameHeadersSec \* TemporalParallelDen/TemporalParallelNum is less than or equal to 588,251,136 (where this number is the decode luma sample rate of 4096x2176 \* 60fps \* 1.1)

**Note:** The purpose of this constraint is to ensure that for decode luma sample rates above 4K60 there is sufficient parallelism for decoder implementations. Parallelism can be chosen by the encoder as either tile level parallelism or temporal layer parallelism or a combination provided the above constraint holds. The constraint has no effect on levels 5.1 and below.

If seq\_level\_idx is equal to 31 (indicating the maximum parameters level), then there are no level-based constraints on the bitstream.

**Note:** The maximum parameters level should only be set for bitstreams that do not conform to any other level. Typically this would be used for large resolution still images.

The buffer model is used to define additional conformance requirements.

These requirements depend on the following level, tier, and profile dependent variables:

- If seq\_tier is equal to 0, MaxBitrate is equal to MainMbps multiplied by 1,000,000
- Otherwise (seq\_tier is equal to 1), MaxBitrate is equal to HighMbps multiplied by 1,000,000

- `MaxBufferSize` is equal to `MaxBitrate` multiplied by 1 second
- If `seq_profile` is equal to 0, `BitrateProfileFactor` is equal to 1.0
- If `seq_profile` is equal to 1, `BitrateProfileFactor` is equal to 2.0
- If `seq_profile` is equal to 2, `BitrateProfileFactor` is equal to 3.0

## A.4. Decoder Conformance

A level X.Y compliant decoder must be able to decode all bitstreams (that can be decoded by the general decoding process) that conform to that level.

When doing that, the decoder must be able to display output frames according to the display schedule if such is indicated by the bitstream.

**Note:** If the level of a bitstream is equal to 31 (indicating the maximum parameters level), the decoder should examine the properties of the bitstream and decide whether to decode it or not. There is no assurance that all pictures will be decoded. A decoder would typically decode pictures up to a certain maximum uncompressed picture size (or maximum compressed picture size or maximum width or maximum tile size) that the decoder maker considers sufficiently extreme for their use case, and not decode anything bigger than that.

A level X.Y compliant decoder should be able to decode tile list OBUs (via the large scale tile decoding process) at a rate of 90 tile list OBUs per second subject to the following level-dependent constraints:

- $\text{UpscaledWidth} * \text{FrameHeight}$  is less than or equal to `MaxPicSize`
- `UpscaledWidth` is less than or equal to `MaxHSize`
- `FrameHeight` is less than or equal to `MaxVSize`
- $\text{TileWidth} * \text{TileHeight} * (\text{tile\_count\_minus\_1} + 1) * 90$  is less than or equal to  $(\text{MaxDecodeRate} / 2)$
- For each tile list OBU,  $\text{BytesPerTileList} * 8 * 90$  is less than or equal to `MaxBitrate`

Where `BytesPerTileList` is defined as the sum of  $(\text{coded\_tile\_data\_size\_minus\_1} + 1)$  for each tile list entry in the tile list OBU.

# Annex B: Length delimited bitstream format

## B.1. Overview

[Section 5](#) defines the syntax for OBUs. [Section 5.2](#) defines the low-overhead bitstream format. This annex defines a length-delimited format for packing OBUs into a format that enables skipping through temporal units and frames more easily.

## B.2. Length delimited bitstream syntax

bitstream( ) {	Type
while ( more_data_in_bitstream() ) {	
temporal_unit_size	leb128()
temporal_unit( temporal_unit_size )	
}	
}	

temporal_unit( sz ) {	Type
while ( sz > 0 ) {	
frame_unit_size	leb128()
sz -= Leb128Bytes	
frame_unit( frame_unit_size )	
sz -= frame_unit_size	
}	
}	

frame_unit( sz ) {	Type
while ( sz > 0 ) {	
obu_length	leb128()
sz -= Leb128Bytes	
open_bitstream_unit( obu_length )	
sz -= obu_length	
}	
}	

## B.3. Length delimited bitstream semantics

**more\_data\_in\_bitstream()** is a system-dependent method of determining whether the end of the bitstream has been reached. It returns 1 when there is more data to be read, or 0 when the end of the bitstream has been reached.

**temporal\_unit\_size** specifies the length in bytes of the next temporal unit.

**frame\_unit\_size** specifies the length in bytes of the next frame unit.

**obu\_length** specifies the length in bytes of the next OBU.

**Note:** It is allowed for the OBU to set `obu_has_size_field` equal to 1 to indicate that the `obu_size` syntax element is present. In this case, the decoding process assumes that `obu_size` and `obu_length` are set consistently. If `obu_size` and `obu_length` are both present, but inconsistent, then the packed bitstream is deemed invalid.

The first OBU in the first `frame_unit` of each `temporal_unit` must be a temporal delimiter OBU (and this is the only place temporal delimiter OBUs can appear).

All the frame header and tile group OBUs required for decoding a single frame must be within the same `frame_unit` (and a `frame_unit` must not contain frame headers for more than one frame).



# Annex C: Error resilience behavior (informative)

## C.1. General

This annex defines additional starting points for decoding.

It is recommended that decoders should support these starting points. (This annex is marked as informative because it is not mandatory for a conformant decoder to support these starting points.)

The intention is to allow decoders to start even when the decoded output may be corrupted.

A property of a bitstream is defined in [section C.2](#).

The recommendations are expressed in [section C.3](#).

The consequences for encoders are specified in [section C.4](#).

The consequences for decoders are specified in [section C.5](#).

## C.2. Definition of processable frames

This section defines a property of frames that is called being “processable”.

Informally, a frame is processable if it is certain (based on the current state and information in the frame header) that everything other than the sample values can be decoded correctly.

In particular, a frame that is processable will have correct values for:

- All syntax elements
- The size, bitdepth, subsampling structure of any output frames
- All values written in the reference frame update process specified in [section 7.20](#), except for the contents of FrameStore (which may or may not be correct).

In most codecs, this concept is unnecessary because it is trivial to determine if frames are processable (either because all frames are automatically processable, or because the conditions are straightforward). However, AV1 makes greater use of state in the reference frames and so the condition for being processable is more complicated.

Formally, the property of being processable is defined as follows.

A frame with `show_existing_frame` equal to 0 is defined to be processable if the following conditions are met:

- Either `primary_ref_frame` is equal to `PRIMARY_REF_NONE` or `ref_frame_idx[primary_ref_frame]` indicates a frame that has been processed
- Either `use_ref_frame_mvs` is equal to 0, or `ref_frame_idx[i]` indicates a frame that has been processed for all  $i = 0..REFS\_PER\_FRAME-1$

- Either `allow_warped_motion` is equal to 0, or `ref_frame_idx[ i ]` indicates a frame that has been processed for all  $i = 0..REFS\_PER\_FRAME-1$  (this is necessary because the `motion_mode` syntax parsing depends on whether the reference frame has been scaled - but this is not known unless the frame has been processed)
- The decoding process for the frame does not use values in `RefOrderHint` before they have been written (written either by the decoding process for the current frame, or written when a previous frame was processed)
- If the syntax element `found_ref` is equal to 1, `ref_frame_idx[ i ]` indicates a frame that has been processed (this is necessary because the frame dimensions are only correct for processed frames)

A frame with `show_existing_frame` equal to 1 is processable if the following condition is met:

- `frame_to_show_map_idx` indicates a frame that has been processed

(A frame being “processed” means that the frame was processable and has been decoded.)

## C.3. Recommendation for processable frames

It is recommended that decoders should support decoding bitstreams if the first temporal unit contains a sequence header and all frames contained in the bitstream are processable according to the definition above.

As the inter prediction may depend on missing reference frames, there is not a requirement that exactly the same output samples as the reference code are produced.

In certain cases (e.g. when the first frame only contains intra coded blocks), it is possible that correct output is produced, but, in general, error concealment techniques may be required.

## C.4. Encoder consequences of processable frames

If an application chooses to use a non-key frame starting point, then the encoder needs to be careful that the resulting bitstream is processable.

There are some features of the bitstream specification that make this easier to achieve:

- `primary_ref_frame`, `use_ref_frame_mvs`, and `allow_warped_motion` can be controlled at a frame level to satisfy the corresponding conditions
- setting `error_resilient_mode` equal to 1 causes the order hints to be transmitted if necessary
- `found_ref` can be cleared to allow the frame resolution to be sent explicitly

## C.5. Decoder consequences of processable frames

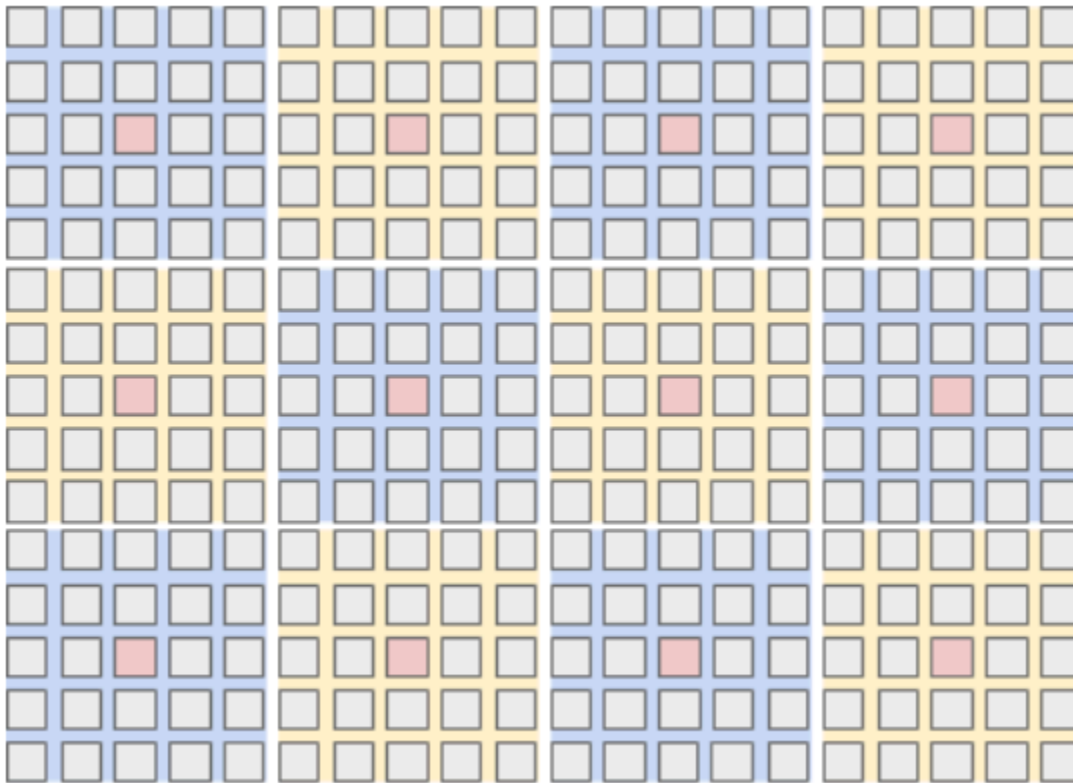
For the decoding process to handle this mode of operation, the following modifications should be used:

- `RefValid[ i ]` should be set equal to 0 for  $i = 0..NUM\_REF\_FRAMES-1$  before the decoding process begins

- If `frame_id_numbers_present_flag` is equal to 1, for the first frame `current_frame_id` should not be compared to `PrevFrameID` (because `PrevFrameID` is uninitialized).
- When the syntax element `ref_order_hint[ i ]` is read, `RefOrderHint[ i ]` is set equal to `ref_order_hint[ i ]`.
- The requirement for bitstream conformance described in the semantics for `ref_frame_idx[ i ]` (that uses `RefValid` to check that the reference frames are available) should be ignored
- When using the inter prediction process, if `RefValid[ refIdx ]` is equal to 0, then the motion vector scaling and block inter prediction processes are not followed. Instead, `preds[ refList ]` should be generated using an alternative approach. For conformance testing, it may help to define the predicted samples in a standard way. The suggested approach is to fill `preds[ refList ]` with neutral gray samples, i.e. all values equal to  $1 \ll (\text{BitDepth} - 1)$ .

## Annex D: Large scale tile use case (informative)

One potential use-case for this process is the representation and rendering of a 3-dimensional environment as seen through an array of cameras that together capture a 360 degree view of the surrounding environment. One such arrangement of cameras is shown in the figure below where each small rectangle represents a full frame captured by a camera placed at a given latitude (row in the diagram) or longitude (column in the diagram), but other variants are possible.



*Example array of 15x20 "camera" frames partitioned into 5x5 sub-arrays, where each sub-array has a central "anchor" frame (in red).  
Each small rectangle represents a frame.*

In order to create a rendered view in an arbitrary direction in the 3-D scene the decoder will potentially need a small number of pixels from many of these frames. Providing a process to decode only a sparsely distributed subset of small regions, each defined as an AV1 tile, can significantly improve efficiency.

The algorithm for constructing the 3-dimensional view from the set of decoded tiles is beyond the scope of this specification; two possible schemes can be found in [8][9].

To reduce the amount of memory required and to improve decoder efficiency, a certain subset of the frames are denoted as "Anchor Frames" and it is assumed that the application makes these frames available to the decoder in uncompressed form. The example presented in the figure above highlights anchor frames in red.

The other (non-anchor) frames are referred to as “Camera Frames” and each one is encoded using only the nearest anchor frame as a reference. Camera frames are highlighted in gray in the figure above, and are shown clustered into 5x5 groups to indicate which anchor frame each uses for prediction.

The application is required to render a new 3-dimensional view at a rate of 90 fps. Each rendered frame is created from a single set of tiles as specified by a `tile_list_obu` as defined in [section 5.12](#).

# Annex E: Decoder model

## E.1. General

The decoder model is used to verify that a bitstream can be decoded within the constraints imposed by one of the coding levels defined in [section A.3](#). The decoder model is also used to verify conformance for a decoder that claims conformance to a certain coding level.

A set of decoder model parameters may be optionally specified for zero or more operating points. If the new Sequence Header OBU does not signal decoder model parameters for a given operating point the previous set of decoder model parameters does not persist.

The decoder model describes the smoothing buffer, decoding process, operation of the frame buffers and the frame output process.

The decoder model is applied to an operating point of a bitstream. Different operating points can have different decoder models that specify conformance to the levels signaled for these operating points.

The decoder model defines two modes of operation. A conformant bitstream shall satisfy constraints imposed by one of these two modes of the decoder model depending on which mode is applicable.

[Section E.2](#) defines additional concepts used by the decoder model.

[Section E.3](#) defines the operating modes.

[Section E.4](#) specifies how the frame timings can be computed in the different operating modes.

[Section E.5](#) specifies the decoder model process.

[Section E.6](#) specifies the conformance requirements.

## E.2. Decoder model definitions

The decoder model uses the following elements to verify bitstream conformance that are not part of the decoding process specified in [section 7](#).

**Note:** The elements defined in this section do not have to be present in a conformant decoder implementation. These elements may be considered examples of elements of a conformant decoder, although the actual decoder implementation may differ.

**BufferPool** is a storage area for a set of frame buffers. Buffer pool area allocated for storing separate frames is defined as `BufferPool[i]`, where `i` takes values from 0 to `BufferPoolMaxSize - 1`. When a frame buffer is used for storing a decoded frame, it is indicated by a VBI slot that points to this frame buffer. `BufferPoolMaxSize` is equal to 10.

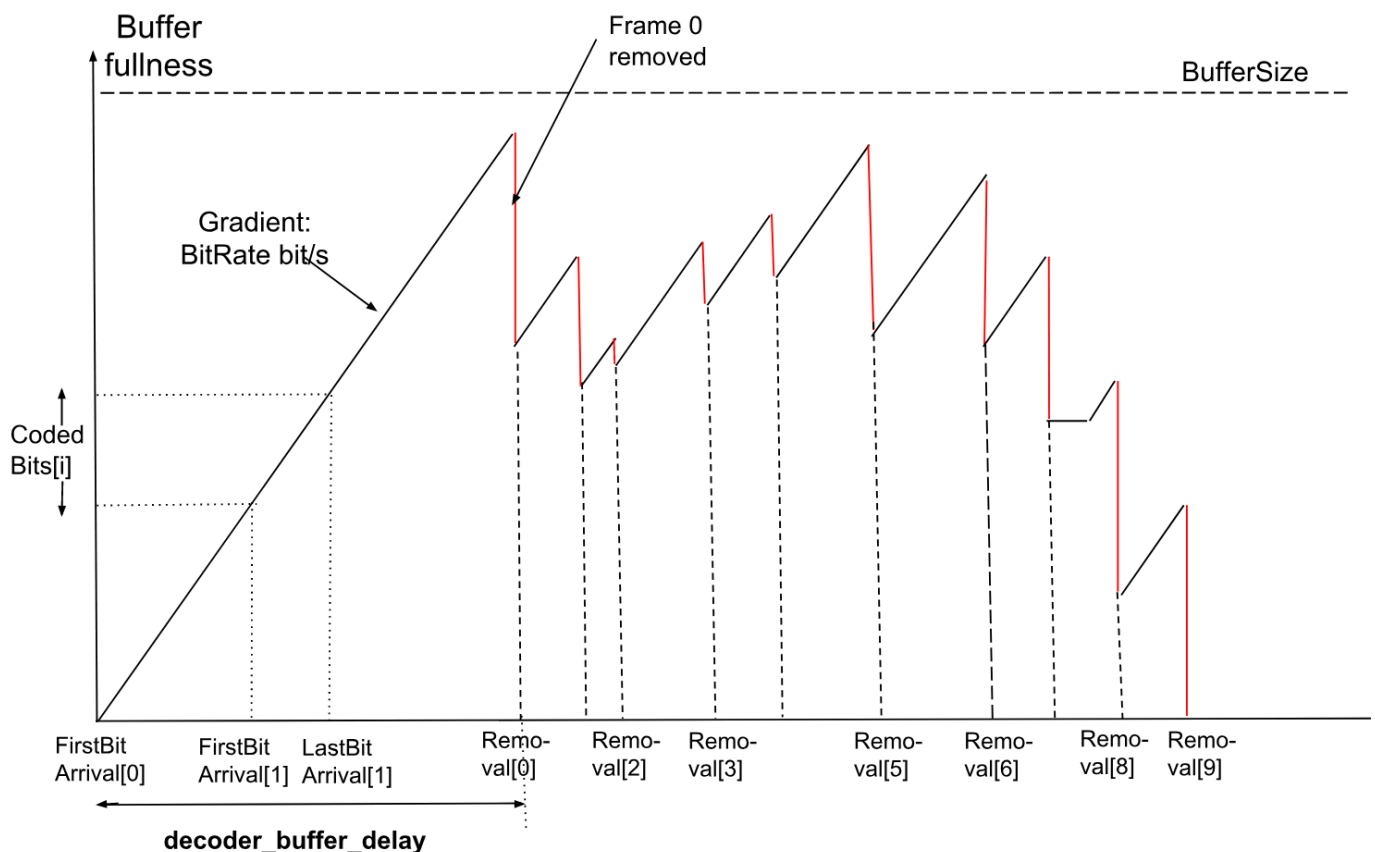
**VBI** (virtual buffer index) is an array of indices of the frame areas in the BufferPool. VBI elements which do not point to any slot in the VBI are set to -1. VBI array size is equal to 8, with the indices taking values from 0 to 7.

**cfbi** (current frame buffer index) is the variable that contains the index to the area in the BufferPool that contains the current frame.

**DecoderRefCount[ i ]** is a variable associated with a frame buffer i. DecoderRefCount[ i ] is initialized to 0, and incremented by 1 each time the decoder adds the buffer i to a VBI index slot. It is decremented by 1 each time the decoder removes the buffer from a VBI index slot i. The decoder may update multiple VBI index slots with the same frame buffer, as specified by refresh\_frame\_flags, so the counter may be incremented several times. The counter is only modified in this way when refresh\_frame\_flags is used to update the VBI index once the frame has been fully decoded. The decoder also increments the counter as it begins the decode process and decrements it again once complete. When the counter is 0 the pixel data becomes permanently invalid and shall not be used by the decode process.

**PlayerRefCount[ i ]** is a variable associated with a frame buffer i. PlayerRefCount[ i ] is initialized to 0, incremented by 1 each time the decoder determines that the frame is a presentation frame. It is reset to 0 after the last time the frame is presented.

**PresentationTimes[ i ]** is an array corresponding to the BufferPool [ i ] that holds the last presentation time for the decoded frame that is kept in the BufferPool [ i ].



*Example of how the coded frame buffer fullness varies as data arrives from the stream, and is subsequently removed for decoding. Relevant timing points and values are indicated.*

Coded frames arrive at the decoder smoothing buffer of the size **BufferSize** at a rate defined by **BitRate**. The following variables are used in this section and below:

**BitRate** is set to a value equal to  $\text{MaxBitrate} * \text{BitrateProfileFactor}$  specified for the level signaled for the operating point that is being decoded.

**BufferSize** is set to a value equal to  $\text{MaxBufferSize} * \text{BitrateProfileFactor}$  value specified for the level signaled for the operating point that is being decoded.

**Decodable Frame Group  $i$**  (DFG  $i$ ) consists of all OBUs, including headers, between the end of the last OBU associated with the previous frame with `show_existing_frame` flag equal to 0 (frame  $k$ ), and the end of the last OBU associated with the current frame with `show_existing_frame` flag equal to 0 (frame  $p$ ). This comprises the OBUs that make up frame  $p$ , plus any additional OBUs present in the bitstream that belong to frame  $p$  (such as the metadata OBU), and OBU that belong to frames with `show_existing_frame` flag equal to 1 which are located between frame  $k$  and frame  $p$ . The decoder model assumes that the decoding time for processing a frame with `show_existing_frame` flag equal to 1, a header, or a metadata OBU is 0, hence the smoothing buffer operates in the units of DFG.

**CodedBits[  $i$  ]** is the amount of data, in bits, that belongs to DFG  $i$ . Note that the index  $i$  of the DFG only increases with frames with `show_existing_frame` flag equal to 0, i.e. frames that need to be decoded by the decoding process.

**FirstBitArrival[  $i$  ]** is the time when the first bit of the  $i$ -th DFG starts entering the decoder smoothing buffer. For the first coded DFG in the sequence, DFG 0 (or after updating decoder model parameters at RAP),  $\text{FirstBitArrival}[0] = 0$ .

**LastBitArrival[  $i$  ]** is the time when the last bit of DFG  $i$  finishes entering the smoothing buffer.

Each **shown frame  $j$**  has a scheduled presentation time, **PresentationTime[  $j$  ]**, defined to be a multiple of the display clock tick **DispCT**.

**DispCT** represents the expected time interval between displaying two consecutive frames, or a common divisor of the expected times between displaying two consecutive frames if the encoded bitstream has a variable display frame rate.

## E.3. Operating modes

### E.3.1. Resource availability mode

In this mode the model simulates the operation of the decoder under the assumption that the complete coded frame is available in the smoothing buffer when decoding of that frame begins. In addition, it is assumed that the decoder will begin to decode a frame immediately after it finishes decoding the previous frame or when a frame buffer becomes available, whichever is later. This model uses the generated time moment, when the decoding of a frame begins, as times when the data is removed from the smoothing buffer to check the conformance of a bitstream to the bitrate specified for a level signaled for the Operating Point of a bitstream.

To verify that a bitstream can be decoded by a decoder under the constraints of a particular level it is assumed that the decoder performs the decoding operations at maximum speed (the minimum time interval) specified for that level in [section A.3](#).

To use Resource Availability mode, the following parameters should be set in the encoded video bitstream:



- `timing_info_present_flag` equal to 1
- `decoder_model_info_present_flag` equal to 0
- `equal_picture_interval` equal to 1

If the parameters listed above are not specified by the bitstream, the parameters necessary to input into this model can be signaled by the application or some other means. If the parameters necessary to run this model are not signaled, it is not possible to check the conformance of the stream to the claimed level.

In this mode of operation, the decoder model parameters below take the following (default) values:

- `encoder_buffer_delay` = 20 000
- `decoder_buffer_delay` = 70 000
- `low_delay_mode_flag[ op ]` = 0

The decoder writes the decoded frame into one of the 10 available frame buffers. Decoding must be delayed until a frame buffer becomes available.

## E.3.2. Decoding schedule mode

This mode imposes additional constraints relating to the operation of the smoothing buffer and the timing points, specified for each frame, defining exactly when the decoder should start decoding a frame and when that frame should be presented.

To use Decoding Schedule Mode, the following parameters should be signaled by the encoded video bitstream:

- `timing_info_present_flag` equal to 1
- `decoder_model_info_present_flag` equal to 1
- `decoder_model_present_for_this_op[ op ]` equal to 1

When these flags are signaled, the bitstream should provide the associated information specified in `decoder_model_info()`, `operating_parameters_info()` and `initial_display_delay_minus_1[ op ]`.

In addition each frame must specify, for operating point `op`, the following parameters:

- `buffer_removal_time[ op ]`
- `frame_presentation_time`

If the parameters listed above are not specified by the bitstream, the parameters necessary to input into this model can be signaled by the application or some other means. If the parameters necessary to run this model are not signaled, it is not possible to check the conformance of the stream to the claimed level with this model.

### E.3.3. When timing information is not present in the bitstream

When the `timing_info()`, and other info necessary as the input to one of the decoder models and associated information is not present in the bitstream, it is impossible to verify whether the bitstream satisfy the levels constraints according to either of the decoder models. In order to enable the verification of the decoder conformance, the equivalent information necessary to verify the bitstream compliance can be provided by some external means.

## E.4. Frame timing definitions

### E.4.1. Start of DFG bits arrival

The bits arrive in the smoothing buffer at a constant bitrate `BitRate` or the bitrate equal to 0. Hence, the average bitrate can be lower than the bitrate `BitRate` specified in the level definition, which, in this case, represents a peak bitrate. The first bit of DFG `i` is expected to arrive by the latest time that would guarantee timely reception of the entire DFG by the time when the decodable frame in the DFG `i` is due to be decoded:

$$\text{FirstBitArrival}[i] = \max(\text{LastBitArrival}[i - 1], \text{LatestArrivalTime}[i]),$$

where `LatestArrivalTime[i]` is the latest time when the first bit of DFG `i` must arrive in the smoothing buffer to ensure that the complete DFG is available at the scheduled removal time, `ScheduledRemoval[i]`, in units of seconds, unless the new set of decoding model parameters is received. In its turn, the latest time the DFG data should start being received is determined as follows:

$$\text{LatestArrivalTime}[i] = \text{ScheduledRemoval}[i] - (\text{encoder\_buffer\_delay} + \text{decoder\_buffer\_delay})$$

90 000

### E.4.2. End of DFG bits arrival

For the bits that belong to the DFG `i`, the time of arrival of the last bit of the DFG `i` is determined as follows:

$$\text{LastBitArrival}[i] = \text{FirstBitArrival}[i] + \text{CodedBits}[i] \div \text{BitRate}$$

### E.4.3. Scheduled removal times

The decoder starts to decode a frame exactly at the moment when the data corresponding to its DFG is removed from the smoothing buffer. Each DFG has a scheduled removal time and an actual removal time. Under certain circumstances these times may be different.

The `ScheduledRemoval[i]` time is determined differently in the resource availability and the decoding schedule mode.

When the decoder model operates in the decoding schedule mode

$$\text{ScheduledRemoval}[i] = \text{ScheduledRemovalTiming}[i]$$

When the decoder model operates in the resource availability mode

$$\text{ScheduledRemoval}[i] = \text{ScheduledRemovalResource}[i]$$

Derivation of  $\text{ScheduledRemovalTiming}[i]$  in the decoding schedule mode is described in [section E.4.4](#), and derivation of  $\text{ScheduledRemovalResource}[i]$  in the resource availability mode is described in [section E.4.5](#).

## E.4.4. Removal times in decoding schedule mode

DFG  $i$  is scheduled for removal from the smoothing buffer at time  $\text{ScheduledRemovalTiming}[i]$  which is defined as an offset,  $\text{buffer\_removal\_time}[i]$ , signaled for the frame of the DFG with `show_existing_frame` flag equal to 0, relative to the moment of time when the first DFG is removed from the smoothing buffer,  $\text{decoder\_buffer\_delay}$ :

$$\text{ScheduledRemovalTiming}[0] = \text{decoder\_buffer\_delay} \div 90\,000$$

$$\text{ScheduledRemovalTiming}[i] = \text{ScheduledRemovalTiming}[0] + \text{buffer\_removal\_time}[i] * \text{DecCT}$$

DFG  $i$  is removed from the smoothing buffer at time  $\text{Removal}[i]$ .

There are two modes of operation of a decoder which determine whether the actual DFG removal time  $\text{Removal}[i]$  may be different from the scheduled DFG removal timing  $\text{ScheduledRemovalTiming}[i]$ . As was mentioned earlier, the decoder starts decoding a frame instantaneously when the data that belongs to its DFG is removed from the smoothing buffer.

In this mode, frame decoding start times / DFG removal times are determined by the  $\text{buffer\_removal\_time}[i]$  for the chosen operating point,  $op$ .

If  $\text{low\_delay\_mode\_flag}[op]$  is equal to 0 the decoder operates in Strict Arrival Mode, and DFG is removed from the smoothing buffer at the scheduled time, that is:

$$\text{Removal}[i] = \text{ScheduledRemovalTiming}[i]$$

Otherwise,  $\text{low\_delay\_mode\_flag}[op]$  is equal to 1 and the decoder operates in Low-Delay Mode, where the DFG data may not be available in the smoothing buffer at the scheduled removal time, i.e.  $\text{ScheduledRemovalTiming}[i] < \text{LastBitArrival}[i]$ . In that case the removal of the DFG is deferred until the first decode clock tick after the complete DFG is present in the smoothing buffer, that is:

$$\text{Removal}[i] = \text{ceil}(\text{LastBitArrival}[i] \div \text{DecCT}) * \text{DecCT}$$

If the entire DFG is available in the smoothing buffer at the scheduled removal time, i.e. `ScheduledRemovalTiming[ i ] >= LastBitArrival[ i ]`, then it is removed at the scheduled time, that is:

```
Removal[ i ] = ScheduledRemovalTiming[ i ]
```

## E.4.5. Removal times in resource availability mode

In the resource availability mode, `buffer_removal_time[ i ]` are not signaled for the chosen operating point. In this mode, timing of the decoder model is driven by the availability of the resources in the decoder, in particular, by times when the decoding of the previous frame with `show_existing_frame` flag equal to 0 has been completed and a free frame buffer is available.

In particular, `ScheduledRemovalResource[ i ]` times are generated as the earliest time that a non-assigned frame buffer becomes available for decoding of the frame `i`. In this mode, the decoder starts to decode a frame as fast as it can after completing decoding of the previous frame and a free frame buffer is available. A frame buffer is defined as being available if it is no longer being used and its content can be overwritten.

Removal times in the resource availability mode are produced by the decode process in [section E.5.2](#)

The following function, `time_next_buffer_is_free`, is used by the decode process to determine the `Removal[ i ]` time for the next DFG and generate the value of `ScheduledRemovalResource[ i ]`.

```
time_next_buffer_is_free ( i, time ) {
    if ( i == 0 ) {
        time = decoder_buffer_delay + 90000
    }
    foundBuffer = 0
    for ( k = 0; k < BufferPoolMaxSize; k++ ) {
        if ( DecoderRefCount[ k ] == 0 ) {
            if ( PlayerRefCount[ k ] == 0 ) {
                ScheduledRemovalResource[ i ] = time
                return time
            }
            if ( !foundBuffer || PresentationTimes[ k ] < bufFreeTime ) {
                bufFreeTime = PresentationTimes[ k ]
                foundBuffer = 1
            }
        }
    }
    ScheduledRemovalResource[ i ] = bufFreeTime
    return bufFreeTime
}
```

## E.4.6. Frame decode timing

The time required to decode a frame (i.e. to process the decodable frame's DFG), `TimeToDecode [ i ]`, is calculated based on the frame type, a maximum number of luma pixels for the frame, and the throughput of the decoder as specified in the definition of the level assigned to the operating point that the frame belongs.

The time that it takes the decoder to decode a frame according to the decoder model is estimated by using the function `time_to_decode_frame( )` as follows.

```
time_to_decode_frame( ) {
    if ( show_existing_frame == 1 ) {
        lumaSamples = 0
    } else if ( frame_type == KEY_FRAME ||
               frame_type == INTRA_ONLY )
        lumaSamples = UpscaledWidth * FrameHeight
    } else {
        if ( spatial_layer_dimensions_present_flag )
            lumaSamples = (spatial_layer_max_width[ spatial_id ]) *
                          (spatial_layer_max_height[ spatial_id ])
        else
            lumaSamples = (max_frame_width_minus_1 + 1) *
                          (max_frame_height_minus_1 + 1)
    }
    return lumaSamples ÷ MaxDecodeRate
}
```

If the `spatial_layer_dimensions_present_flag` syntax element is not present in the coded video sequence, it is taken to be equal to 0.

The `MaxDecodeRate` value is defined in [section A.3](#) for the level signaled for the operating point the decoder has chosen to decode.

## E.4.7. Frame presentation timing

Initial presentation delay is determined as follows:

```
InitialPresentationDelay = Removal [ initial_display_delay_minus_1 ] + TimeToDecode [
initial_display_delay_minus_1 ]
```

When `equal_picture_interval` is equal to 0, the decoder operates in variable frame rate mode, the frame presentation time is defined as follows:

```

PresentationTime[ 0 ] = InitialPresentationDelay

PresentationTime[ j ] = InitialPresentationDelay + ( frame_presentation_time[ j ] -
frame_presentation_time[ 0 ] ) * DispCT

```

When `equal_picture_interval` is equal to 1, the decoder operates in the constant frame rate mode, and the frame presentation time is defined as follows:

```

PresentationTime[ 0 ] = InitialPresentationDelay

PresentationTime[ j ] = PresentationTime[ j - 1 ] + ( num_ticks_per_picture_minus_1 + 1 ) * DispCT

```

where `PresentationTime[ j - 1 ]` refers to the previous frame in presentation order.

The presentation interval, i.e. the time interval between the display of consecutive frames `j` and `j + 1` in presentation order, is defined as follows:

```

PresentationInterval[ j ] = PresentationTime[ j + 1 ] - PresentationTime[ j ]

```

## E.5. Decoder model

### E.5.1. Decoder model functions

This section defines the buffer management functions invoked by the decoder model process.

The `free_buffer` function clears the variables for a particular index in the `BufferPool`.

```

free_buffer( idx ) {
    DecoderRefCount[ idx ] = 0
    PlayerRefCount[ idx ] = 0
    PresentationTimes[ idx ] = -1
}

```

The `initialize_buffer_pool` function resets the `BufferPool` and the `VBI`.

```

initialize_buffer_pool( ) {
    for ( i = 0; i < BufferPoolMaxSize; i++ )
        free_buffer( i )
    for ( i = 0; i < 8; i++ )
        VBI[ i ] = -1
}

```

The `get_free_buffer` function searches for an un-assigned frame in the BufferPool. (The decoder needs an un-assigned frame buffer from the BufferPool for each frame that it decodes.)

```
get_free_buffer( ) {
    for ( i = 0; i < BufferPoolMaxSize; i++ ) {
        if ( DecoderRefCount[ i ] == 0 &&
            PlayerRefCount[ i ] == 0 )
            return i
    }
    return -1
}
```

Once decoded, frames may update one or more of the VBI index slots, as defined by `refresh_frame_flags`. Each time a VBI index slot is updated the decoder reference count is incremented by 1 for the corresponding frame buffer. If the VBI index slot being updated is currently occupied, the decoder reference count for the frame buffer being displaced must be decremented by 1.

The `update_ref_buffers` function updates the VBI and reference counts when the reference frames are updated.

```
update_ref_buffers ( idx, refresh_frame_flags ) {
    for ( i = 0; i < 8; i++ ) {
        if ( refresh_frame_flags & ( 1 << i ) ) {
            if ( VBI[ i ] != -1 )
                DecoderRefCount[ VBI[ i ] ] --
            VBI[ i ] = idx
            DecoderRefCount[ idx ] ++
        }
    }
}
```

In decoding schedule mode the decoder only starts to decode a frame at the time designated by a Removal time associated with that frame, and expects a free frame buffer to be immediately available.

In resource availability mode the decoder may start to decode the next frame as soon as a free reference buffer is available. If a free frame buffer is not available immediately, the `PresentationTimes[ i ]` may be used to compute the time when such a buffer will become available.

The function `start_decode_at_removal_time` returns buffers to the BufferPool when they are no longer required for decode or display.

```

start_decode_at_removal_time( removal ) {
    for ( i = 0; i < BufferPoolMaxSize; i++ ) {
        if ( PlayerRefCount[ i ] > 0 ) {
            if ( PresentationTimes[ i ] < removal ) {
                PlayerRefCount[ i ] = 0
                if ( DecoderRefCount[ i ] == 0 )
                    free_buffer( i )
            }
            break
        }
    }
    return removal
}

```

The decoder needs to know the number of decoded frames in the BufferPool in order to determine the presentation delay for the first frame. A buffer is un-assigned if both DecoderRefCount[ i ] is equal to 0, and PlayerRefCount[ i ] is equal to 0.

The function frames\_in\_buffer\_pool returns the number of assigned frames in the BufferPool.

```

frames_in_buffer_pool( ) {
    framesInPool = 0
    for ( i = 0; i < BufferPoolMaxSize; i++ )
        if ( DecoderRefCount[ i ] != 0 || PlayerRefCount[ i ] != 0 )
            framesInPool++
    return framesInPool
}

```

The function get\_next\_frame switches to decoding the next frame in decoding order for the operating point. Variable DfgNum increments with each frame that needs decoding and corresponds to the DFG index. Variable ShownFrameNum increments with each shown frame read from the bitstream.

```

get_next_frame( frameNum )
{
    if ( read_frame_header( ) ) {
        frameNum++
        if ( !show_existing_frame ) {
            DfgNum++
        }
        if ( show_frame || show_existing_frame ) {
            ShownFrameNum++
        }
        return frameNum
    } else {
        return -1
    }
}

```



When the function `read_frame_header()` is invoked, the syntax elements and variables are set to the values at the conceptual point (in the decoding process specified in [section 7](#)) when the next uncompressed header has just been parsed. If there are no more frame headers in the bitstream, then a value of 0 is returned. Otherwise, a value of 1 is returned.

## E.5.2. Decoder model process

The decoder model process simulates the values of selected timing points as successive frames are decoded. This timing incorporates the time that the decoder has to wait for a free frame buffer, the time required to decode the frame and various basic checks to make sure that buffer slots are occupied when they are supposed to be. Non-conformance is signaled by a call to the function `bitstream_non_conformant`; the various error codes are tabulated below.

```

decode_process ( ) {
    initialize_buffer_pool( )
    time = 0
    frameNum = -1
    DfgNum = -1
    ShownFrameNum = -1
    cfbi = -1
    InitialPresentationDelay = 0
    while( (frameNum = get_next_frame( frameNum ) ) != - 1) {
        // Decode.
        if ( !show_existing_frame ) {
            if ( UsingResourceAvailabilityMode )
                Removal [ DfgNum ] = time_next_buffer_is_free( DfgNum, time )
            time = start_decode_at_removal_time( Removal[ DfgNum ] )
            if ( show_frame == 1 && time > PresentationTime[ ShownFrameNum ] )
                bitstream_non_conformant( DECODE_BUFFER_AVAILABLE_LATE )
            cfbi = get_free_buffer( )
            if ( cfbi == -1 )
                bitstream_non_conformant( DECODE_FRAME_BUF_UNAVAILABLE )
            time += time_to_decode_frame ( )
            update_ref_buffers ( cfbi, refresh_frame_flags )
            displayIdx = cfbi
            if ( InitialPresentationDelay == 0 &&
                ( frames_in_buffer_pool( ) >= initial_display_delay_minus_1[ operatingPoint ] + 1 ) )
                InitialPresentationDelay = time
        } else {
            displayIdx = VBI[ frame_to_show_map_idx ]
            if ( displayIdx == -1 )
                bitstream_non_conformant( DECODE_EXISTING_FRAME_BUF_EMPTY )
            if ( RefFrameType[ frame_to_show_map_idx ] == KEY_FRAME )
                update_ref_buffers( displayIdx, 0xFF )
        }
        // Display.
        if ( InitialPresentationDelay != 0 &&
            ( show_existing_frame == 1 || show_frame == 1 ) ) {
            // Presentation frame.
            if ( time > PresentationTime[ ShownFrameNum ] )
                bitstream_non_conformant( DISPLAY_FRAME_LATE )
            PresentationTimes[ displayIdx ] = PresentationTime[ ShownFrameNum ]
            PlayerRefCount[ displayIdx ] ++
        }
    }
}

```

where UsingResourceAvailabilityMode is a variable that is set to 1 when using resource availability mode, or 0 when using decoding schedule mode.

The various non-conformant error codes are:

Error Type	Description
DECODE_BUFFER_AVAILABLE_LATE	A free frame buffer only became available to the decoder after the time that the frame should have been displayed.
DECODE_FRAME_BUF_UNAVAILABLE	All the frame buffers were in use.
DECODE_EXISTING_FRAME_BUF_EMPTY	The index of the frame designated for display by a frame with <code>show_existing_frame = 1</code> was empty.
DISPLAY_FRAME_LATE	The frame was decoded too late for timely display, i.e. by the <code>PresentationTime[ i ]</code> time associated with the frame.

## E.6. Bitstream conformance

### E.6.1. General

A conformant coded bitstream shall satisfy the following set of constraints.

For the decoder model a DFG shall be available in the smoothing buffer at the scheduled removal time, i.e.  $\text{ScheduledRemoval}[ i ] \geq \text{LastBitArrival}[ i ]$ .

It is a requirement of the bitstream conformance that after each RAP, the  $\text{PresentationTime}[ j ]$ , where  $j$  corresponds to the frame decoding order is strictly increasing until the next RAP or the end of the coded video sequence, i.e.  $\text{PresentationTime}[ j + 1 ] > \text{PresentationTime}[ j ]$ .

When  $\text{buffer\_removal\_time}[ i ]$  is not present in the bitstream, a bitstream is conformant if the decoder model in resource availability mode can decode pictures successfully before they are scheduled for presentation.

If  $\text{buffer\_removal\_time}[ i ]$  is signaled, it shall have a value greater or equal than the equivalent value that would have been assigned if the decoder model was decoding frames in the resource availability mode.

Conformance requirements based on a decoder model are not applicable to a bitstream with `seq_level_idx` equal to 31.

In addition to these, a conformant bitstream shall satisfy the constraints specified in the following sections.

### E.6.2. Decoder buffer delay consistency across RAP (applies to decoding schedule mode)

For frame  $i$ , where  $i > 0$ ,  $\text{TimeDelta}[ i ]$  is defined as follows:

$$\text{TimeDelta}[ i ] = ( \text{ScheduledRemoval}[ i ] - \text{LastBitArrival}[ i - 1 ] ) * 90\,000$$

For the video sequence that includes one or more random access points, for each key frame, where the `decoder_buffer_delay` is signaled, the following expression should hold to provide smooth playback without the need to rebuffer.

$$\text{decoder\_buffer\_delay} \leq \text{ceil}(\text{TimeDelta}[i])$$

### E.6.3. Smoothing buffer overflow

Smoothing buffer overflow is defined as the state where the total number of bits in the smoothing buffer exceeds the size of the smoothing buffer `BufferSize`. The smoothing buffer shall never overflow.

### E.6.4. Smoothing buffer underflow

Smoothing buffer underflow is defined as the state where a complete DFG is not present in the smoothing buffer at the scheduled removal time, `ScheduledRemoval[i]`:

$$\text{ScheduledRemoval}[i] < \text{LastBitArrival}[i]$$

When the `low_delay_mode_flag[op]` is equal to 0, the smoothing buffer shall never underflow.

### E.6.5. Minimum decode time (applies to decoding schedule mode)

There must be enough time between a DFG being removed from the smoothing buffer, `Removal[i]`, and the scheduled removal of the next DFG, `ScheduledRemoval[i + 1]`:

$$\text{ScheduledRemoval}[i + 1] - \text{Removal}[i] \geq \text{Max}(\text{TimeToDecode}[i], 1 \div \text{MaxNumFrameHeadersPerSec}),$$

where `MaxNumFrameHeadersPerSec` is defined in the level constraints.

### E.6.6. Minimum presentation Interval

The difference between presentation times for consecutive shown frames, shall satisfy the following constraint:

$$\text{MinFrameTime} = \text{MaxTotalDecodedSampleRate} \div (\text{MaxNumFrameHeadersSec} * \text{MaxTotalDisplaySampleRate})$$

$$\text{PresentationInterval}[j] \geq \text{Max}(\text{LumaPels} \div \text{MaxTotalDisplaySampleRate}, \text{MinFrameTime})$$

Where `MaxTotalDecodedSampleRate`, `MaxNumFrameHeadersPerSec`, and `MaxTotalDisplaySampleRate` are defined in the level constraints.

### E.6.7. Decode deadline

It is a requirement of the bitstream conformance that each frame shall be fully decoded at, or before, the time that it is scheduled for presentation:

$$\text{Removal}[i] + \text{TimeToDecode}[i] \leq \text{PresentationTime}[i]$$

## E.6.8. Level imposed constraints

When operating in the decoding schedule mode, `decoder_buffer_delay` shall not be equal to 0 and shall not exceed  $90000 * (\text{BufferSize} \div \text{BitRate})$ .

**Note:** It is common to choose  $( (\text{encoder\_buffer\_delay} + \text{decoder\_buffer\_delay}) \div 90000 ) * \text{BitRate}$  equal to a constant within a coded video sequence, and for this constant to be equal to `BufferSize`, but these are not strict requirements for bitstream conformance.

## E.6.9. Decode Process constraints

It is a requirement of bitstream conformance that the decoder model process can be invoked with the bitstream data for any signaled operating point without triggering a call to the `bitstream_non_conformant` function.

# Bibliography

1. Recommendation ITU-R BT.601-7 (2011), Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios.
2. Recommendation ITU-R BT.709-6 (2015), Parameter values for the HDTV standards for production and international programme exchange.
3. SMPTE ST 170 (2004), Television – Composite Analog Video Signal – NTSC for Studio Applications.
4. SMPTE ST 240 (1999), For Television – 1125-Line High-Definition Production Systems – Signal Parameters.
5. Recommendation ITU-R BT.2020-2 (2015), Parameter values for ultra-high definition television systems for production and international programme exchange.
6. IEC 61966-2-1 (1999), Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB.
7. ISO/IEC 23091-4/ITU-T H.273, Coding-independent code points for video signal type identification.
8. Zhang, Cha, and Jin Li. “Compression of lumigraph with multiple reference frame (MRF) prediction and just-in-time rendering.” Data Compression Conference, 2000. Proceedings. DCC 2000. IEEE, 2000.
9. Birklbauer, Clemens, Simon Opelt, and Oliver Bimber. “Rendering gigaray light fields.” Computer Graphics Forum. Vol. 32. No. 2pt4. Blackwell Publishing Ltd, 2013.