



# AV1 Bitstream & Decoding Process Specification

*Last modified: 2018-03-28 09:28 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

## Draft Document

This is a **draft document** that will change substantially before release. Do not rely on its current contents. To comment on the document, use 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 format 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. [Arithmetic operators](#)
  - 4.2. [Logical operators](#)
  - 4.3. [Relational operators](#)
  - 4.4. [Bitwise operators](#)
  - 4.5. [Assignment](#)
  - 4.6. [Mathematical functions](#)
  - 4.7. [Method of describing bitstream syntax](#)
  - 4.8. [Functions](#)
  - 4.9. [Descriptors](#)
    - 4.9.1. [f\(n\)](#)
    - 4.9.2. [uvlc\(\)](#)
    - 4.9.3. [le\(n\)](#)
    - 4.9.4. [leb128\(\)](#)
    - 4.9.5. [su\(n\)](#)
    - 4.9.6. [ns\(n\)](#)
    - 4.9.7. [L\(n\)](#)
    - 4.9.8. [S\(\)](#)
    - 4.9.9. [NS\(n\)](#)
5. [Syntax Structures](#)
  - 5.1. [OBU Syntax](#)
    - 5.1.1. [OBU Header Syntax](#)
    - 5.1.2. [OBU Extension Header Syntax](#)
    - 5.1.3. [Trailing Bits Syntax](#)
    - 5.1.4. [Byte alignment Syntax](#)
  - 5.2. [Reserved OBU Syntax](#)
  - 5.3. [Sequence Header OBU Syntax](#)
    - 5.3.1. [Color Config Syntax](#)
    - 5.3.2. [Timing Info Syntax](#)
  - 5.4. [Temporal Delimiter OBU Syntax](#)
  - 5.5. [Padding OBU Syntax](#)
  - 5.6. [Metadata OBU Syntax](#)
    - 5.6.1. [Metadata Private Data Syntax](#)
    - 5.6.2. [Metadata High Dynamic Range Content Light Level Syntax](#)
    - 5.6.3. [Metadata High Dynamic Range Mastering Display Color Volume Syntax](#)
    - 5.6.4. [Metadata Scalability Syntax](#)

- 5.6.5. Scalability structure syntax
- 5.7. Frame Header OBU Syntax
  - 5.7.1. Uncompressed Header Syntax
  - 5.7.2. Get Relative Distance Function
  - 5.7.3. Reference Frame Marking Function
  - 5.7.4. Frame Size Syntax
  - 5.7.5. Render Size Syntax
  - 5.7.6. Frame Size with Refs Syntax
  - 5.7.7. Superres Params Syntax
  - 5.7.8. Compute Image Size Function
  - 5.7.9. Interpolation Filter Syntax
  - 5.7.10. Loop Filter Params Syntax
  - 5.7.11. Quantization Params Syntax
  - 5.7.12. Delta Quantizer Syntax
  - 5.7.13. Segmentation Params Syntax
  - 5.7.14. Tile Info Syntax
  - 5.7.15. Tile Size Calculation Function
  - 5.7.16. Quantizer Index Delta Parameters Syntax
  - 5.7.17. Loop Filter Delta Parameters Syntax
  - 5.7.18. CDEF Params Syntax
  - 5.7.19. Loop Restoration Params Syntax
  - 5.7.20. TX Mode Syntax
  - 5.7.21. Skip Mode Params Syntax
  - 5.7.22. Frame Reference Mode Syntax
  - 5.7.23. Global Motion Params Syntax
  - 5.7.24. Global Param Syntax
  - 5.7.25. Decode Signed Subexp With Ref Syntax
  - 5.7.26. Decode Unsigned Subexp With Ref Syntax
  - 5.7.27. Decode Subexp Syntax
  - 5.7.28. Inverse Recenter Function
  - 5.7.29. Inv Recenter Noneg Function
  - 5.7.30. Film Grain Params Syntax
- 5.8. Frame OBU Syntax
- 5.9. Tile Group OBU Syntax
  - 5.9.1. Decode Tile Syntax
  - 5.9.2. Clear Block Decoded Flags Function
  - 5.9.3. Decode Partition Syntax
  - 5.9.4. Decode Block Syntax
  - 5.9.5. Mode Info Syntax
  - 5.9.6. Intra Frame Mode Info Syntax
  - 5.9.7. Intra Segment ID Syntax
  - 5.9.8. Read Segment ID Syntax
  - 5.9.9. Skip Mode Syntax
  - 5.9.10. Skip Syntax
  - 5.9.11. Quantizer Index Delta Syntax

5.9.12. Loop Filter Delta Syntax  
5.9.13. Segmentation Feature Active Function  
5.9.14. TX Size Syntax  
5.9.15. Block TX Size Syntax  
5.9.16. Var TX Size Syntax  
5.9.17. Inter Frame Mode Info Syntax  
5.9.18. Inter Segment ID Syntax  
5.9.19. Is Inter Syntax  
5.9.20. Get Segment ID Function  
5.9.21. Intra Block Mode Info Syntax  
5.9.22. Inter Block Mode Info Syntax  
5.9.23. Filter Intra Mode Info Syntax  
5.9.24. Ref Frames Syntax  
5.9.25. Assign MV Syntax  
5.9.26. Read Motion Mode Syntax  
5.9.27. Read Inter Intra Syntax  
5.9.28. Read Compound Type Syntax  
5.9.29. Get Mode Function  
5.9.30. MV Syntax  
5.9.31. MV Component Syntax  
5.9.32. Residual Syntax  
5.9.33. Transform Block Syntax  
5.9.34. Transform Tree Syntax  
5.9.35. Get TX Size Function  
5.9.36. Get Plane Residual Size Function  
5.9.37. Coefficients Syntax  
5.9.38. Compute Transform Type Function  
5.9.39. Get Scan Function  
5.9.40. Intra Angle Info Luma Syntax  
5.9.41. Intra Angle Info Chroma Syntax  
5.9.42. Is Directional Mode Function  
5.9.43. Read CFL Alphas Syntax  
5.9.44. Palette Mode Info Syntax  
5.9.45. Transform Type Syntax  
5.9.46. Get Transform Set Function  
5.9.47. Palette Tokens Syntax  
5.9.48. Palette Color Context Function  
5.9.49. Is Inside Function  
5.9.50. Is Inside Filter Region Function  
5.9.51. Check References Functions  
5.9.52. Clamp MV Row Function  
5.9.53. Clamp MV Col Function  
5.9.54. Clear CDEF Function  
5.9.55. Read CDEF Syntax  
5.9.56. Decode Loop Restoration Syntax

## 5.9.57. Decode Loop Restoration Unit Syntax

## 6. Syntax Structures Semantics

## 6.1. OBU Semantics

## 6.1.1. OBU Header Semantics

## 6.1.2. OBU Extension Header Semantics

## 6.1.3. Trailing Bits Semantics

## 6.1.4. Byte Alignment Semantics

## 6.2. Reserved OBU Semantics

## 6.3. Sequence Header OBU Semantics

## 6.3.1. Color Config Semantics

## 6.3.2. Timing Info Semantics

## 6.4. Temporal Delimiter OBU Semantics

## 6.5. Padding OBU Semantics

## 6.6. Metadata OBU Semantics

## 6.6.1. Metadata Private Data Semantics

## 6.6.2. Metadata High Dynamic Range Content Light Level Semantics

## 6.6.3. Metadata High Dynamic Range Mastering Display Color Volume Semantics

## 6.6.4. Metadata Scalability Semantics

## 6.6.5. Scalability Structure Semantics

## 6.7. Frame Header OBU Semantics

## 6.7.1. Uncompressed Header Semantics

## 6.7.2. Reference Frame Marking Semantics

## 6.7.3. Frame Size Semantics

## 6.7.4. Render Size Semantics

## 6.7.5. Frame Size with Refs Semantics

## 6.7.6. Superres Params Semantics

## 6.7.7. Compute Image Size Semantics

## 6.7.8. Interpolation Filter Semantics

## 6.7.9. Loop Filter Semantics

## 6.7.10. Quantization Params Semantics

## 6.7.11. Delta Quantizer Semantics

## 6.7.12. Segmentation Params Semantics

## 6.7.13. Tile Info Semantics

## 6.7.14. Quantizer Index Delta Parameters Semantics

## 6.7.15. Loop Filter Delta Parameters Semantics

## 6.7.16. Global Motion Params Semantics

## 6.7.17. Global Param Semantics

## 6.7.18. Decode Subexp Semantics

## 6.7.19. Decode Uniform Semantics

## 6.7.20. Film Grain Params Semantics

## 6.7.21. TX Mode Semantics

## 6.7.22. Skip Mode Params Semantics

## 6.7.23. Frame Reference Mode Semantics

## 6.8. Frame OBU Semantics

## 6.9. Tile Group OBU Semantics

- 6.9.1. Decode Tile Semantics
- 6.9.2. Clear Block Decoded Flags Semantics
- 6.9.3. Decode Partition Semantics
- 6.9.4. Decode Block Semantics
- 6.9.5. Intra Frame Mode Info Semantics
- 6.9.6. Intra Segment ID Semantics
- 6.9.7. Read Segment ID Semantics
- 6.9.8. Inter Segment ID Semantics
- 6.9.9. Skip Mode Semantics
- 6.9.10. Skip Semantics
- 6.9.11. Quantizer Index Delta Semantics
- 6.9.12. Loop Filter Delta Semantics
- 6.9.13. CDEF Params Semantics
- 6.9.14. Loop Restoration Params Semantics
- 6.9.15. TX Size Semantics
- 6.9.16. Block TX Size Semantics
- 6.9.17. Var TX Size Semantics
- 6.9.18. Transform Type Semantics
- 6.9.19. Is Inter Semantics
- 6.9.20. Intra Block Mode Info Semantics
- 6.9.21. Inter Block Mode Info Semantics
- 6.9.22. Filter Intra Mode Info Semantics
- 6.9.23. Ref Frames Semantics
- 6.9.24. Read Motion Mode Semantics
- 6.9.25. Read Inter Intra Semantics
- 6.9.26. Read Compound Type Semantics
- 6.9.27. MV Semantics
- 6.9.28. MV Component Semantics
- 6.9.29. Residual Semantics
- 6.9.30. Transform Block Semantics
- 6.9.31. Coefficients Semantics
- 6.9.32. Intra Angle Info Semantics
- 6.9.33. Read CFL Alphas Semantics
- 6.9.34. Palette Mode Info Semantics
- 6.9.35. Palette Tokens Semantics
- 6.9.36. Palette Color Context Semantics
- 6.9.37. Read CDEF Semantics
- 6.9.38. Decode Loop Restoration Unit Semantics

## 7. Decoding Process

- 7.1. General Decoding Process
- 7.2. Large Scale Tile Decoding Process
- 7.3. Decode Frame Process
- 7.4. Ordering of OBUs
- 7.5. Random Access Decoding
- 7.6. CDF Update Process

- 7.7. Set Frame Refs Process
- 7.8. Motion Field Estimation Process
  - 7.8.1. Projection Process
  - 7.8.2. Get MV Projection Process
  - 7.8.3. Get Block Position Process
- 7.9. Motion Vector Prediction Processes
  - 7.9.1. Find MV Stack Process
    - 7.9.1.1. Setup Zero MV Process
    - 7.9.1.2. Scan Row Process
    - 7.9.1.3. Scan Col Process
    - 7.9.1.4. Scan Point Process
    - 7.9.1.5. Temporal Scan Process
    - 7.9.1.6. Temporal Sample Process
    - 7.9.1.7. Add Reference Motion Vector Process
    - 7.9.1.8. Search Stack Process
    - 7.9.1.9. Compound Search Stack Process
    - 7.9.1.10. Lower Precision Process
    - 7.9.1.11. Sorting Process
    - 7.9.1.12. Extra Search Process
    - 7.9.1.13. Add Extra Mv Candidate Process
    - 7.9.1.14. Context and Clamping Process
  - 7.9.2. Has Overlappable Candidates Process
  - 7.9.3. Find Warp Samples Process
    - 7.9.3.1. Add Sample Process
- 7.10. Prediction Processes
  - 7.10.1. Intra Prediction Process
    - 7.10.1.1. Basic Intra Prediction Process
    - 7.10.1.2. Recursive Intra Prediction Process
    - 7.10.1.3. Directional Intra Prediction Process
    - 7.10.1.4. DC Intra Prediction Process
    - 7.10.1.5. Smooth Intra Prediction Process
    - 7.10.1.6. Filter Corner Process
    - 7.10.1.7. Intra Filter Type Process
    - 7.10.1.8. Intra Edge Filter Strength Selection Process
    - 7.10.1.9. Intra Edge Upsample Selection Process
    - 7.10.1.10. Intra Edge Upsample Process
    - 7.10.1.11. Intra Edge Filter Process
  - 7.10.2. Inter Prediction Process
    - 7.10.2.1. Rounding Variables Derivation Process
    - 7.10.2.2. Motion Vector Scaling Process
    - 7.10.2.3. Block Inter Prediction Process
    - 7.10.2.4. Block Warp Process
    - 7.10.2.5. Setup Shear Process
    - 7.10.2.6. Resolve Divisor Process
    - 7.10.2.7. Warp Estimation Process

- 7.10.2.8. Overlapped Motion Compensation Process
      - 7.10.2.9. Overlap Blending Process
      - 7.10.2.10. Wedge Mask Process
      - 7.10.2.11. Segment Mask Process
      - 7.10.2.12. Inter Intra Mask Process
      - 7.10.2.13. Mask Blend Process
      - 7.10.2.14. Distance Weights Process
    - 7.10.3. Palette Prediction Process
    - 7.10.4. Predict Chroma From Luma Process
  - 7.11. Reconstruction and Dequantization
    - 7.11.1. Dequantization Functions
    - 7.11.2. Reconstruct Process
  - 7.12. Inverse Transform Process
    - 7.12.1. 1D Transforms
      - 7.12.1.1. Butterfly Functions
      - 7.12.1.2. Inverse DCT Array Permutation Process
      - 7.12.1.3. Inverse DCT Process
      - 7.12.1.4. Inverse ADST Input Array Permutation Process
      - 7.12.1.5. Inverse ADST Output Array Permutation Process
      - 7.12.1.6. Inverse ADST4 process
      - 7.12.1.7. Inverse ADST8 Process
      - 7.12.1.8. Inverse ADST16 Process
      - 7.12.1.9. Inverse ADST Process
      - 7.12.1.10. Inverse Walsh-Hadamard Transform Process
      - 7.12.1.11. Inverse Identity Transform 4 Process
      - 7.12.1.12. Inverse Identity Transform 8 Process
      - 7.12.1.13. Inverse Identity Transform 16 Process
      - 7.12.1.14. Inverse Identity Transform 32 Process
      - 7.12.1.15. Inverse Identity Transform 64 Process
      - 7.12.1.16. Inverse Identity Transform Process
    - 7.12.2. 2D Inverse Transform
  - 7.13. Loop Filter Process
    - 7.13.1. Edge Loop Filter Process
    - 7.13.2. Filter Size Process
    - 7.13.3. Adaptive Filter Strength Process
    - 7.13.4. Adaptive Filter Strength Selection Process
    - 7.13.5. Sample Filtering Process
      - 7.13.5.1. Filter Mask Process
      - 7.13.5.2. Narrow Filter Process
      - 7.13.5.3. Wide Filter Process
  - 7.14. CDEF Process
    - 7.14.1. CDEF Block Process
    - 7.14.2. CDEF Direction Process
    - 7.14.3. CDEF Filter Process
  - 7.15. Upscaling Process



- 7.16. Loop Restoration Process
  - 7.16.1. Loop Restore Block Process
  - 7.16.2. Self Guided Filter Process
  - 7.16.3. Box Filter Process
  - 7.16.4. Wiener Filter Process
  - 7.16.5. Wiener Coefficient Process
  - 7.16.6. Get Source Sample Process
- 7.17. Output Process
  - 7.17.1. Film Grain Synthesis Process
    - 7.17.1.1. Random Number Process
    - 7.17.1.2. Generate Grain Process
    - 7.17.1.3. Scaling Lookup Initialization Process
    - 7.17.1.4. Add Noise Synthesis Process
- 7.18. Motion Field Motion Vector Storage Process
- 7.19. Reference Frame Update Process
- 7.20. Reference Frame Loading Process
- 8. Parsing Process
  - 8.1. Parsing Process for  $f(n)$
  - 8.2. Parsing Process for Symbol Decoder
    - 8.2.1. Initialization Process for Symbol Decoder
    - 8.2.2. Boolean Decoding Process
    - 8.2.3. Exit Process for Symbol Decoder
    - 8.2.4. Parsing Process for read\_literal
    - 8.2.5. Symbol Decoding Process
  - 8.3. Parsing process for CDF encoded syntax elements
    - 8.3.1. CDF Selection Process
- 9. Additional Tables
  - 9.1. Scan Tables
  - 9.2. Conversion Tables
  - 9.3. Default CDF Tables
  - 9.4. Quantizer Matrix Tables
    - 9.4.1. Derivation Process (Informative)
    - 9.4.2. Tables
- 10. Annex A: Profiles and Levels
  - 10.1. Profiles
  - 10.2. Levels
- 11. Annex B: Bitstream Format
  - 11.1. Overview
  - 11.2. Bitstream Syntax
  - 11.3. Bitstream semantics
- 12. Annex C: Included Experiments
  - 12.1. Overview
  - 12.2. Included Experiments
  - 12.3. Excluded Experiments
- 13. Bibliography

# 1. Scope

This document specifies the Alliance for Open Media AV1 bitstream format 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 pixels consisting of one Luma and two Chroma matrices.

**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.

**Frame buffer**

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

**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-only frame**

A type of intra frame that does not reset the decoding process.

**Note:** A key frame is different to an intra-only frame even though both only use intra prediction. The difference is that a key frame fully resets the decoding process. For example, for a sequence of frames A, B, C, frame C can use frame A as a reference if B was an intra-only frame - but not if B was a keyframe.

**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**

A frame where the decoding process is reset. Key frames, and following frames, are always decodable without access to preceding frames. A key frame only uses intra prediction.

**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 frame**

A previously decoded frame used during inter prediction.

**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 buffer 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 pixels or 64x64 pixels. A superblock may contain 1 or 2 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 frame buffers 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. The decoded pictures after a switch may be slightly incorrect, but this approach allows a 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 be applied.

**Transform block**

A square 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	14	Sentinel value to mark partition choices that are illegal
MAX_SB_SIZE	128	Maximum size of a superblock in pixels
MI_SIZE	4	Smallest size of a mode info block in pixels
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 pixels 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	9	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	2708	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	6	Number of 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
CDEF_VERY_LARGE	30000	Arbitrary large value used to signal unavailable pixels
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

Symbol name	Value	Description
MFMV_STACK_SIZE	3	Stack size for motion field motion vectors
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	8	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

Symbol name	Value	Description
WARP_PARAM_REDUCE_BITS	6	Rounding bitwidth for the parameters to the shear process
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	46	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	Pixel 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	22	Number of contexts for <code>eob_extra</code>



Symbol name	Value	Description
DC_SIGN_CONTEXTS	3	Number of contexts for <code>dc_sign</code>
LEVEL_CONTEXTS	21	Number of contexts for <code>coeff_br</code>
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	$(1 \ll 12) - 1$	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 <code>coeff( )</code> syntax structure
PRIMARY_REF_NONE	7	Value of <code>primary_ref_frame</code> indicating that there is no primary reference frame

## 4. Conventions

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 normal 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 normal 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`.

### 4.1. 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.

### 4.2. 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.

### 4.3. Relational operators

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to

!=	Not equal to
----	--------------

## 4.4. 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> .

## 4.5. 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>

## 4.6. 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 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
}
```

## 4.7. 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>	
<b><code>syntax_element</code></b>	<code>f(1)</code>
<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</code>	

true. */	
while ( condition )	
statement	
/* A “do .. while” structure executes the statement once,	
and then tests the condition. It repeatedly evaluates the	
statement while the condition remains true. */	
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	

## 4.8. 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 8.2.1](#).

**exit\_symbol( readPadding )**: Exit the arithmetic decode process as described in [section 8.2.3](#).

## 4.9. Descriptors

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.

### 4.9.1. 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.

### 4.9.2. 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	
}	

### 4.9.3. 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.

## 4.9.4. 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:

	Type
leb128() {	
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	
}	

**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.

## 4.9.5. 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:

	Type
su(n) {	
value	f(n)
signMask = 1 << (n - 1)	



if (value & signMask)	
value = value - 2 * signMask	
return value	
}	

## 4.9.6. ns(n)

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

This descriptor is similar to  $f(\text{ceil}(\log_2(n)))$ , 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.

## 4.9.7. 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 8.2.4](#) for a specification of this process).

## 4.9.8. 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).

4.9.9. 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

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

### 5.1. 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 &&	
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_TD )	
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_PADDING )	
padding_obu( )	
else	
reserved_obu( )	
currentPosition = get_position( )	
payloadBits = currentPosition - startPosition	
if ( obu_size > 0 )	
trailing_bits( obu_size * 8 - payloadBits )	

}	
---	--

### 5.1.1. 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()	
}	

### 5.1.2. OBU Extension Header Syntax

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

### 5.1.3. Trailing Bits Syntax

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

### 5.1.4. Byte alignment Syntax

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

## 5.2. 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.

## 5.3. Sequence Header OBU Syntax

sequence_header_obu( ) {	Type
seq_profile	f(2)
operating_points_minus1_cnt	f(5)
for ( i = 0; i <= operating_points_minus1_cnt; i++ ) {	
operating_point_idc[ i ]	f(12)
level[ i ]	f(4)
decoder_rate_model_param_present_flag[ i ]	f(1)
if ( decoder_rate_model_param_present_flag[ i ] ) {	
decode_to_display_rate_ratio[ i ]	f(12)
initial_display_delay[ i ]	f(24)
extra_frame_buffers[ 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)
frame_id_numbers_present_flag	f(1)
if ( frame_id_numbers_present_flag ) {	
delta_frame_id_length_minus2	f(4)
additional_frame_id_length_minus1	f(3)
}	
use_128x128_superblock	f(1)
enable_filter_intra	f(1)
enable_intra_edge_filter	f(1)
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_minus1	f(3)
OrderHintBits = order_hint_bits_minus1 + 1	
} else {	
OrderHintBits = 0	
}	
enable_superres	f(1)
enable_cdef	f(1)
enable_restoration	f(1)
color_config( )	
timing_info_present_flag	f(1)
if ( timing_info_present_flag ) {	
timing_info( )	
}	
film_grain_params_present	f(1)
}	

### 5.3.1. 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 {	
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)

<pre> } else {     color_primaries = CP_UNSPECIFIED     transfer_characteristics = TC_UNSPECIFIED     matrix_coefficients = MC_UNSPECIFIED } if ( mono_chrome ) {     color_range = 1     subsampling_x = 1     subsampling_y = 1     chroma_sample_position = CSP_UNKNOWN } else if ( color_primaries == CP_BT_709 &amp;&amp;             transfer_characteristics == TC_SRGB &amp;&amp;             matrix_coefficients == MC_IDENTITY ) {     subsampling_x = 0     subsampling_y = 0 } else {     color_range     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             if ( subsampling_x )                 subsampling_y             else                 subsampling_y = 0         } else {             subsampling_x = 1             subsampling_y = 0         }     }     if (subsampling_x &amp;&amp; subsampling_y) {         chroma_sample_position     } } separate_uv_delta_q } </pre>	<pre> f(1) f(1) f(1) f(2) f(1) </pre>
--	---------------------------------------

### 5.3.2. Timing Info Syntax

<pre> timing_info( ) {     num_units_in_tick     time_scale     equal_picture_interval     if (equal_picture_interval) </pre>	<pre> Type f(32) f(32) f(1) </pre>
---	------------------------------------

num_ticks_per_picture_minus1	uvlc()
}	

## 5.4. Temporal Delimiter OBU Syntax

temporal_delimiter_obu( ) {	Type
SeenFrameHeader = 0	
}	

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

## 5.5. 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 neither that length nor the number of trailing bytes. They can ignore the entire OBU. Ignoring the OBU can be done based on obu\_size.

## 5.6. Metadata OBU Syntax

metadata_obu( ) {	Type
metadata_type	f(16)
if ( metadata_type == METADATA_TYPE_PRIVATE_DATA )	
metadata_private_data( )	
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( )	
}	

### 5.6.1. Metadata Private Data Syntax

metadata_private_data( ) {	Type
}	



**Note:** The exact syntax of the private metadata OBU is not defined. 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 the `metadata_type`. The last byte of the valid content of the private 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.

## 5.6.2. Metadata High Dynamic Range Content Light Level Syntax

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

## 5.6.3. 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)
}	

## 5.6.4. Metadata Scalability Syntax

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

## 5.6.5. Scalability structure syntax

scalability_structure( ) {	Type
spatial_layers_cnt	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 ; 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; 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)
}	
}	
}	
}	

## 5.7. 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( )	
SeenFrameHeader = 0	
} else {	
TileNum = 0	
MaxTileSize = 0	
SeenFrameHeader = 1	
}	
}	
}	

### 5.7.1. Uncompressed Header Syntax

uncompressed_header( ) {	Type
idLen = ( additional_frame_id_length_minus1 +	
delta_frame_id_length_minus2 + 3 )	
show_existing_frame	f(1)
if ( show_existing_frame == 1 ) {	
frame_to_show_map_idx	f(3)
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 = (1 << NUM_REF_FRAMES) - 1	
}	
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 ) {	
showable_frame = 0	
} else {	
showable_frame	f(1)
}	
if ( frame_type == KEY_FRAME && show_frame ) {	
for ( i = 0; i < NUM_REF_FRAMES; i++ )	
RefValid[ i ] = 0	
}	
if ( frame_type == SWITCH_FRAME )	
error_resilient_mode = 1	
else	
error_resilient_mode	f(1)
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 ( frame_id_numbers_present_flag ) {	
PrevFrameID = current_frame_id	
current_frame_id	f(idLen)
mark_ref_frames( )	
}	
if ( frame_type == SWITCH_FRAME )	
frame_size_override_flag = 1	
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)
}	
allow_high_precision_mv = 0	
use_ref_frame_mvs = 0	
allow_intrabc = 0	
if ( frame_type == KEY_FRAME ) {	
if ( show_frame )	
refresh_frame_flags = (1 << NUM_REF_FRAMES) - 1	
else	
refresh_frame_flags	f(8)
frame_size( )	
render_size( )	
if ( allow_screen_content_tools && UpscaledWidth == FrameWidth ) {	
allow_intrabc	f(1)
}	
} else {	
if ( error_resilient_mode && enable_order_hint ) {	
for ( i = 0; i < NUM_REF_FRAMES; i++ ) {	
ref_order_hint[ i ]	f(OrderHintBits)
}	
}	
if ( frame_type == INTRA_ONLY_FRAME ) {	
refresh_frame_flags	f(8)
frame_size( )	
render_size( )	
if ( allow_screen_content_tools && UpscaledWidth == FrameWidth ) {	
allow_intrabc	f(1)
}	
} else {	
if ( frame_type == SWITCH_FRAME ) {	
refresh_frame_flags = (1 << NUM_REF_FRAMES) - 1	
} else {	
refresh_frame_flags	f(8)
}	
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_minus2 + 2	
delta_frame_id_minus1	f(n)
DeltaFrameId = delta_frame_id_minus1 + 1	
RefFrameId[ 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 ( 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( )	
}	
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( )	
if ( show_frame    showable_frame) {	
film_grain_params( )	
}	
}	

## 5.7.2. 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	
}	

## 5.7.3. Reference Frame Marking Function

mark_ref_frames( ) {	Type
diffLen = delta_frame_id_length_minus2 + 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	
}	
}	
}	

## 5.7.4. 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	
}	
compute_image_size( )	
superres_params( )	
}	

### 5.7.5. 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	
}	
}	

### 5.7.6. 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 = RefFrameWidth[ ref_frame_idx[ i ] ]	
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 {	
compute_image_size( )	
superres_params( )	
}	
}	

### 5.7.7. 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	
}	

### 5.7.8. Compute Image Size Function

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

### 5.7.9. 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)
}	
}	

### 5.7.10. 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)
}	
}	
}	
}	

### 5.7.11. 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	

DeltaQUdc = 0	
DeltaQVAc = 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)
}	
}	

## 5.7.12. 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	
}	

## 5.7.13. 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 }
```

### 5.7.14. 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)	
maxTileHeightSb = sbRows >> minLog2TileRows	
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	

<pre> } else {     widestTileSb = 0     startSb = 0     for ( i = 0; startSb &lt; sbCols &amp;&amp; i &lt; MAX_TILE_COLS; i++ ) {         MiColStarts[ i ] = startSb &lt;&lt; sbShift         maxWidth = Min(sbCols - startSb, maxTileWidthSb)         width_in_sbs_minus_1         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 &gt; 0 )         maxTileAreaSb = (sbRows * sbCols) &gt;&gt; (minLog2Tiles + 1)     else         maxTileAreaSb = sbRows * sbCols     maxTileHeightSb = Max( maxTileAreaSb / widestTileSb, 1 )      startSb = 0     for ( i = 0; startSb &lt; sbRows &amp;&amp; i &lt; MAX_TILE_ROWS; i++ ) {         MiRowStarts[ i ] = startSb &lt;&lt; sbShift         maxHeight = Min(sbRows - startSb, maxTileHeightSb)         height_in_sbs_minus_1         sizeSb = height_in_sbs_minus_1 + 1         startSb += sizeSb     }     MiRowStarts[ i ] = MiRows     TileRows = i     TileRowsLog2 = tile_log2(1, TileRows) } if ( TileColsLog2 &gt; 0    TileRowsLog2 &gt; 0 ) {     tile_size_bytes_minus_1     TileSizeBytes = tile_size_bytes_minus_1 + 1 } } </pre>	<pre> ns(maxWidth) ns(maxHeight) f(2) </pre>
--	--

## 5.7.15. Tile Size Calculation Function

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

<pre> tile_log2( blkSize, target ) {     for (k = 0; (blkSize &lt;&lt; k) &lt; target; k++) {     }     return k } </pre>	Type
---	------

## 5.7.16. 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)
}	
}	

## 5.7.17. 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 ) {	
delta_lf_present	f(1)
if ( delta_lf_present ) {	
delta_lf_res	f(2)
delta_lf_multi	f(1)
}	
}	
}	

## 5.7.18. 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	
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	
}	
}	
}	

## 5.7.19. 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	
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:

<pre>Remap_Lr_Type[4] = {     RESTORE_NONE, RESTORE_SWITCHABLE, RESTORE_WIENER, RESTORE_SGRPROJ }</pre>
---

## 5.7.20. 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	
}	
}	
}	

## 5.7.21. 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	
}	
}	

## 5.7.22. Frame Reference Mode Syntax

frame_reference_mode( ) {	Type
compoundReferenceAllowed = 0	
if ( !FrameIsIntra ) {	
bufIdx = ref_frame_idx[0]	
refOffset0 = RefOrderHint[bufIdx]	
for (ref = LAST_FRAME + 1; ref <= ALTREF_FRAME; ref++) {	
bufIdx = ref_frame_idx[ref - LAST_FRAME]	
refOffset = RefOrderHint[bufIdx]	
if (refOffset != refOffset0) {	
compoundReferenceAllowed = 1	
break	
}	
}	
}	

<pre>     }     if ( compoundReferenceAllowed ) {         reference_select     } else {         reference_select = 0     } } </pre>	
	f(1)

### 5.7.23. Global Motion Params Syntax

<pre> global_motion_params( ) {     if ( FrameIsIntra )         return     for (ref = LAST_FRAME; ref &lt;= ALTREF_FRAME; ref++) {         is_global         if (is_global) {             is_rot_zoom             if (is_rot_zoom) {                 type = ROTZOOM             } else {                 is_translation                 type = is_translation ? TRANSLATION : AFFINE             }         } else {             type = IDENTITY         }         GmType[ref] = type          if (type &gt;= 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 &gt;= TRANSLATION) {             read_global_param(type,ref,0)             read_global_param(type,ref,1)         }     } } </pre>	Type
	f(1)
	f(1)
	f(1)

### 5.7.24. Global Param Syntax

<pre> read_global_param( type, ref, idx ) {     absBits = GM_ABS_ALPHA_BITS </pre>	Type
--	------

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 = (gm_params[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 Zero MV process.

### 5.7.25. 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	
}	

### 5.7.26. 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)	
}	
}	

### 5.7.27. 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	
}	
}	
}	

## 5.7.28. 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)	
}	

## 5.7.29. Inv Recenter Nonneg Function

inv_recenter_nonneg( v, m ) {	Type
if ( v > 2 * m )	
return v	
if ( v & 1 )	
return m - ((v + 1) >> 1)	
return m + (v >> 1)	
}	

## 5.7.30. Film Grain Params Syntax

film_grain_params( ) {	Type
if ( !film_grain_params_present ) {	
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)
}	

## 5.8. 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 )	
}	

## 5.9. 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( !lastTile )	
}	
if (tg_end == NumTiles - 1) {	
if ( !disable_frame_end_update_cdf ) {	
update_cdf( )	
}	
decode_frame( )	
SeenFrameHeader = 0	
}	
}	

### 5.9.1. 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( c < ( MiColEnd - 1 ) )	
decode_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 }
```

## 5.9.2. Clear Block Decoded Flags Function

clear_block_decoded_flags( notLastColumn ) {	Type
sbSize = use_128x128_superblock ? 128 : 64	
for ( plane = 0; plane < NumPlanes; plane++ ) {	
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
for ( y = -1; y <= ( sbSize >> ( MI_SIZE_LOG2 + subY ) ); y++ )	
for ( x = -1; x <= ( sbSize >> ( MI_SIZE_LOG2 + subX ) ); x++ ) {	
BlockDecoded[ plane ][ y ][ x ] = ( y < 0    x < 0 )	
}	
BlockDecoded[ plane ][ -1 ][ sbSize >> ( MI_SIZE_LOG2 + subX ) ] = notLastColumn	
BlockDecoded[ plane ][ sbSize >> ( MI_SIZE_LOG2 + subY ) ][ -1 ] = 0	
}	
}	

## 5.9.3. Decode Partition Syntax

decode_partition( r, c, bSize ) {	Type
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 ) {	
partition	S()
} else if ( hasCols ) {	
split_or_horz	S()
partition = split_or_horz ? PARTITION_SPLIT : PARTITION_HORZ	
} else if ( hasRows ) {	
split_or_vert	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 )	
} else if ( partition == PARTITION_HORZ_4 ) {	

decode_block( r + quarterBlock4x4 * 0, c, subSize )	
decode_block( r + quarterBlock4x4 * 1, c, subSize )	
decode_block( r + quarterBlock4x4 * 2, c, subSize )	
decode_block( r + quarterBlock4x4 * 3, c, subSize )	
} else {	
decode_block( r, c + quarterBlock4x4 * 0, subSize )	
decode_block( r, c + quarterBlock4x4 * 1, subSize )	
decode_block( r, c + quarterBlock4x4 * 2, subSize )	
decode_block( r, c + quarterBlock4x4 * 3, subSize )	
}	
}	

## 5.9.4. 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 )	
for ( y = 0; y < bh4; y++ ) {	
for ( x = 0; x < bw4; x++ ) {	
YModes [ r + y ][ c + x ] = YMode	
if ( !is_inter && 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 < 2; refList++ ) {	
Mvs[ r + y ][ c + x ][ refList ] = Mv[ refList ]	
}	
}	
}	
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	
TileNums[ r + y ][ c + x ] = TileNum	
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 ) {
    mask = use_128x128_superblock ? 255 : 127
    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 & mask ] = 0
            LeftDcContext[ plane ][ i & mask ] = 0
        }
    }
}

```

### 5.9.5. Mode Info Syntax

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

### 5.9.6. 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	
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( )	
}	

### 5.9.7. Intra Segment ID Syntax

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

### 5.9.8. 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)
    }
}

```

## 5.9.9. Skip Mode Syntax

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

## 5.9.10. Skip Syntax

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

## 5.9.11. 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))	
}	
}	
}	

## 5.9.12. 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( 0, MAX_LOOP_FILTER, DeltaLF[ i ] +	
(reducedDeltaLfLevel << delta_lf_res) )	
}	
}	
}	
}	

### 5.9.13. 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 )	
}	

### 5.9.14. TX Size Syntax

read_tx_size( allowSelect ) {	Type
largestTxSize = Tx_Mode_To_Biggest_Tx_Size[ TxMode ]	
maxTxSize = Max_Tx_Size[ MiSize ]	
maxRectTxSize = Max_Tx_Size_Rect[ MiSize ]	
maxTxDepth = Max_Tx_Depth[ MiSize ]	
if ( MiSize > BLOCK_4X4 ) {	
if ( allowSelect && TxMode == TX_MODE_SELECT ) {	

tx_depth	S()
TxSize = maxRectTxSize	
for ( i = 0; i < tx_depth; i++ )	
TxSize = Split_Tx_Size[ TxSize ]	
} else {	
if ( MiSize == BLOCK_4X4 )	
TxSize = Min( maxTxSize, largestTxSize )	
else if ( Tx_Size_Sqr[ maxRectTxSize ] <= largestTxSize )	
TxSize = maxRectTxSize	
else	
TxSize = largestTxSize	
}	
} else {	
TxSize = maxRectTxSize	
}	
}	

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,
}
```

The Tx\_Mode\_To\_Biggest\_Tx\_Size table is defined as:

```
Tx_Mode_To_Biggest_Tx_Size[ TX_MODES ] = {
    TX_4X4,
    TX_64X64,
    TX_64X64
}
```

### 5.9.15. 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) {	
MinTxSize = TX_32X32	
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	
MinTxSize = TxSize	
}	
}	

## 5.9.16. Var TX Size Syntax

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

The original transform size may be square or rectangular, but only square transform sizes are produced if split.

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 ]	
step4 = Tx_Width[ subTxSz ] / MI_SIZE	
for (i = 0; i < h4; i += step4)	
for (j = 0; j < w4; j += step4)	
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	
MinTxSize = ( Tx_Width[ MinTxSize ] * Tx_Height[ MinTxSize ] <=	
Tx_Width[ txSz ] * Tx_Height[ txSz ] ) ? MinTxSize : txSz	
TxSize = txSz	
}	
}	

## 5.9.17. 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( )	
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( )	
}	

## 5.9.18. 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	
}	
}	

### 5.9.19. 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 {	
is_inter	S()
}	
}	

### 5.9.20. 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	
}	

### 5.9.21. Intra Block Mode Info Syntax

intra_block_mode_info( ) {	Type
----------------------------	------

RefFrame[ 0 ] = INTRA_FRAME	
RefFrame[ 1 ] = NONE	
<b>y_mode</b>	S()
YMode = y_mode	
intra_angle_info_y( )	
if (HasChroma) {	
<b>uv_mode</b>	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( )	
}	

## 5.9.22. Inter Block Mode Info Syntax

inter_block_mode_info( ) {	<b>Type</b>
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 ) {	
<b>compound_mode</b>	S()
YMode = NEAREST_NEARESTMV + compound_mode	
} else {	
<b>new_mv</b>	S()
if ( new_mv == 0 ) {	
YMode = NEWMV	
} else {	
<b>zero_mv</b>	S()
if ( zero_mv == 0 ) {	
YMode = GLOBALMV	
} else {	
<b>ref_mv</b>	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( ) {
    if ( skip_mode || motion_mode == LOCALWARP ) {
        return 0
    } else if (YMode == GLOBALMV) {
        return GmType[ RefFrame[ 0 ] ] <= TRANSLATION
    } else if (YMode == GLOBAL_GLOBALMV ) {
        return GmType[ RefFrame[ 0 ] ] <= TRANSLATION || GmType[ RefFrame[ 1 ] ] <= TRANSLATION
    } else {
        return 1
    }
}
```

### 5.9.23. 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()
}	
}	
}	

### 5.9.24. 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 {	
if ( reference_select )	
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 ?	
LAST_FRAME : LAST2_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	
}	
}	
}	

### 5.9.25. Assign MV Syntax

assign_mv( isCompound ) {	Type
bw = Block_Width[ MiSize ]	
bh = Block_Height[ MiSize ]	
Mv[ 1 ] = ZeroMvs[ 1 ]	
PredMv[ 1 ] = ZeroMvs[ 1 ]	
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 ] = (-sbSize - INTRABC_DELAY_PIXELS) * 8	
} else {	
PredMv[ 0 ][ 0 ] = -sbSize * 8	
PredMv[ 0 ][ 1 ] = 0	
}	
}	
else if ( compMode == GLOBALMV ) {	
PredMv[ i ] = ZeroMvs[ 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 )	
}	
}	
}	

## 5.9.26. 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    !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
}

```

## 5.9.27. Read Inter Intra Syntax

read_interintra_mode( isCompound ) {	Type
if ( !skip_mode && enable_interintra_compound && !isCompound &&	
MiSize >= BLOCK_8X8 && MiSize <= BLOCK_32X32 ) {	
interintra	S()
if (interintra) {	
interintra_mode	S()
RefFrame[1] = INTRA_FRAME	
AngleDeltaY = 0	
AngleDeltaUV = 0	
n = Wedge_Bits[ MiSize ]	
if ( n > 0 ) {	
wedge_interintra	S()
if (wedge_interintra) {	
wedge_index	S()
wedge_sign = 0	
}	
} else {	
wedge_interintra = 0	
}	
}	
} else {	
interintra = 0	
}	
}	

## 5.9.28. Read Compound Type Syntax

read_compound_type( isCompound ) {	Type
comp_group_idx = 0	
compound_idx = 1	
if ( skip_mode ) {	
compound_type = COMPOUND_AVERAGE	
return	
}	
if ( isCompound ) {	
if ( enable_masked_compound && Wedge_Bits[ MiSize ] > 0 ) {	
comp_group_idx	S()
} else {	
comp_group_idx = 0	
}	
if ( comp_group_idx == 0 ) {	
if ( enable_jnt_comp ) {	
compound_idx	S()
compound_type = compound_idx ? COMPOUND_DISTANCE :	
COMPOUND_AVERAGE	
} else {	
compound_type = COMPOUND_AVERAGE	
}	
return	
}	
}	
if ( enable_masked_compound && isCompound &&	
motion_mode == SIMPLE && MiSize >= BLOCK_8X8 ) {	
n = Wedge_Bits[ MiSize ]	
if ( n == 0 ) {	
compound_type = COMPOUND_SEG	
} else {	
compound_type	S()
}	
if ( compound_type == COMPOUND_WEDGE ) {	
wedge_index	S()
wedge_sign	L(1)
} else if ( compound_type == COMPOUND_SEG ) {	
mask_type	L(1)
}	
} else {	
if ( interintra ) {	
compound_type = wedge_interintra ? COMPOUND_WEDGE : COMPOUND_INTRA	
} else {	
compound_type = COMPOUND_AVERAGE	
}	
}	
}	

## 5.9.29. 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	
}	

## 5.9.30. 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 ]	
}	

## 5.9.31. 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	
}	

### 5.9.32. 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 ]	
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 = (miColChunk >> subX) * MI_SIZE	
baseY = (miRowChunk >> subY) * MI_SIZE	
candRow = (MiRow >> subY) << subY	
candCol = (MiCol >> subX) << subX	
IsInterIntra = ( is_inter && RefFrame[ 1 ] == INTRA_FRAME )	
IsCFL = (plane > 0 && !is_inter && UVMode == UV_CFL_PRED)	
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	
} else {	
mode = DC_PRED	
}	
if ( IsInterIntra    IsCFL ) {	
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[ miSizeChunk ] >> subX	
predH = Block_Height[ miSizeChunk ] >> subY	
someUseIntra = 0	
for( r = 0; r < num4x4H; r++ )	



for( c = 0; c < num4x4W; 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++	
}	
}	
if ( is_inter ) {	
transform_tree( plane, 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 ),	
y + ( chunkY << 4 ) )	
}	
}	
}	
}	
}	
}	

### 5.9.33. 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 if ( IsCFL ) {	
predict_chroma_from_luma( plane, startX, startY, txSz )	
} else {	
mode = ( plane == 0 ) ? YMode : 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 ) + y - 1 ]	
[ ( subBlockMiCol >> subX ) + x + stepX ],	
BlockDecoded[ plane ]	
[ ( subBlockMiRow >> subY ) + y + stepY ]	
[ ( subBlockMiCol >> subX ) + x - 1 ],	
mode,	
log2W, log2H )	
}	
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 ) + y + i ]	
[ ( subBlockMiCol >> subX ) + x + j ] = 1	

}	
}	
}	

## 5.9.34. Transform Tree Syntax

transform\_tree is used to read a number of transform blocks arranged in a transform tree.

transform_tree( plane, startX, startY, w, h ) {	Type
subX = (plane > 0) ? subsampling_x : 0	
subY = (plane > 0) ? subsampling_y : 0	
maxX = (MiCols * MI_SIZE) >> subX	
maxY = (MiRows * MI_SIZE) >> subY	
if ( startX >= maxX    startY >= maxY ) {	
return	
}	
row = ( startY << subY ) >> MI_SIZE_LOG2	
col = ( startX << subX ) >> MI_SIZE_LOG2	
lumaTxSz = InterTxSizes[ row ][ col ]	
lumaW = Tx_Width[ lumaTxSz ]	
lumaH = Tx_Height[ lumaTxSz ]	
uses64 = w >= 64    h >= 64	
isSubsampled = subX    subY	
if ( (isSubsampled && !uses64)	
(!isSubsampled && w <= lumaW && h <= lumaH) ) {	
txSz = find_tx_size( w, h )	
transform_block( plane, startX, startY, txSz, 0, 0 )	
} else {	
if ( w > h ) {	
transform_tree( plane, startX, startY, w/2, h )	
transform_tree( plane, startX + w / 2, startY, w/2, h )	
} else if ( w < h ) {	
transform_tree( plane, startX, startY, w, h/2 )	
transform_tree( plane, startX, startY + h/2, w, h/2 )	
} else {	
transform_tree( plane, startX, startY, w/2, h/2 )	
transform_tree( plane, startX + w/2, startY, w/2, h/2 )	
transform_tree( plane, startX, startY + h/2, w/2, h/2 )	
transform_tree( plane, startX + w/2, startY + h/2, w/2, h/2 )	
}	
}	
}	
}	

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
}

```

### 5.9.35. Get TX Size Function

get_tx_size( plane, txSz ) {	Type
if ( plane == 0 )	
return txSz	
uvTx = Max_Tx_Size_Rect[ get_plane_residual_size( MiSize, plane ) ]	
if ( Tx_Width[ uvTx ] == 64    Tx_Height[ uvTx ] == 64 )	
return TX_32X32	
return uvTx	
}	

### 5.9.36. 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.)

get_plane_residual_size( subsize, plane ) {	Type
subx = plane > 0 ? subsampling_x : 0	
suby = plane > 0 ? subsampling_y : 0	
return Subsampled_Size[ subsize ][ subx ][ suby ]	
}	

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} },
}

```

### 5.9.37. 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 = Tx_Width_Log2[ txSz ] + Tx_Height_Log2[ txSz ] - 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-- ) {	
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[ scan [ c ] ] = level	
}	
for ( c = 0; c < eob; c++ ) {	
pos = scan[ c ]	
if ( Quant[ pos ] != 0 ) {	
culLevel += Quant[ pos ]	
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 )	

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

### 5.9.38. Compute Transform Type Function

<pre> compute_tx_type( plane, txSz, blockX, blockY ) {     txSzSqrUp = Tx_Size_Sqr_Up[ txSz ]      if ( Lossless    txSzSqrUp &gt; TX_32X32 )         return DCT_DCT      txSet = get_tx_set( txSz )      if ( plane == 0 ) {         txType = TxTypes[ blockY ][ blockX ]         if ( !Tx_Type_In_Set[ txSet ][ txType ] )             return DCT_DCT         return txType     } } </pre>	Type
--	------



if ( RefFrame[ 0 ] != INTRA_FRAME ) {	
txType = TxTypes[ blockY << subsampling_y ][ blockX << subsampling_x ]	
if ( !Tx_Type_In_Set[ txSet ][ txType ] )	
return DCT_DCT	
return txType	
}	
txType = Mode_To_Txfrm[ UVMode ]	
if ( !Tx_Type_In_Set[ txSet ][ txType ] )	
return DCT_DCT	
return txType	
}	

### 5.9.39. 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_16X32 )	
return Mrow_Scan_16x32	
else if ( txSz == TX_32X16 )	
return Mrow_Scan_32x16	
else if ( txSz == TX_4X16 )	
return Mrow_Scan_4x16	
else if ( txSz == TX_16X4 )	
return Mrow_Scan_16x4	
else if ( txSz == TX_8X32 )	
return Mrow_Scan_8x32	
else if ( txSz == TX_32X8 )	
return Mrow_Scan_32x8	
return Mrow_Scan_32x32	
}	
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_16X32 )	
return Mcol_Scan_16x32	
else if ( txSz == TX_32X16 )	
return Mcol_Scan_32x16	
else if ( txSz == TX_4X16 )	
return Mcol_Scan_4x16	
else if ( txSz == TX_16X4 )	
return Mcol_Scan_16x4	
else if ( txSz == TX_8X32 )	
return Mcol_Scan_8x32	
else if ( txSz == TX_32X8 )	
return Mcol_Scan_32x8	
return Mcol_Scan_32x32	
}	
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 )	
}	

## 5.9.40. 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	
}	
}	
}	

## 5.9.41. Intra Angle Info Chroma Syntax

	Type
<code>intra_angle_info_uv( ) {</code>	
<code>AngleDeltaUV = 0</code>	
<code>if ( MiSize &gt;= BLOCK_8X8 ) {</code>	
<code>if ( is_directional_mode( UVMode ) ) {</code>	
<code>angle_delta_uv</code>	S()
<code>AngleDeltaUV = angle_delta_uv - MAX_ANGLE_DELTA</code>	
<code>}</code>	
<code>}</code>	
<code>}</code>	

## 5.9.42. Is Directional Mode Function

	Type
<code>is_directional_mode( mode ) {</code>	
<code>if ( ( mode &gt;= V_PRED ) &amp;&amp; ( mode &lt;= D67_PRED ) ) {</code>	
<code>return 1</code>	
<code>}</code>	
<code>return 0</code>	
<code>}</code>	

## 5.9.43. Read CFL Alphas Syntax

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

## 5.9.44. Palette Mode Info Syntax

	Type
<code>palette_mode_info( ) {</code>	
<code>bsizeCtx = Mi_Width_Log2[ MiSize ] + Mi_Height_Log2[ MiSize ]</code>	
<code>if ( YMode == DC_PRED ) {</code>	
<code>has_palette_y</code>	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, ceil( log2( range ) ) )	
idx++	
}	
sort( palette_colors_y, 0, PaletteSizeY - 1 )	
}	
if ( UVMode == DC_PRED && HasChroma ) {	
has_palette_uv	S()
if ( has_palette_uv ) {	
palette_size_uv_minus_2	S()
PaletteSizeUV = palette_size_uv_minus_2 + 2	
cacheN = get_palette_cache( 1 )	
idx = 0	
for ( i = 0; i < cacheN && idx < PaletteSizeUV; i++ ) {	
use_palette_color_cache_u	L(1)
if ( use_palette_color_cache_u ) {	
palette_colors_u[ idx ] = PaletteCache[ i ]	
idx++	
}	
}	
if ( idx < PaletteSizeUV ) {	

palette_colors_u[ idx ]	L(BitDepth)
idx++	
}	
if ( idx < PaletteSizeUV ) {	
minBits = BitDepth - 3	
palette_num_extra_bits_u	L(2)
paletteBits = minBits + palette_num_extra_bits_u	
}	
while ( idx < PaletteSizeUV ) {	
palette_delta_u	L(paletteBits)
palette_colors_u[ idx ] =	
Clip1( palette_colors_u[ idx - 1 ] +	
palette_delta_u )	
range = ( 1 << BitDepth ) - palette_colors_u[ idx ]	
paletteBits = Min( paletteBits, ceil( log2( range ) ) )	
idx++	
}	
sort( palette_colors_u, 0, PaletteSizeUV - 1 )	
delta_encode_palette_colors_v	L(1)
if ( delta_encode_palette_colors_v ) {	
minBits = BitDepth - 4	
maxVal = 1 << BitDepth	
palette_num_extra_bits_v	L(2)
paletteBits = minBits + palette_num_extra_bits_v	
palette_colors_v[ 0 ]	L(BitDepth)
for ( idx = 1; idx < PaletteSizeUV; idx++ ) {	
palette_delta_v	L(paletteBits)
if ( palette_delta_v ) {	
palette_delta_sign_bit_v	L(1)
if ( palette_delta_sign_bit_v ) {	
palette_delta_v = -palette_delta_v	
}	
}	
val = palette_colors_v[ idx - 1 ] + palette_delta_v	
if ( val < 0 ) val += maxVal	
if ( val >= maxVal ) val -= maxVal	
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 ) && AvailU ) {	
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.

## 5.9.45. 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 &&	
!skip &&	
!seg_feature_active( SEG_LVL_SKIP ) ) {	
if ( is_inter ) {	
inter_tx_type	S()
if ( set == 1 )	
TxType = Tx_Type_Inter_Inv_Set1[ inter_tx_type ]	
else if ( set == 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 == 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 }

```

## 5.9.46. 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	
}	
}	

## 5.9.47. 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 ) {	
color_index_map_y	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 )	

palette_color_idx_y	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 ) {	
color_index_map_uv	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 )	
palette_color_idx_uv	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 ]	

}	
}	
}	
}	

## 5.9.48. 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 ]	
}	
}	

## 5.9.49. 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 )	
}	

## 5.9.50. 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)	
}	

## 5.9.51. Check References Functions

This section defines some simple helper functions used in other parts of the specification:

check_backward(refFrame) {	Type
return ( (refFrame >= BWDREF_FRAME) && (refFrame <= ALTREF_FRAME) )	
}	
check_last_or_last2(refFrame) {	
return ( (refFrame == LAST_FRAME)    (refFrame == LAST2_FRAME) )	
}	
check_golden_or_last3(refFrame) {	
return ( (refFrame == GOLDEN_FRAME)    (refFrame == LAST3_FRAME) )	
}	
check_last(refFrame) {	
return (refFrame == LAST_FRAME)	
}	
check_last2(refFrame) {	
return (refFrame == LAST2_FRAME)	
}	

check_last3(refFrame) {	
return (refFrame == LAST3_FRAME)	
}	
check_golden(refFrame) {	
return (refFrame == GOLDEN_FRAME)	
}	

### 5.9.52. 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 )	
}	

### 5.9.53. 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 )	
}	

### 5.9.54. 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	
}	
}	

### 5.9.55. Read CDEF Syntax

read_cdef( ) {	Type
if ( skip    cdef_bits == 0 ) {	
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 ]	
}	
}	
}	
}	

## 5.9.56. Decode Loop Restoration Syntax

decode_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)	
continue	
subX = (plane == 0) ? 0 : subsampling_x	
subY = (plane == 0) ? 0 : subsampling_y	
unitSize = LoopRestorationSize[ plane ]	
tileLeft = upscale_mi_to_pos(MiColStart, plane)	
tileRight = upscale_mi_to_pos(MiColEnd, plane)	
unitRows = count_units_in_tile( unitSize,	
MiRowEnd - MiRowStart ) * ( MI_SIZE >> subY )	
unitCols = count_units_in_tile( unitSize, tileRight - tileLeft )	
relMiRow = r - MiRowStart	
relMiCol = c - MiColStart	
unitRowStart = ( relMiRow * ( MI_SIZE >> subY ) +	
unitSize - 1 ) / unitSize	
unitRowEnd = Min( unitRows, ( (relMiRow + 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 = ( relMiCol * numerator + denominator - 1 ) / denominator	
unitColEnd = Min( unitCols, ( (relMiCol + w) * numerator +	
denominator - 1 ) / denominator)	
for (unitRow = unitRowStart; unitRow < unitRowEnd; unitRow++) {	
for (unitCol = unitColStart; unitCol < unitColEnd; unitCol++) {	
decode_lr_unit(plane, unitRow, unitCol)	
}	

}	
}	
}	

where `count_units_in_tile` is a function specified as:

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

and `upscale_mi_to_pos` is a function that converts a x position (in units of 4x4 luma blocks) to an upsampled sample position in the specified plane specified as:

```
upscale_mi_to_pos( miX, plane ) {
    if (plane > 0) {
        subX = subsampling_x
    } else {
        subX = 0
    }
    if ( miX == MiCols ) {
        return Round2( UpscaledWidth, subX )
    }
    downscaleX = ( miX * MI_SIZE ) >> subX
    if ( use_superres ) {
        upscaleX = ( downscaleX * SuperresDenom ) / SUPERRES_NUM
    } else {
        upscaleX = downscaleX
    }
    return upscaleX
}
```

### 5.9.57. Decode Loop Restoration Unit Syntax

<code>decode_lr_unit(plane, unitRow, unitCol)</code> {	Type
if (FrameRestorationType[ plane ] == RESTORE_NONE) {	
restoration_type = RESTORE_NONE	
} else 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[ TileNum ][ plane ][ unitRow ][ unitCol ] = restoration_type	
if (restoration_type == RESTORE_WIENER) {	

for ( pass = 0; pass < 2; pass++ ) {	
if (plane) {	
firstCoeff = 1	
LrWiener[ TileNum ][ 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[ TileNum ][ plane ]	
[ unitRow ][ unitCol ][ pass ][ j ] = v	
RefLrWiener[ plane ][ pass ][ j ] = v	
}	
}	
} else if (restoration_type == RESTORE_SGRPROJ) {	
lr_sgr_set	NS(SGRPROJ_PARAMS_BITS)
LrSgrSet[ TileNum ][ 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[ TileNum ][ 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 7.16.3](#) and decode\_signed\_subexp\_with\_ref\_bool is a function specified as follows:

	Type
decode_signed_subexp_with_ref_bool( low, high, k, r ) {	
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.

Draft Document

## 6. Syntax Structures Semantics

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

Important variables and function calls are also described.

### 6.1. 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.

**`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.

#### 6.1.1. 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:

obu_type	Name of obu_type
0	Reserved
1	OBU_SEQUENCE_HEADER
2	OBU_TD
3	OBU_FRAME_HEADER
4	OBU_TILE_GROUP
5	OBU_METADATA
6	OBU_FRAME
7	OBU_REDUNDANT_FRAME_HEADER
8-14	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.

## 6.1.2. 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' <= T and S' <= S.

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

### 6.1.3. Trailing Bits Semantics

**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.

### 6.1.4. 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.

## 6.2. 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.

## 6.3. Sequence Header OBU Semantics

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

AV1 supports 3 profiles:

seq_profile	Bit depth	SRGB Colorspace support	Chroma subsampling
0	8 or 10	No	YUV 4:2:0
1	8 or 10	Yes	YUV 4:4:4
2	8 or 10	No	YUV 4:2:2
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 equal to 3.

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

**reserved\_zero** shall be equal to 0.

**operating\_points\_minus1\_cnt** indicates the number of operating points minus 1 present in this bitstream.

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 i is equal to 1 if temporal layer i should be decoded (for i 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 bitstream has no scalability information in OBU extension headers and the operating point applies to the entire bitstream. This means that all OBUs must be decoded.

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

**decoder\_rate\_model\_param\_present\_flag[ i ]** indicates if the decoder rate model parameters are present in the bitstream.

**decode\_to\_display\_rate\_ratio**, **initial\_display\_delay**, and **extra\_frame\_buffers** define parameters of the decoder rate model that correspond to this operating point, but do not affect the decoding process.

**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\_minus1\_cnt**, 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.

**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 bitstream.

**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\_minus1** is used to calculate the number of bits used to encode the `frame_id` syntax element.

**delta\_frame\_id\_length\_minus2** 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 are 128x128 pixels. When equal to 0, it indicates that superblocks are 64x64 pixels.

**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\_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\_minus1** 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).

**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 legal 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\_cdef** equal to 0 specifies that loop restoration filtering is disabled.

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

**timing\_info\_present\_flag** specifies whether timing info is present in the bitstream.

**film\_grain\_params\_present** specifies whether film grain parameters are present in the bitstream.

### 6.3.1. 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\_coeffs** are present. **color\_description\_present\_flag** equal to 0 specifies that **color primaries**, **transfer characteristics** and **matrix\_coeffs** are not present.

**color\_primaries** is an integer that is defined by the “Color primaries” section of ISO/IEC 23001-8:2016.

<b>color_primaries</b>	<b>Name of color primaries</b>	<b>Description</b>
1	CP_BT_709	BT.709
2	CP_UNSPECIFIED	Unspecified



color_primaries	Name of color primaries	Description
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 23001-8:2016.

transfer_characteristics	Name of transfer characteristics	Description
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
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

transfer_characteristics	Name of transfer characteristics	Description
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 23001-8:2016.

matrix_coefficients	Name of matrix coefficients	Description
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
14	MC_ICTCP	BT.2100 ICtCp

If (color primaries == CP\_BT\_709 and transfer\_characteristics == TC\_SRGB and matrix\_coefficients == MC\_IDENTITY), it is a requirement of bitstream conformance that ( seq\_profile == 1 || (seq\_profile == 2 && BitDepth==12) ).

**color\_range** specifies the black level and range of the luma and chroma signals as specified in Rec. ITU-R BT.709-6 and Rec. ITU-R BT.2020-2:

color_range	Description	Details
0	Studio swing	For BitDepth equals 8: Y is between 16 and 235 inclusive. U and V are between 16 and 240 inclusive.

color_range	Description	Details
		For BitDepth equals 10: Y is between 64 and 940 inclusive. U and V are between 64 and 960 inclusive.
		For BitDepth equals 12: Y is between 256 and 3760. U and V are between 256 and 3840 inclusive.
1	Full swing	No restriction on Y, U, V values.

**Note:** Note that this specification does not enforce the range of YUV values when the YUV range is 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:

subsampling_x	subsampling_y	Description
0	0	YUV 4:4:4
0	1	YUV 4:4:0
1	0	YUV 4:2:2
1	1	YUV 4:2:0

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

chroma_sample_position	Name of chroma sample position	Description
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.

## 6.3.2. Timing Info Semantics

**num\_units\_in\_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 clock tick, in seconds, is equal to `num_units_in_tick` divided by `time_scale`.

It is a requirement of bitstream conformance that `num_units_in_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_minus1` + 1. `equal_picture_interval` equal to 0 indicates that the interval between two consecutive pictures is not specified.

**num\_ticks\_per\_picture\_minus1** 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_minus1` 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. Temporal Delimiter OBU Semantics

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

## 6.5. 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.

## 6.6. Metadata OBU Semantics

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

<b>metadata_type</b>	<b>Name of metadata_type</b>
0	METADATA_TYPE_PRIVATE_DATA

metadata_type	Name of metadata_type
1	METADATA_TYPE_HDR_CLL
2	METADATA_TYPE_HDR_MDCV
3	METADATA_TYPE_SCALABILITY
4-65535	Reserved

### 6.6.1. Metadata Private Data Semantics

Private data allows the transfer of arbitrary, application-specific binary information.

Private data may have arbitrary values and have no effect on the decoding process.

### 6.6.2. 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.

### 6.6.3. 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[ i ]** specifies a 0.16 fixed-point white X chromaticity coordinate as defined by CIE 1931.

**white\_point\_chromaticity\_y[ i ]** 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.

### 6.6.4. Metadata Scalability Semantics

**scalability\_mode\_idc** indicates the picture prediction structure of the bitstream.

scalability_mode_idc	Name of scalability_mode_idc
0	SCALABILITY_L1T2
1	SCALABILITY_L1T3

scalability_mode_idc	Name of scalability_mode_idc
2	SCALABILITY_L2T1
3	SCALABILITY_L2T2
4	SCALABILITY_L2T3
5	SCALABILITY_S2T1
6	SCALABILITY_S2T2
7	SCALABILITY_S2T3
8	SCALABILITY_L2T2h
9	SCALABILITY_L2T3h
10	SCALABILITY_S2T1h
11	SCALABILITY_S2T2h
12	SCALABILITY_S2T3h
13	SCALABILITY_SS
14-255	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 2 or 3 temporal layers, in combinations with zero or one spatial layer. The spatial layer may have twice or one and a half times the resolution of the base layer in each dimension, depending on the mode.

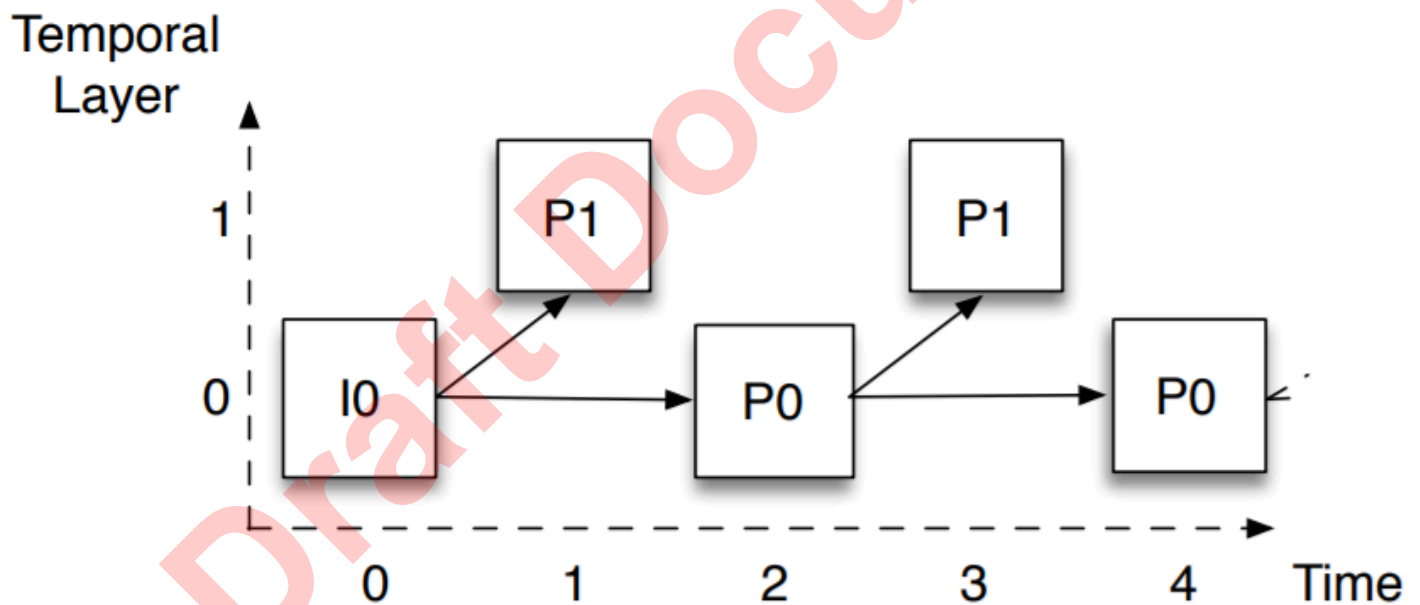
There is also support for an spatial layer that uses no inter-layer prediction (i.e., the spatial layer does not use its corresponding base layer as a reference).

The following table lists the predefined scalability structures.

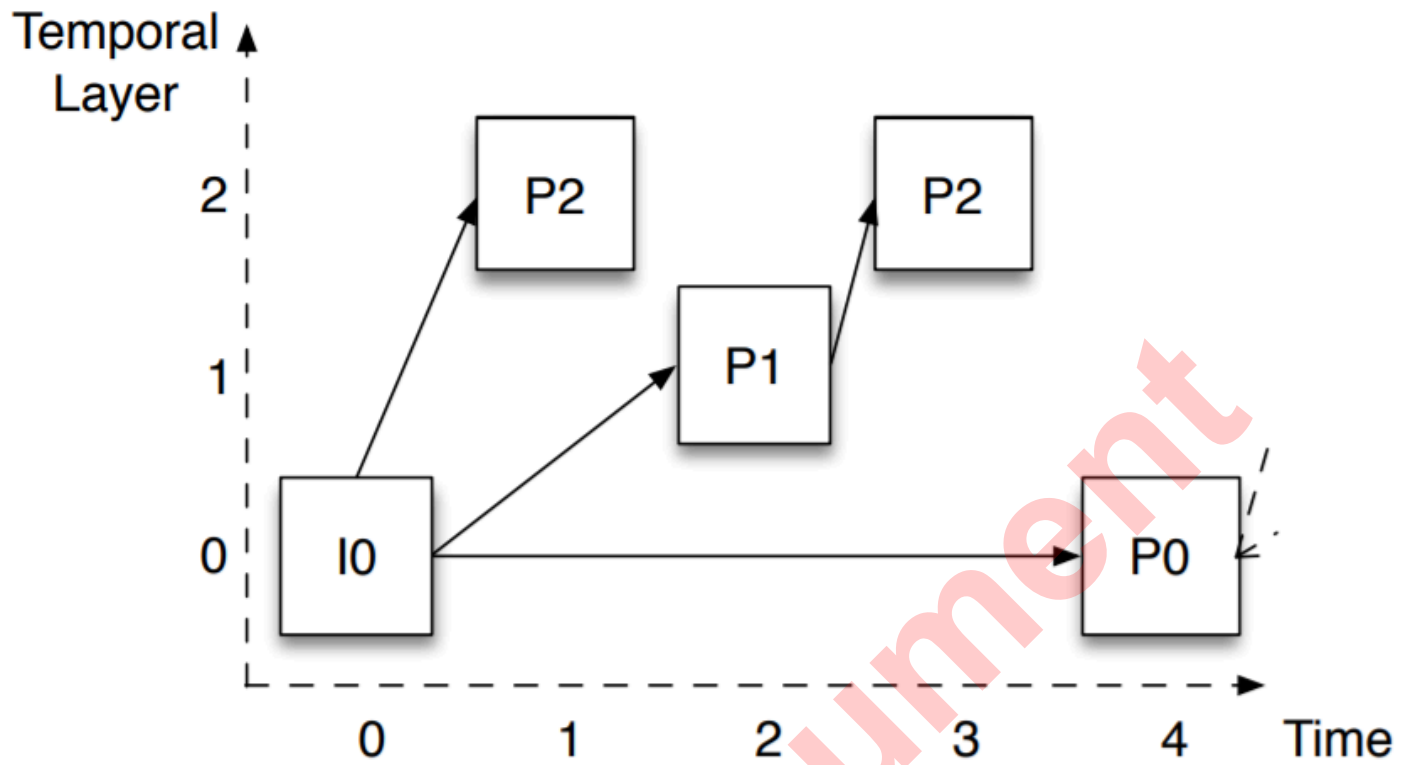
Name of scalability_mode_idc	Spatial Layers	Resolution Ratio	Temporal Layers	Inter-layer dependency
SCALABILITY_L1T2	1		2	
SCALABILITY_L1T3	1		3	
SCALABILITY_L2T1	2	2:1	1	Yes

Name of scalability_mode_idc	Spatial Layers	Resolution Ratio	Temporal Layers	Inter-layer dependency
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_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

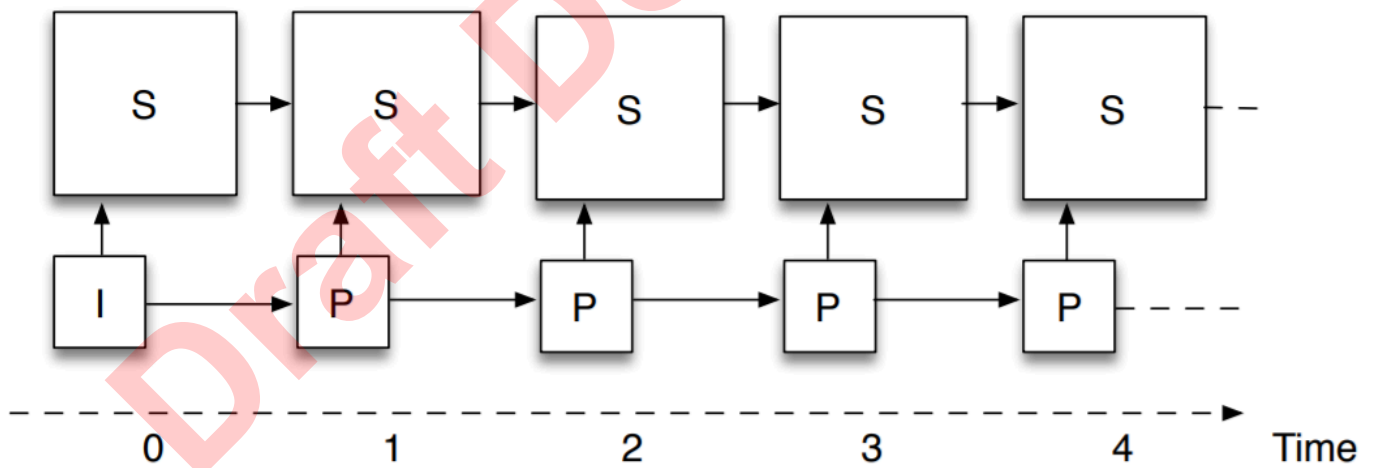
The following figures show the picture prediction structures for certain modes:



L1T2

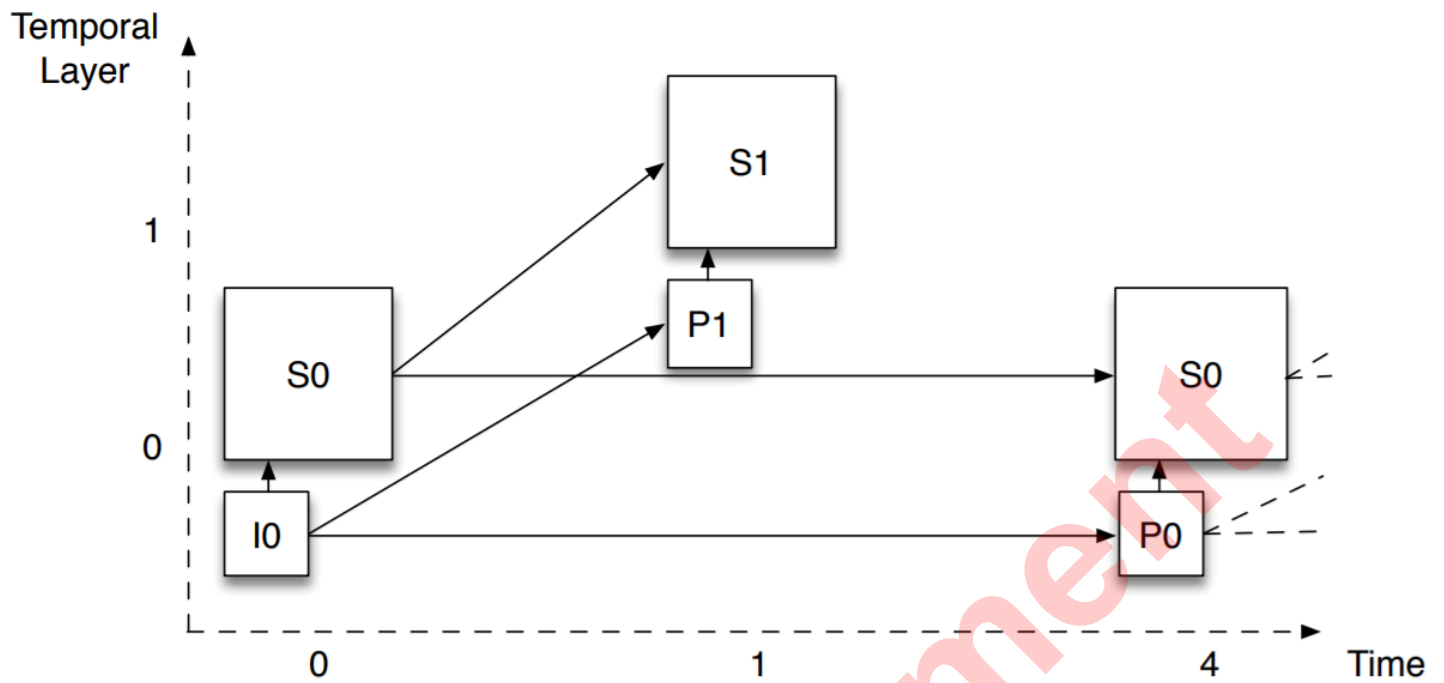


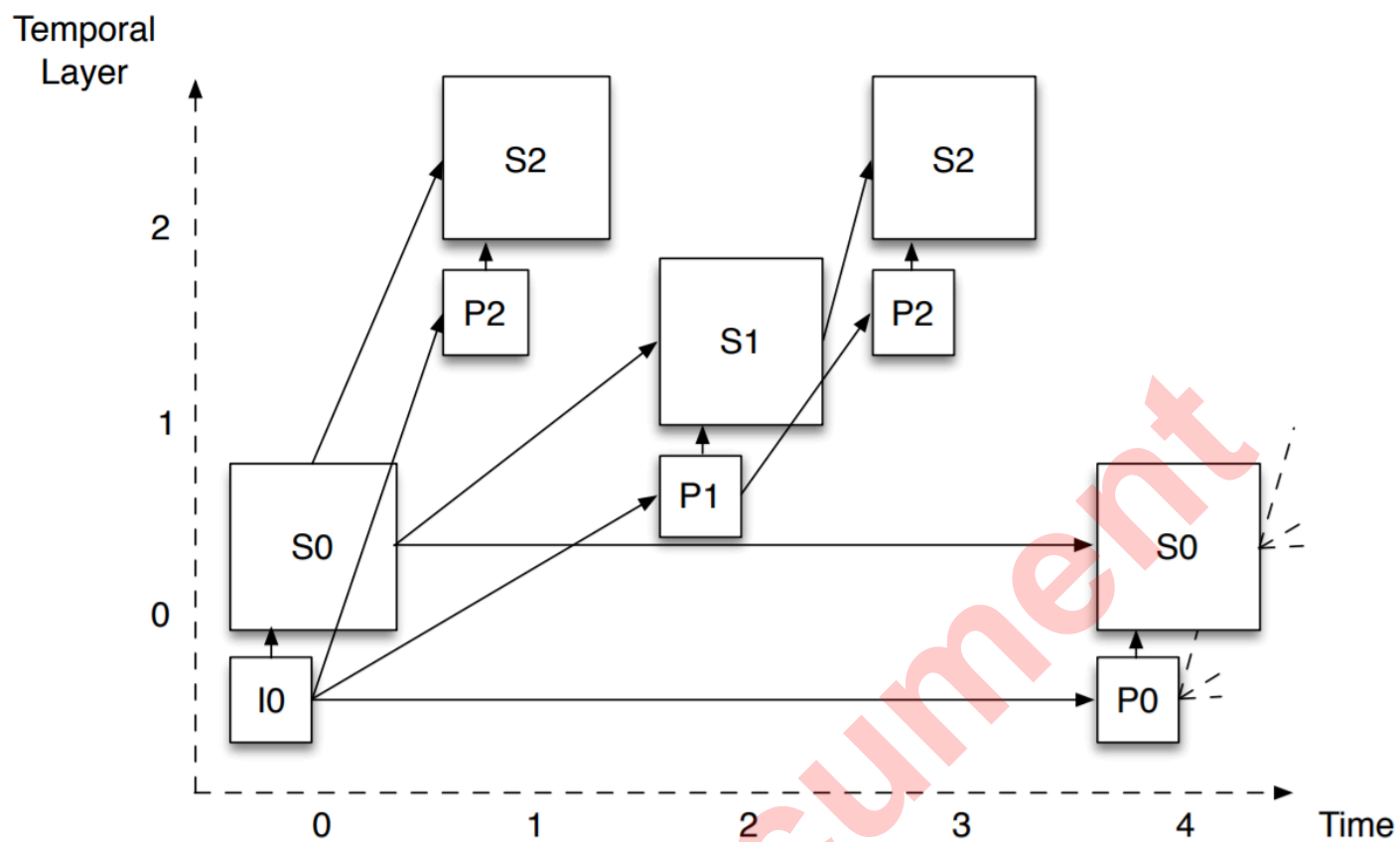
L1T3



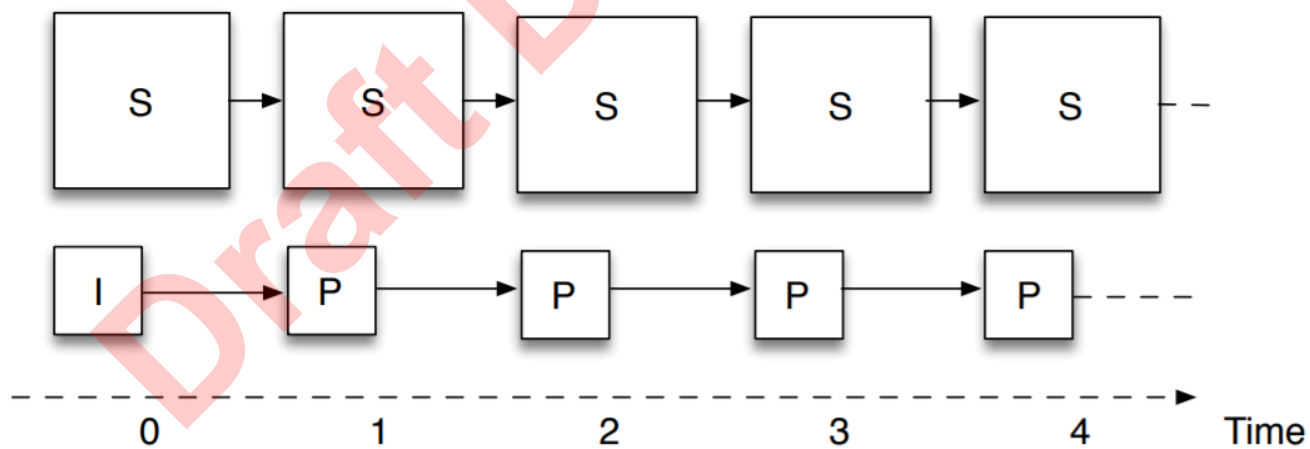
L2T1



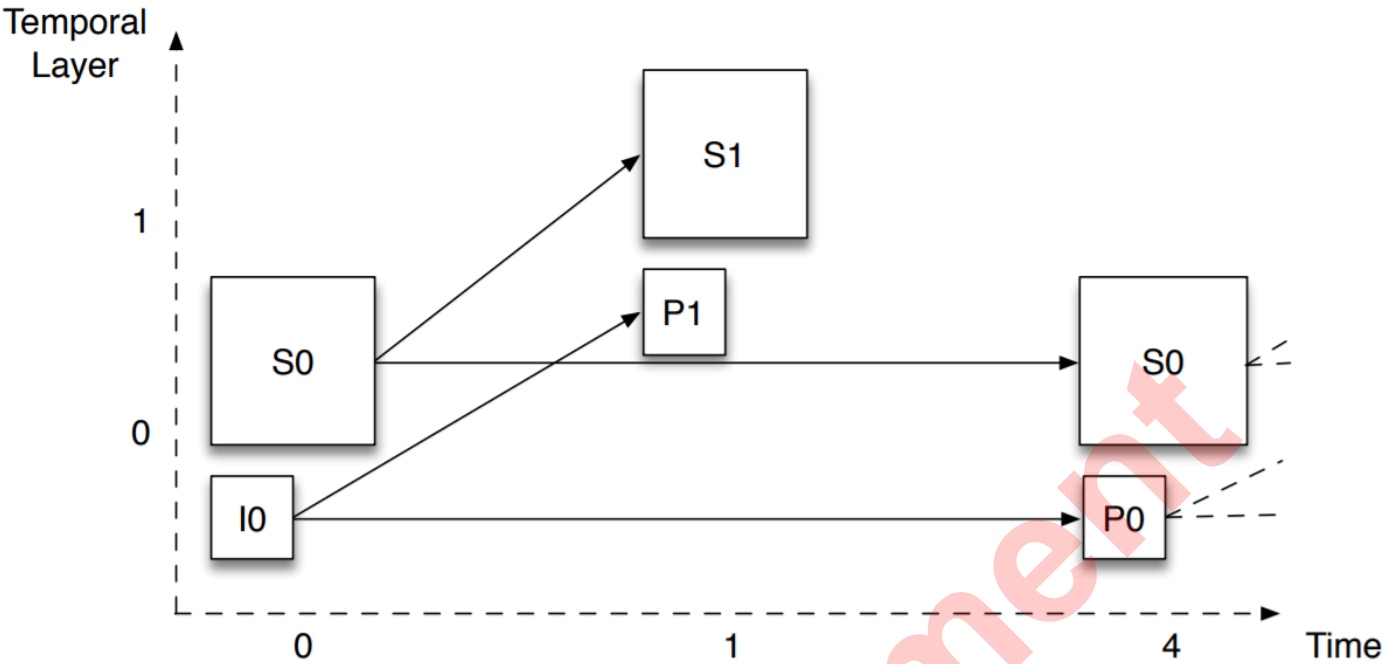




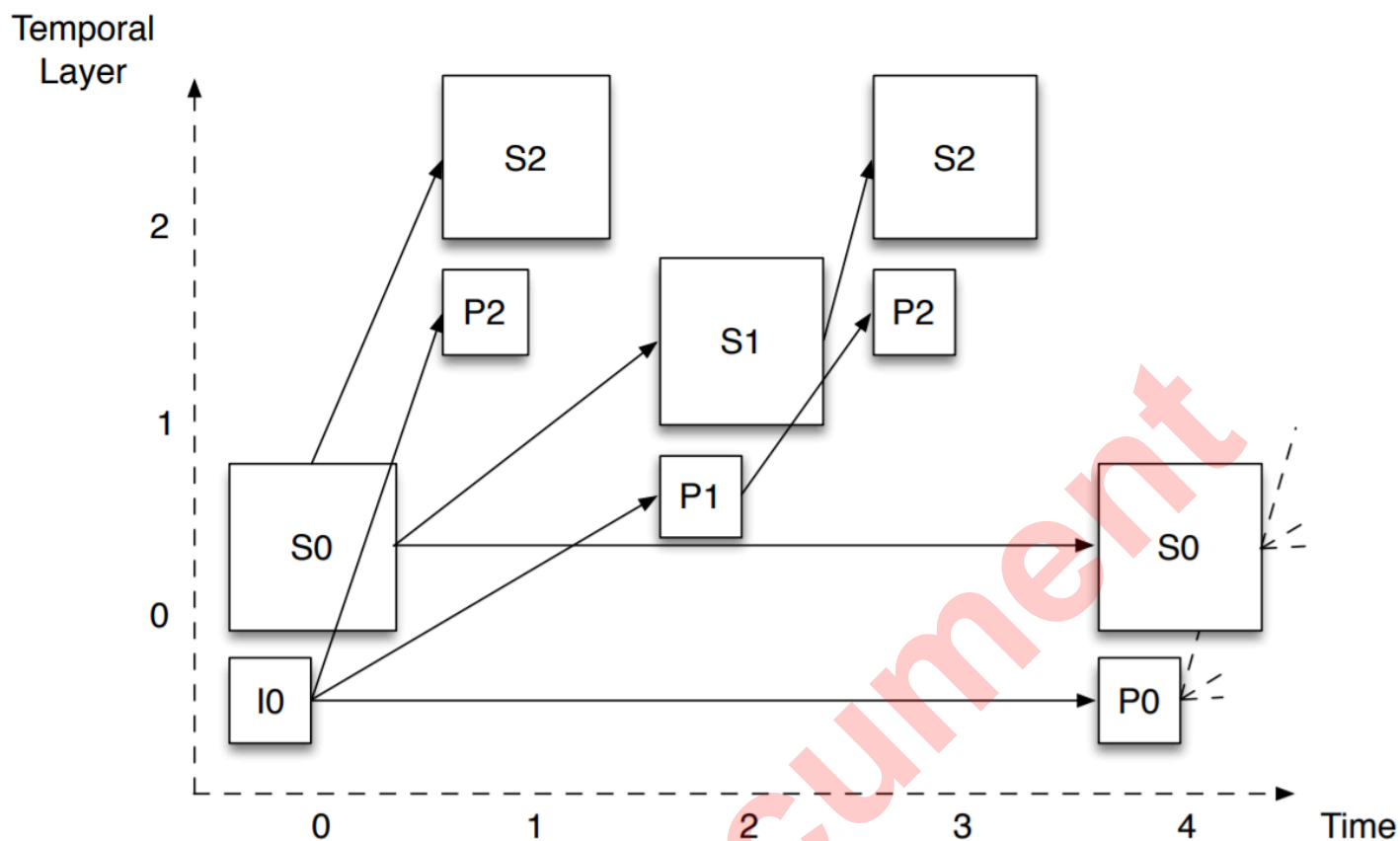
L2T3



S2T1



S2T2



S2T3

### 6.6.5. Scalability Structure Semantics

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

**`spatial_layer_description_present_flag`** indicates when set to 1 that the `spatial_layer_ref_id` is present for each of the `spatial_layers_cnt` 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 `spatial_layers_cnt` 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.

## 6.7. 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.

**MaxTileSize** is a variable that tracks the largest size in bytes of each tile. (This is needed because the CDFs are updated based on the largest tile decoded.)

**decode\_frame** is a function call that indicates that the decode frame process specified in [section 7.3](#) should be invoked.

### 6.7.1. 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.

**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 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`  $\leq$  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 a particular **showable\_frame** is output via the **show\_existing\_frame** mechanism at most once.

**Note:** This requirement also forbids storing a frame into multiple reference buffers and then using **show\_existing\_frame** for each buffer.

**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 decoded independently of previous 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 1 specifies that the frame size will be contained in the bitstream.

frame\_size\_override\_flag equal to 0 specifies that the frame size is equal to the size in the sequence header.

**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.

**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 buffer.

It is a requirement of bitstream conformance that ref\_order\_hint[ i ] is equal to RefOrderHint[ i ].

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

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



See [section 7.19](#) 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 7.7](#) is invoked.

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

**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\_minus1** 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. It is a requirement of bitstream conformance that whenever RefFrameId[ i ] is calculated, the value matches 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 7.8](#) 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.

**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..7 and j = 0..3.
- 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.
- gm\_params[ ref ][ i ] is set equal to  $((i \% 3 == 2) ? 1 << \text{WARPEDMODEL\_PREC\_BITS} : 0)$  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`
- `TxfrmSplitCdf` 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
- DeltaLFMultiCdf[ 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. `gm_params` is set equal to `SavedGmParams[ prevFrame ]`.
3. 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`.

4. The function `load_loop_filter_params( prevFrame )` specified in [section 7.20](#) is invoked.
5. The function `load_segmentation_params( prevFrame )` specified in [section 7.20](#) is invoked.

## 6.7.2. 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.

## 6.7.3. Frame Size Semantics

`frame_width_minus_1` plus one is the width of the frame in pixels.

**frame\_height\_minus\_1** plus one is the height of the frame in pixels.

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**.

## 6.7.4. 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 in the bitstream.

**Note:** It is legal 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 pixels.

**render\_height\_minus\_1** plus one is the render height of the frame in pixels.

## 6.7.5. 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.

## 6.7.6. 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**.

### 6.7.7. 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.

### 6.7.8. 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
3	BILINEAR
4	SWITCHABLE

### 6.7.9. 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 7.13](#).

**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 the bitstream contains additional syntax elements 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.

If **primary\_ref\_frame** is equal to **PRIMARY\_REF\_NONE**, it is a requirement of bitstream conformance that **loop\_filter\_delta\_update** is equal to 1.

**update\_ref\_delta** equal to 1 means that the bitstream contains the syntax element `loop_filter_ref_delta`; `update_ref_delta` equal to 0 means that the bitstream does not contain this syntax element.

**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 in the bitstream, it maintains its previous value.

**update\_mode\_delta** equal to 1 means that the bitstream contains the syntax element `loop_filter_mode_deltas`; `update_mode_delta` equal to 0 means that the bitstream does not contain this syntax element.

**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 bitstream, 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.

## 6.7.10. 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 7.11](#).

**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.

## 6.7.11. Delta Quantizer Semantics

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

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

## 6.7.12. 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 in the bitstream.

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

## 6.7.13. 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** indicates whether the tile width is increased.

**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 pixels) for each tile across the image.

**MiRowStarts** is an array specifying the start row (in units of 4x4 pixels) 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).

**maxTileHeightMi** contains the maximum height (in units of 4x4 pixels) that can be used for a tile.

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

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

## 6.7.14. Quantizer Index Delta Parameters Semantics

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

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

## 6.7.15. Loop Filter Delta Parameters Semantics

**delta\_if\_present** specifies whether loop filter delta values are present in the bitstream.

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

**delta\_if\_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\_if\_multi** equal to 0 specifies that the same loop filter delta is used for all edges.

## 6.7.16. 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.

## 6.7.17. Global Param Semantics

**absBits** is used to compute the range of values that can be used for this parameter. 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 this parameter in the bitstream. All global motion parameters are stored in the model with `WARPEDMODEL_PREC_BITS` fractional bits, but the parameters are encoded with less precision in the bitstream.

## 6.7.18. 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 0 specifies that the parameter is greater than  $m_k+a-1$ .

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

## 6.7.19. Decode Uniform Semantics

**v** and **extra\_bit** are used to compute the value of an unsigned number in the range 0 to n-1.

## 6.7.20. 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 normally 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 normally read in **film\_grain\_params** should be set equal to the values stored in an area of memory indexed by **idx**.

It is a requirement of bitstream conformance that **load\_grain\_params(idx)** is invoked, then there must have been a previous frame in decoding order for which **save\_grain\_params(idx)** was invoked (this ensures that the parameters are always defined before being used).

**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 legal 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.

## 6.7.21. TX Mode Semantics

**tx\_mode\_select** is used to compute TxMode.

**TxMode** specifies how the transform size is determined:

<b>TxMode</b>	<b>Name of TxMode</b>
0	ONLY_4X4
1	TX_MODE_LARGEST



TxMode	Name of TxMode
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.

## 6.7.22. 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 coded in the bitstream.

`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.)

## 6.7.23. 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. 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.

## 6.9. 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 in the bitstream. If `tg_start` and `tg_end` are not present in the bitstream, this tile group covers the entire frame.

**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.

**update\_cdf** is a function call that indicates that the frame CDF arrays are set equal to the final saved CDFs at the end of a frame, based on saved CDF arrays updated at the end of the largest tile (measured in bytes). This process is described in [section 7.6](#).

**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** is a function call that indicates that the decode frame process specified in [section 7.3](#) should be invoked.

## 6.9.1. 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 is the first one in the current superblock. Delta values for the quantizer index and loop filter are only read on the first block of a superblock.

## 6.9.2. Clear Block Decoded Flags Semantics

**notLastColumn** equal to 1 indicates that the current superblock is not in the rightmost column of the current tile.

**notLastColumn** equal to 0 indicates that the current superblock is in the rightmost column of the current tile.

**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.

## 6.9.3. Decode Partition Semantics

**partition** specifies how a block is partitioned:

partition	Name of partition
0	PARTITION_NONE
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:

subSize	Name of subSize
0	BLOCK_4X4
1	BLOCK_4X8
2	BLOCK_8X4
3	BLOCK_8X8
4	BLOCK_8X16

subSize	Name of subSize
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
17	BLOCK_16X4
18	BLOCK_8X32
19	BLOCK_32X8
20	BLOCK_16X64
21	BLOCK_64X16
22	BLOCK_32X128
23	BLOCK_128X32

The dimensions of these blocks are given in width, height order (e.g. BLOCK\_8X16 corresponds to a block that is 8 pixels wide, and 16 pixels 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 legal 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 legal because of overlap with the bottom edge of the frame.

## 6.9.4. Decode Block Semantics

**MiRow** is a variable holding the vertical location of the block in units of 4x4 pixels.

**MiCol** is a variable holding the horizontal location of the block in units of 4x4 pixels.

**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.

## 6.9.5. 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:

intra_frame_y_mode	Name of intra_frame_y_mode
0	DC_PRED
1	V_PRED
2	H_PRED
3	D45_PRED
4	D135_PRED
5	D113_PRED
6	D157_PRED

intra_frame_y_mode	Name of intra_frame_y_mode
7	D203_PRED
8	D67_PRED
9	SMOOTH_PRED
10	SMOOTH_V_PRED
11	SMOOTH_H_PRED
12	TM_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.

uv_mode	Name of uv_mode
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	TM_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  $\text{Max}(\text{Block\_Width}[\text{MiSize}], \text{Block\_Height}[\text{MiSize}]) \leq 32$ .

## 6.9.6. 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.

## 6.9.7. 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).

## 6.9.8. 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 legal for **seg\_id\_predicted** to be equal to 0 even if the value coded for the **segment\_id** is equal to **predictedSegmentId**.

## 6.9.9. 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.

## 6.9.10. 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.

## 6.9.11. 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.

## 6.9.12. 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.

## 6.9.13. 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.

## 6.9.14. Loop Restoration Params Semantics

**lr\_type** is used to compute **FrameRestorationType**.

**FrameRestorationType** specifies the type of restoration used for each plane as follows:

lr_type	FrameRestorationType	Name of FrameRestorationType
0	0	RESTORE_NONE
1	3	RESTORE_SWITCHABLE
2	1	RESTORE_WIENER
3	2	RESTORE_SGRPROJ

**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.



## 6.9.15. 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:

TxSize	Name of TxSize
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
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.

**Note:** When TxSize is first computed, it will only take values corresponding to square sizes. Rectangular sizes are inferred if this square TxSize is larger than the block size. TxSize is then set to be the largest rectangular size that fits in the block.

## 6.9.16. Block TX Size Semantics

**MinTxSize** is a variable that contains a square transform size that would fit inside all the transforms within the block. In other words, if the block contains transforms of size  $W[i]$  by  $H[i]$ , then **MinTxSize** represents a transform of size  $m$  by  $m$ , where  $m$  is equal to the smallest value in both  $W$  and  $H$ .

**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.

## 6.9.17. 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.

## 6.9.18. Transform Type Semantics

**set** specifies the transform set.

is_inter	set	Name of transform set
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
DCT_DCT	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

Transform type	TX_SET_DCTONLY	TX_SET_INTRA_1	TX_SET_INTRA_2	TX_SET_INTER_1	TX_SET_INTER_2
ADST_FLIPADST				X	X
FLIPADST_ADST				X	X
IDTX		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.

## 6.9.19. 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.

## 6.9.20. 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  $\text{Max}(\text{Block\_Width}[\text{MiSize}], \text{Block\_Height}[\text{MiSize}]) \leq 32$ .

## 6.9.21. 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.

**RefMvIdx** 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”.

## 6.9.22. 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:

<b>filter_intra_mode</b>	<b>Name of filter_intra_mode</b>
0	INTRA_DC_PRED
1	INTRA_V_PRED
2	INTRA_H_PRED
3	INTRA_D153_PRED
4	INTRA_TM_PRED

## 6.9.23. Ref Frames Semantics

**comp\_mode** specifies whether single or compound prediction is used:

<b>comp_mode</b>	<b>Name of comp_mode</b>
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.

**comp\_ref\_type** is used for compound prediction to specify whether both reference frames come from the same direction or not:

<b>comp_ref_type</b>	<b>Name of comp_ref_type</b>	<b>Description</b>
0	UNIDIR_COMP_REFERENCE	Both reference frames from the same direction
1	BIDIR_COMP_REFERENCE	One forward and one backward reference frame

**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 direction.

**comp\_ref**, **comp\_ref\_p1**, and **comp\_ref\_p2** specify the first reference frame when the two reference frames come from different directions.

**comp\_bwdref** and **comp\_bwdref\_p1** specify the second reference frame when the two reference frames come from different directions.

**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
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.

## 6.9.24. 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:

<b>motion_mode</b>	<b>Name of motion_mode</b>
0	SIMPLE
1	OBMC
2	LOCALWARP

**Note:** A motion\_mode equal to SIMPLE is used for blocks requiring global motion.

## 6.9.25. 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:

<b>interintra_mode</b>	<b>Name of interintra_mode</b>
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.

## 6.9.26. 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:

compound_type	Name of compound_type
0	COMPOUND_WEDGE
1	COMPOUND_SEG
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:

mask_type	Name of mask_type
0	UNIFORM_45
1	UNIFORM_45_INV

## 6.9.27. 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:

mv_joint	Name of mv_joint	Changes row	Changes col
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. It is a requirement of bitstream conformance that the resulting motion vector satisfies  $-(1 << 14) < \text{Mv}[\text{ref}][\text{comp}] < (1 << 14) - 1$  for  $\text{comp} = 0..1$ .

It is a requirement of bitstream conformance that whenever read\_mv returns, the function is\_dv\_valid would return 1, where is\_dv\_valid is defined as:



```

is_dv_valid( ) {
    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 + deltaRow
    srcLeftEdge = MiCol + deltaCol
    srcBottomEdge = srcTopEdge + bh
    srcRightEdge = srcLeftEdge + bw
    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 / 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 such that, if `use_intrabc` is equal to 1, the motion vector is additionally constrained 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).

## 6.9.28. 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 in the bitstream for class 0 motion vector differences.

**mv\_class0\_fr** specifies the first 2 fractional bits of the motion vector difference. This is only present in the bitstream for class 0 motion vector differences.

**mv\_class0\_hp** specifies the third fraction bit of the motion vector difference. This is only present in the bitstream 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.

## 6.9.29. Residual Semantics

**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 7.10.1](#) is invoked.

**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 7.10.2](#) is invoked.

**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.)

**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. Normally, 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 a smaller block size for each of the corresponding luma blocks.

**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 7.10.3](#) is invoked.

**Note:** The **predict\_inter**, **predict\_intra**, **predict\_palette**, and **predict\_chroma\_from\_luma** functions do not affect the syntax decode process.

## 6.9.30. 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 7.11.2](#) 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 7.10.4](#) is invoked.

**MaxLumaW** and **MaxLumaH** are needed for chroma from luma prediction and store the extent of luma pixels 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.

## 6.9.31. Coefficients Semantics

**TxTypes** is an array which stores at a 4x4 pixel 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 the corresponding luma 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.

**eob\_extra** and **eob\_extra\_bit** specify the exact position of the last non-zero coefficient.

**Note:** The value of **eob** indicates the index of the end of block. This will be immediately following the last non-zero coefficient.

**coeff\_base\_eob** is used to compute the base level of the last non-zero coefficient.

**coeff\_base** specifies the base level of a coefficient.

**dc\_sign** specifies the sign of the DC coefficient.

**sign\_bit** specifies the sign of a non-zero 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.

**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).

## 6.9.32. 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.

### 6.9.33. 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 unnecessary.)

**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.

### 6.9.34. 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.

## 6.9.35. Palette Tokens Semantics

**color\_index\_map\_y** holds the index in **palette\_colors\_y** for the block's Y plane top left pixel.

**color\_index\_map\_uv** holds the index in **palette\_colors\_u** and **palette\_colors\_v** for the block's UV plane top left pixel.

**palette\_color\_idx\_y** holds the index in **ColorOrder** for a pixel in the block's Y plane.

**palette\_color\_idx\_uv** holds the index in **ColorOrder** for a pixel in the block's UV plane.

## 6.9.36. 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 neighbourhood of the current block, weighted by closeness to the current block.

**ColorContextHash** is a variable derived from the distribution of colors in the neighbourhood of the current block, which is used to determine the probability context used to decode **palette\_color\_idx\_y** and **palette\_color\_idx\_uv**.

## 6.9.37. Read CDEF Semantics

**cdef\_idx** specifies which CDEF filtering parameters should be used for a particular 64 by 64 pixel block. A value of -1 means that CDEF is disabled for that block.

## 6.9.38. Decode 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 0 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  $mk$ .

## 7. Decoding Process

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 data for a specific tile, output is decoded content of just that tile)

The general decoding process is specified in [section 7.1](#).

The large scale tile coding process is specified in [section 7.2](#).

### 7.1. 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 5.1](#).

The syntax tables include function calls indicating when the remaining decode processes are triggered.

### 7.2. Large Scale Tile Decoding Process

The large scale tile decoding process allows a decoder to extract only an interesting section in a frame without the need to decompress the entire frame, which helps reduce decoder complexity.

This feature is extremely useful for VR applications (e.g. light fields), which renders a single section of the frame following the viewers's head movement.

One expected use case will have a mixture of anchor frames and camera frames. Anchor frames can be decoded with the general decoding process, while camera frames are decoded with the large scale tile decoding process. The entire light field at a particular time will be represented by a small number of anchor frames (representing the view from a particular viewing position and gaze direction), plus a larger number of camera frames (representing the view from a larger number of positions and directions near to the anchor frame). Depending on the viewer's head position and direction, the application will first decode the whole of an anchor frame, and then just the portions of the camera frame that are required to generate the viewable part of the display.

The format of the container format for such applications is outside the scope of this specification.

**Note:** The reference decoder implements a particular way of sending a modified frame header and sizes that allow the required tiles to be efficiently retrieved. This format may be useful for conformance testing purposes, but these modifications are not specified as part of the normative decoding process and may change in the future.



The aim of this process is to define a small set of requirements on a decoder implementation that will ensure that a compliant decoder will be able to support such use cases.

The input to this process are:

- contents of all syntax elements and variables normally produced when parsing a sequence header OBU,
- contents of all syntax elements and variables normally produced when parsing a frame header OBU,
- contents of all the values normally stored by the reference frame update process specified in [section 7.19](#),
- a bitstream corresponding to the tile data for a single tile,
- a variable `tileWidth` representing the width of the tile in super blocks,
- a variable `tileHeight` representing the height of the tile in super blocks,
- variables `startRowSb` and `startColSb` specifying the top-left location of the tile in units of superblocks,
- A variable `tileBytes` representing the size in bytes of the tile data.

The output of this process is the decoded samples for the tile.

Decoders shall produce output samples that are identical in all respects as those produced by this decoding process.

**Note:** In large scale tile decoding, the decode process is defined for one tile at a time. The expectation is that the reference buffers have been produced using the general decoding process, but this is not a normative requirement and applications can choose to provide the reference buffers in alternative manners as well.

It is a requirement of bitstream conformance that the following conditions are met:

- `use_ref_frame_mvs` is equal to 0
- `disable_frame_end_update_cdf` is equal to 1
- `FrameRestorationType[ plane ]` is equal to `RESTORE_NONE` for plane equal to 0, 1, and 2
- `cdef_bits` is equal to 0
- `delta_lf_present` is equal to 0

**Note:** The decoding process defined here does not invoke the normal post-processing steps of deblock, cdef, superres, loop restoration and reference buffer update. Implementations may choose to implement this process by using the general decode process with these tools disabled.

The process is specified as:

```

init_symbol( tileBytes )
clear_above_context( )
sbSize = use_128x128_superblock ? BLOCK_128X128 : BLOCK_64X64
sbSize4 = Num_4x4_Blocks_Wide[ sbSize ]
MiColStart = startColSb * sbSize4
MiColEnd = Min( MiCols, MiColStart + tileWidth * sbSize4 )
MiRowStart = startRowSb * sbSize4
MiRowEnd = Min( MiRows, MiRowStart + tileHeight * sbSize4 )
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( 0 )
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:

- w is less than or equal to 4096
- h is less than or equal to 4096
- at most 512 tiles are decoded for each frame (the frame may include more than 512 tiles, but at most 512 will be required for each render from a single frame)

Arrays OutY, OutU, OutV (presenting 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  $\text{OutY}[y][x] = \text{CurrFrame}[0][y0 + y][x0 + x]$  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  $\text{OutU}[y][x] = \text{CurrFrame}[1][yC0 + y][xC0 + x]$  with  $x = 0..(w \gg \text{subX}) - 1$  and  $y = 0..(h \gg \text{subY}) - 1$ .

- 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  $\text{OutV}[y][x] = \text{CurrFrame}[2][yC0 + y][xC0 + x]$  with  $x = 0..(w \gg \text{subX}) - 1$  and  $y = 0..(h \gg \text{subY}) - 1$ .

If `mono_chrome` is equal to 0, the output of this process is arrays OutY, OutU, OutV representing the Y, U, and V samples. Otherwise (`mono_chrome` is equal to 1), the output of this process is array OutY.

The bitdepth of each output sample is given by `BitDepth`.

## 7.3. Decode Frame Process

This process is triggered by a call to `decode_frame` from within the syntax tables.

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 7.13](#) is invoked (this process modifies the contents of `CurrFrame`).
2. The CDEF process specified in [section 7.14](#) is invoked (this process takes `CurrFrame` and produces `CdefFrame`).
3. The upscaling process specified in [section 7.15](#) is invoked with `CdefFrame` as input and the output is assigned to `UpscaledCdefFrame`.
4. The upscaling process specified in [section 7.15](#) is invoked with `CurrFrame` as input and the output is assigned to `UpscaledCurrFrame`.
5. The loop restoration process specified in [section 7.16](#) is invoked (this process takes `UpscaledCurrFrame` and `UpscaledCdefFrame` and produces `LrFrame`).
6. The motion field motion vector storage process specified in [section 7.18](#) is invoked.

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 7.20](#) is invoked (this process loads frame state from the reference buffers into the current frame state variables).

The following ordered steps now apply:

1. The reference frame update process as specified in [section 7.19](#) is invoked (this process saves the current frame state into the reference buffers).
2. If `show_frame` is equal to 1 or `show_existing_frame` is equal to 1, the output process as specified in [section 7.17](#) is invoked (this will output the current frame or a saved frame).

**Note:** Although it is specified that all samples in CurrFrame are upscaled, 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.

## 7.4. 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.

The first frame header in a coded video sequence must have `frame_type` equal to `KEY_FRAME` and `show_frame` equal to 1.

If operating points are being used, then only a selection of frames will be decoded. The first frame header that will be decoded for each valid operating point must have `frame_type` equal to `KEY_FRAME` and `show_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 containing a key frame must contain a sequence header OBU before the 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. The values of the parameters must be identical each time the sequence header appears.

**Note:** A bitstream can change the sequence header parameters during decode, but this is interpreted as a video consisting of multiple independent coded video sequences.

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.

OBUs types that are not defined in this specification can be ignored by a decoder.

## 7.5. Random Access Decoding

This specification defines the decoding process when decoding the entirety of a valid bitstream.

Conformant decoders must also be able to decode a valid bitstream from any random access point.

A random access point is defined as being either of the following two cases:

- A frame with `frame_type` equal to `KEY_FRAME` and `show_frame` equal to 1 contained in a temporal unit that also contains a sequence header OBU
- A frame with `show_existing_frame` equal to 1 that refers to a previously decoded frame that had `frame_type` equal to `KEY_FRAME` and was contained in a temporal unit that also contained a sequence header OBU

The result of decoding from a random access point is defined as producing exactly the same output frames (and in the same order) as decoding the whole bitstream, except that any frames output before the random access point are discarded.

**Note:** Implementations may also choose to be able to decode streams robustly even if some frames have been lost, but the techniques for error concealment are not considered normative and are therefore not included in this specification.

## 7.6. CDF Update Process

This process is triggered when the function `update_cdf` is called from the tile group syntax table.

The frame CDF arrays are set equal to the final tile CDF arrays for the largest tile (measured in bytes) 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`.

## 7.7. 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 buffers 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 buffer index.

## 7.8. Motion Field Estimation Process

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 stack of up to three motion vectors (for each reference frame type) is prepared at each location on an 8x8 pixel grid.

The variable w8 (representing the width of the motion field in units of 8x8 pixels) is set equal to MiCols >> 1.

The variable h8 (representing the height of the motion field in units of 8x8 pixels) 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 ( i = 0; i < MFMV_STACK_SIZE; i++ )
        for ( j = 0; j < 2; j++ )
          MotionFieldMvs[ ref ][ y ][ x ][ i ][ j ] = -1 << 15
```

The variable refStamp (representing where in the stack to place projected motion vectors) is set equal to MFMV\_STACK\_SIZE - 1.

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 7.8.1](#) is invoked with src equal to LAST\_FRAME, dir equal to 0, offsetSign equal to 1, and refStamp also passed as an input. (The output of this process is discarded.)

The variable refStamp is set equal to MFMV\_STACK\_SIZE - 2.

**Note:** refStamp is decreased even if the projection process is not invoked.

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 7.8.1](#) is invoked with src equal to BWDREF\_FRAME, dir equal to 0, offsetSign equal to 0, and refStamp also passed as an input, 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  $\text{get\_relative\_dist}(\text{OrderHints}[\text{ALTREF2\_FRAME}], \text{OrderHint}) > 0$ .

If useAlt2 is equal to 1, the following steps apply:

- The projection process in [section 7.8.1](#) is invoked with src equal to ALTREF2\_FRAME, dir equal to 0, offsetSign equal to 0, and refStamp also passed as an input, 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  $\text{get\_relative\_dist}(\text{OrderHints}[\text{ALTREF\_FRAME}], \text{OrderHint}) > 0$ .

If useAlt is equal to 1 and  $(\text{refStamp} \geq 0)$ , the following steps apply:

- The projection process in [section 7.8.1](#) is invoked with src equal to ALTREF\_FRAME, dir equal to 0, offsetSign equal to 0, and refStamp also passed as an input, and the output assigned to projOutput.
- If projOutput is equal to 1, refStamp is set equal to refStamp - 1.

If  $(\text{refStamp} \geq 0)$ , the projection process in [section 7.8.1](#) is invoked with src equal to LAST2\_FRAME, dir equal to 0, offsetSign equal to 1, and refStamp also passed as an input. (The output of this process is discarded.)

## 7.8.1. Projection Process

The inputs to this process are:

- a variable src specifying which reference frame's motion vectors should be projected,
- a variable refStamp specifying which entry in the stack to store the motion vectors,
- a variable dir specifying whether the forward (dir equal to 0) or backward motion vectors in the reference frame should be used,
- a variable offsetSign specifying whether to negate the offset applied to the projected vector.

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  $\text{ref\_frame\_idx}[\text{src} - \text{LAST\_FRAME}]$ .

The variable w8 (representing the width of the motion field in units of 8x8 pixels) is set equal to  $\text{MiCols} \gg 1$ .

The variable h8 (representing the height of the motion field in units of 8x8 pixels) is set equal to  $\text{MiRows} \gg 1$ .

The variable sign is set equal to  $(\text{dir} == 1 \ \&\& \ \text{offsetSign} == 0) ? -1 : 1$ .

The variable `dstSign` is set equal to  $(dir == 0 \ \&\& \ offsetSign == 0) ? 1 : -1$ .

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 ][ dir ]
        if ( srcRef > INTRA_FRAME ) {
            posValid = Abs( get_relative_dist(SavedOrderHints[ srcIdx ][ srcRef ], OrderHints[ src ])
) <= MAX_FRAME_DISTANCE &&
                Abs( get_relative_dist(SavedOrderHints[ srcIdx ][ srcRef ], OrderHint) ) <=
MAX_FRAME_DISTANCE
            if ( posValid ) {
                mv = SavedMvs[ srcIdx ][ row ][ col ][ dir ]
                projMv = get_mv_projection( mv, srcRef, src, src, sign, dstSign )
                posValid = get_block_position( y8, x8, offsetSign, projMv )
                if ( ProjDenominator < 0 )
                    posValid = 0
                if ( posValid ) {
                    for ( dst = LAST_FRAME; dst <= ALTREF_FRAME; dst++ ) {
                        projMv = get_mv_projection( mv, srcRef, src, dst, sign, -sign )
                        MotionFieldMvs[ dst ][ PosY8 ][ PosX8 ][ refStamp ] = projMv
                    }
                }
            }
        }
    }
}

```

When the function `get_mv_projection` is called, the get mv projection process specified in [section 7.8.2](#) is invoked and the output assigned to `projMv`.

When the function `get_block_position` is called, the get block position process specified in [section 7.8.3](#) 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.

## 7.8.2. Get MV Projection Process

The inputs to this process are:

- a length 2 array `mv` specifying a motion vector,

- a variable `ref` specifying the reference frame associated with the motion vector,
- a variable `src` specifying which reference frame's motion vectors should be projected,
- a variable `dst` specifying which reference frame to target when adjusting the motion vector.
- a variable `sign` specifying a negation multiplier for the source motion vector direction,
- a variable `dstSign` specifying a negation multiplier for the motion vector direction before projection.

The outputs of this process are:

- a length 2 array `projMv` containing a projected motion vector

This process starts with a motion vector `mv` from a previous frame (`src`). This motion vector gives the displacement expected when moving a certain number of frames (between `src` and `ref`). In order to use the motion vector for predictions using a different reference frame (`dst`), the length of the motion vector must be scaled.

The variable `srcIdx` (representing the reference frame containing the original motion vector) is set equal to `ref_frame_idx[ src - LAST_FRAME ]`.

The variable `srcOrderHint` (representing the expected output order of the reference frame containing the original motion vector) is set equal to `OrderHints[ src ]`.

The variable `refOrderHint` (representing the expected output order of the frame referenced by the motion vector) is set equal to `SavedOrderHints[ srcIdx ][ ref ]`.

The variable `ProjDenominator` (representing the number of frames covered by the motion vector) is set equal to `sign * get_relative_dist(srcOrderHint, refOrderHint)`.

The variable `clippedDenominator` is set equal to `Clip3( 1, MAX_FRAME_DISTANCE, ProjDenominator )`.

The variable `dstOrderHint` (representing the expected output order of the reference frame that is being targeted) is set equal to `OrderHints[ dst ]`.

The variable `dstOffset` (representing the number of frames that the motion vector should cover after it is scaled) is set equal to `Clip3( -MAX_FRAME_DISTANCE, MAX_FRAME_DISTANCE, -dstSign * get_relative_dist(OrderHint, dstOrderHint) )`.

The projected motion vector is specified as follows:

```
for ( i = 0; i < 2; i++ )
    projMv[ i ] = Round2Signed( mv[ i ] * dstOffset * Div_Mult[ clippedDenominator ], 14 )
```

where `Div_Mult` is a constant lookup table specified as:

```

Div_Mult[64] = {
    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, 512,
    496, 481, 468, 455, 442, 431, 420, 409, 399, 390, 381,
    372, 364, 356, 348, 341, 334, 327, 321, 315, 309, 303,
    297, 292, 287, 282, 277, 273, 268, 264, 260,
}

```

### 7.8.3. Get Block Position Process

The inputs to this process are:

- variables x8 and y8 specifying a location in units of 8x8 pixels,
- a variable offsetSign specifying the direction to offset the projection,
- 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 pixels.

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 ], offsetSign, MiRows >> 1, MAX_OFFSET_HEIGHT)`.

The variable PosX8 is set equal to `project(x8, projMv[ 1 ], offsetSign, MiCols >> 1, MAX_OFFSET_WIDTH)`.

where the function project is specified as follows:

```

project( v8, delta, offsetSign, 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 += ( offsetSign ? -1 : 1 ) * 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.

## 7.9. Motion Vector Prediction Processes

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 5.9.22](#). This function call invokes the Find MV Stack Process specified in [section 7.9.1](#).

### 7.9.1. 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 pixels is set equal to Num\_4x4\_Blocks\_Wide[ MiSize ].

The variable bh4 specifying the height of the block in 4x4 pixels 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 NewMvFound (representing a candidate is found that used NEWMV encoding) is set equal to 0.
3. The setup zero mv process specified in [section 7.9.1.1](#) is invoked with the input 0 and the output is assigned to ZeroMvs[ 0 ].
4. If isCompound is equal to 1, the setup zero mv process specified in [section 7.9.1.1](#) is invoked with the input 1 and the output is assigned to ZeroMvs[ 1 ].
5. The variable FoundMatch is set equal to 0.
6. The scan row process in [section 7.9.1.2](#) 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 7.9.1.3](#) 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  $\text{Max}(bw4, bh4)$  is less than or equal to 16, the scan point process in [section 7.9.1.4](#) 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 neighbourhood) is set equal to foundAboveMatch + foundLeftMatch.
13. The variable numNearest (representing the number of motion vectors found in the immediate neighbourhood) is set equal to NumMvFound
14. The variable numNew (representing if a NEWMV candidate was found in the immediate neighbourhood) is set equal to NewMvFound
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 7.9.1.5](#) is invoked with isCompound as input (the temporal scan process affects ZeroMvContext).
18. The scan point process in [section 7.9.1.4](#) 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 7.9.1.2](#) 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 7.9.1.3](#) 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 7.9.1.2](#) 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 7.9.1.3](#) 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 7.9.1.11](#) is invoked with start equal to 0 and end equal to numNearest.
34. The sorting process in [section 7.9.1.11](#) is invoked with start equal to numNearest and end equal to NumMvFound.
35. If NumMvFound is less than 2, the extra search process in [section 7.9.1.12](#) is invoked with isCompound as input.
36. The context and clamping process in [section 7.9.1.14](#) is invoked with isCompound and numNew as input.

### 7.9.1.1. Setup Zero 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 ].

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 pixels) is set equal to Block\_Width[ MiSize ].

The variable bh (representing the height of the block in units of pixels) is set equal to Block\_Height[ MiSize ].

The output motion vector mv is specified by projecting the central pixel 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(xc, WARPEDMODEL_PREC_BITS - 3)
        mv[1] = Round2Signed(yc, WARPEDMODEL_PREC_BITS - 3)
    } else {
        mv[0] = Round2Signed(xc, WARPEDMODEL_PREC_BITS - 2) * 2
        mv[1] = Round2Signed(yc, WARPEDMODEL_PREC_BITS - 2) * 2
    }
}
lower_mv_precision( mv )

```

where the call to `lower_mv_precision` invokes the lower precision process specified in [section 7.9.1.10](#).

### 7.9.1.2. Scan Row Process

The inputs to this process are:

- a variable `deltaRow` specifying (in units of 4x4 pixels) 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 pixels is set equal to `Num_4x4_Blocks_Wide[ MiSize ]`.

The variable `end4` specifying the last block to be scanned in horizontal 4x4 pixels 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  $(bw4 \geq 16)$ .

**Note:** `useStep16` is equal to 1 when the block is 64 pixels wide or wider. This means only 4 locations will be searched in this case. However, a 32 pixel wide block may still search 8 locations.

If  $\text{Abs}(\text{deltaRow})$  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)
    add_ref_mv_candidate( mvRow, mvCol, isCompound, len)
    i += len
}
```

where the call to `add_ref_mv_candidate` invokes the process in [section 7.9.1.7](#).

### 7.9.1.3. Scan Col Process

The inputs to this process are:

- a variable `deltaCol` specifying (in units of 4x4 pixels) 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 pixels is set equal to `Num_4x4_Blocks_High[ MiSize ]`.

The variable end4 specifying the last block to be scanned in vertical 4x4 pixels is set equal to  $\text{Min}(\text{Min}(\text{bh4}, \text{MiRows} - \text{MiRow}), 16)$ .

The variable deltaRow is set equal to 0.

The variable useStep16 is set equal to  $(\text{bh4} \geq 16)$ .

If  $\text{Abs}(\text{deltaCol})$  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)
    add_ref_mv_candidate( mvRow, mvCol, isCompound, len)
    i += len
}
```

where the call to `add_ref_mv_candidate` invokes the process in [section 7.9.1.7](#).

#### 7.9.1.4. Scan Point Process

The inputs to this process are:

- a variable deltaRow specifying (in units of 4x4 pixels) how far above to look for a motion vector,
- a variable deltaCol specifying (in units of 4x4 pixels) 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  $\text{MiRow} + \text{deltaRow}$ .

The variable mvCol is set equal to  $\text{MiCol} + \text{deltaCol}$ .

The variable len is set equal to 8.

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 7.9.1.7](#) is invoked with `mvRow`, `mvCol`, `isCompound`, `len` as inputs.

### 7.9.1.5. 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 pixels is set equal to `Num_4x4_Blocks_Wide[ MiSize ]`.

The variable `bh4` specifying the height of the block in 4x4 pixels 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 7.9.1.6](#).

If `bh4` is less than 2 or `bw4` is less than 2, the process exits at this point.

Otherwise, the process then scans positions around the block (but still within the same superblock) as follows:

```
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 pixels) 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 )  
}
```

### 7.9.1.6. Temporal Sample Process

The inputs to this process are:

- variables `deltaRow` and `deltaCol` specifying (in units of 4x4 pixels) 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  $\text{mvRow} \gg 1$ .

(`x8` and `y8` represent the position of the candidate in units of 8x8 pixels.)

The process is specified as follows:

```

foundValidMv = 0
if ( !isCompound ) {
    for ( i = 0; i < MFMV_STACK_SIZE; i++ ) {
        candMv = MotionFieldMvs[ RefFrame[ 0 ] ][ y8 ][ x8 ][ i ]
        if ( candMv[ 0 ] == -1 << 15 )
            continue
        foundValidMv = 1
        lower_mv_precision( candMv )
        if ( deltaRow == 0 && deltaCol == 0 ) {
            if ( Abs( candMv[ 0 ] - ZeroMvs[ 0 ][ 0 ] ) >= 16 ||
                Abs( candMv[ 1 ] - ZeroMvs[ 0 ][ 1 ] ) >= 16 )
                ZeroMvContext = 1
        }
        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
        }
    }
    return
}
} else {
    for ( i = 0; i < MFMV_STACK_SIZE; i++ ) {
        candMv0 = MotionFieldMvs[ RefFrame[ 0 ] ][ y8 ][ x8 ][ i ]
        if ( candMv0[ 0 ] == -1 << 15 )
            continue
        candMv1 = MotionFieldMvs[ RefFrame[ 1 ] ][ y8 ][ x8 ][ i ]
        if ( candMv1[ 0 ] == -1 << 15 )
            continue
        foundValidMv = 1
        lower_mv_precision( candMv0 )
        lower_mv_precision( candMv1 )
        if ( deltaRow == 0 && deltaCol == 0 ) {
            if ( Abs( candMv0[ 0 ] - ZeroMvs[ 0 ][ 0 ] ) >= 16 ||
                Abs( candMv0[ 1 ] - ZeroMvs[ 0 ][ 1 ] ) >= 16 ||
                Abs( candMv1[ 0 ] - ZeroMvs[ 1 ][ 0 ] ) >= 16 ||
                Abs( candMv1[ 1 ] - ZeroMvs[ 1 ][ 1 ] ) >= 16 )
                ZeroMvContext = 1
        }
        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
    }
    return
}
}
if ( deltaRow == 0 && deltaCol == 0 ) {
    if ( !foundValidMv )
        ZeroMvContext = 1
}

```

where the call to `lower_mv_precision` invokes the lower precision process specified in [section 7.9.1.10](#).

### 7.9.1.7. Add Reference Motion Vector Process

The inputs to this process are:

- variables `mvRow` and `mvCol` specifying (in units of 4x4 pixels) the candidate location,
- a variable `isCompound` containing 0 for single prediction, or 1 to signal compound prediction,
- a variable `len` 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 7.9.1.8](#) is invoked with `mvRow`, `mvCol`, `len`, 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 7.9.1.9](#) is invoked with `mvRow`, `mvCol`, and `len` as inputs.

### 7.9.1.8. Search Stack Process

The inputs to this process are:



- variables mvRow and mvCol specifying (in units of 4x4 pixels) the candidate location,
- a variable candList specifying which list in the candidate matches our reference frame,
- a variable len specifying the weight attached to this motion vector.

This process searches the stack for an exact match with a candidate motion vector. If present, the weight is increased, 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 ( Min( Block\_Width[ candSize ], Block\_Height[ candSize ] ) >= 8 ).

The candidate motion vector candMv is set as follows:

- If ( candMode == GLOBALMV || candMode == GLOBAL\_GLOBALMV ) and ( GmType[ RefFrame[ 0 ] ] > TRANSLATION ) and ( large == 1 ), candMv is set equal to ZeroMvs[ 0 ].
- Otherwise, candMv is set equal to Mvs[ mvRow ][ mvCol ][ candList ].

The lower precision process specified in [section 7.9.1.10](#) is invoked with candMv.

The variable weight is set equal to len \* 2.

If candMode is equal to NEWMV, NewMvFound is set equal to 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.

### 7.9.1.9. Compound Search Stack Process

The inputs to this process are:

- variables mvRow and mvCol specifying (in units of 4x4 pixels) the candidate location,

- a variable len specifying the weight attached to this motion vector.

This process searches the stack for an exact match with a candidate pair of motion vectors. If present, the weight is increased, 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 ].

The variable large is set equal to ( Min( Block\_Width[ candSize ], Block\_Height[ candSize ] ) >= 8 ).

If candMode is equal to GLOBAL\_GLOBALMV and large is equal to 1, for refList = 0..1 the following applies:

- If GmType[ RefFrame[ refList ] ] > TRANSLATION, candMvs[ refList ] is set equal to ZeroMvs[ refList ].

For i = 0..1, the lower precision process specified in [section 7.9.1.10](#) is invoked with candMvs[ i ].

The variable weight is set equal to len \* 2.

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 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( YModes[ mvRow ][ mvCol ] ) is equal to 1, NewMvFound is set equal to 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.

### 7.9.1.10. 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 ]++
    }
}

```

### 7.9.1.11. 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.

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 < 2; 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
        }
    }
}
```

### 7.9.1.12. 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 neighbours 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++ ) {
    for ( idx = 0; idx < num4x4 && NumMvFound < 2; idx++ ) {
        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 7.9.1.13](#) 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 ]; idx++ ) {
        combinedMvs[ compCount ][ list ] = RefDiffMvs[ list ][ idx ]
        compCount++
    }
    while ( compCount < 3 ) {
        combinedMvs[ compCount ][ list ] = ZeroMvs[ 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 ] = ZeroMvs[ 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.

### 7.9.1.13. Add Extra Mv Candidate Process

The inputs to this process are:

- variables mvRow and mvCol specifying (in units of 4x4 pixels) the candidate location,
- a variable isCompound containing 0 for single prediction, or 1 to signal compound prediction.

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 ]
    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 ( candRef > INTRA_FRAME && 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++
    }
  }
}

```

### 7.9.1.14. 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 neighbourhood.

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 pixels) is set equal to `Block_Width[ MiSize ]`.

The variable `bh` (representing the height of the block in units of pixels) is set equal to `Block_Height[ MiSize ]`.

The variable `numLists` specifying the number of reference frames used for this block is set equal to  $(\text{isCompound} ? 2 : 1)$ .

The array `Dr1CtxStack` 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 {
            if (w1 < REF_CAT_LEVEL) {
                z = 2
            }
        }
    }
    Dr1CtxStack[ 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
}

```

## 7.9.2. Has Overlappable Candidates Process

This process is triggered by a call to has\_overlappable\_candidates.

It returns 1 to indicate that the block has neighbours 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
}

```

## 7.9.3. Find Warp Samples Process

This process is triggered when the find\_warp\_samples function is invoked.

The process examines the neighbouring 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.

The variable w4 specifying the width of the block in 4x4 pixels is set equal to Num\_4x4\_Blocks\_Wide[ MiSize ].

The variable h4 specifying the height of the block in 4x4 pixels is set equal to Num\_4x4\_Blocks\_High[ MiSize ].

The following ordered steps apply:

1. The add sample process in [section 7.9.3.1](#) is invoked with deltaRow equal to -1, and deltaCol equal to 0..(w4 - 1) (this attempts to add candidates above the current block).
2. The add sample process in [section 7.9.3.1](#) is invoked with deltaCol equal to -1, and deltaRow equal to 0..(h4 - 1) (this attempts to add candidates to the left of the current block).
3. The add sample process in [section 7.9.3.1](#) is invoked with deltaRow equal to -1, and deltaCol equal to -1 (this attempts to add a candidate to the top left).
4. The add sample process in [section 7.9.3.1](#) is invoked with deltaRow equal to -1, and deltaCol equal to w4 (this attempts to add a candidate to the top right).
5. If NumSamples is equal to 0 and NumSamplesScanned is greater than 0, NumSamples is set equal to 1.

### 7.9.3.1. Add Sample Process

The inputs to this process are:

- a variable deltaRow specifying (in units of 4x4 pixels) how far above to look for a motion vector,
- a variable deltaCol specifying (in units of 4x4 pixels) 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  $\text{RefFrames}[\text{mvRow}][\text{mvCol}][1]$  is not equal to NONE, then this process immediately returns.

The variable  $\text{candSz}$  is set equal to  $\text{MiSizes}[\text{mvRow}][\text{mvCol}]$ .

The variable  $\text{candW4}$  is set equal to  $\text{Num\_4x4\_Blocks\_Wide}[\text{candSz}]$ .

The variable  $\text{candH4}$  is set equal to  $\text{Num\_4x4\_Blocks\_High}[\text{candSz}]$ .

The variable  $\text{candRow}$  is set equal to  $\text{mvRow} \& \sim(\text{candH4} - 1)$ .

The variable  $\text{candCol}$  is set equal to  $\text{mvCol} \& \sim(\text{candW4} - 1)$ .

The variable  $\text{midY}$  is set equal to  $\text{candRow} * 4 + \text{candH4} * 2 - 1$ .

The variable  $\text{midX}$  is set equal to  $\text{candCol} * 4 + \text{candW4} * 2 - 1$ .

The variable  $\text{threshold}$  is set equal to  $\text{Clip3}(16, 112, \text{Max}(\text{Block\_Width}[\text{MiSize}], \text{Block\_Height}[\text{MiSize}]))$ .

The variable  $\text{mvDiffRow}$  is set equal to  $\text{Abs}(\text{Mvs}[\text{candRow}][\text{candCol}][0][0] - \text{Mv}[0][0])$ .

The variable  $\text{mvDiffCol}$  is set equal to  $\text{Abs}(\text{Mvs}[\text{candRow}][\text{candCol}][0][1] - \text{Mv}[0][1])$ .

The variable  $\text{valid}$  is set equal to  $((\text{mvDiffRow} + \text{mvDiffCol}) \leq \text{threshold})$ .

**Note:**  $\text{candRow}$  and  $\text{candCol}$  give the top-left position of the candidate block in units of 4x4 blocks.  $\text{midX}$  and  $\text{midY}$  give the central position of the candidate block in units of pixels.

A candidate array (representing source and destination locations in units of 1/8 pixels) is specified as:

```
cand[ 0 ] = midY * 8
cand[ 1 ] = midX * 8
cand[ 2 ] = midY * 8 + Mvs[ candRow ][ candCol ][ 0 ][ 0 ]
cand[ 3 ] = midY * 8 + Mvs[ candRow ][ candCol ][ 0 ][ 1 ]
```

If this array  $\text{cand}$  is equal (in all 4 locations) to  $\text{CandList}[i]$  for any  $i < \text{NumSamples}$ , then this process immediately returns.

Otherwise, the following ordered steps apply:

1.  $\text{NumSamplesScanned}$  is increased by 1.
2. If  $\text{valid}$  is equal to 0 and  $\text{NumSamplesScanned}$  is greater than 1, the process exits.
3.  $\text{CandList}[\text{NumSamples}][j]$  is set equal to  $\text{cand}[j]$  for  $j=0..3$ .
4. If  $\text{valid}$  is equal to 1,  $\text{NumSamples}$  is increased by 1.

**Note:** The test for `cand` being equal to `CandList[i]` ensures that each candidate block is only added once to the list. If the first two entries are equal, all four entries will automatically be equal. The reference implementation shows how to use additional logic to avoid calling the add sample process more than once per candidate.

## 7.10. Prediction Processes

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 5.9.32](#).

### 7.10.1. Intra Prediction Process

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  $1 \ll \log_2 W$ .

The variable  $h$  is set equal to  $1 \ll \log_2 H$ .

The variable  $\text{maxX}$  is set equal to  $(\text{MiCols} * \text{MI\_SIZE}) - 1$ .

The variable  $\text{maxY}$  is set equal to  $(\text{MiRows} * \text{MI\_SIZE}) - 1$ .

If  $\text{plane}$  is greater than 0, then:

- $\text{maxX}$  is set equal to  $((\text{MiCols} * \text{MI\_SIZE}) \gg \text{subsampling\_x}) - 1$ .
- $\text{maxY}$  is set equal to  $((\text{MiRows} * \text{MI\_SIZE}) \gg \text{subsampling\_y}) - 1$ .

The array  $\text{AboveRow}[i]$  for  $i = 0..w + h - 1$  is derived as follows:

- If  $\text{haveAbove}$  is equal to 0 and  $\text{haveLeft}$  is equal to 1,  $\text{AboveRow}[i]$  is set equal to  $\text{CurrFrame}[\text{plane}][y][x - 1]$ .
- Otherwise, if  $\text{haveAbove}$  is equal to 0 and  $\text{haveLeft}$  is equal to 0,  $\text{AboveRow}[i]$  is set equal to  $(1 \ll (\text{BitDepth} - 1)) - 1$ .
- Otherwise, the following applies:
  - The variable  $\text{aboveLimit}$  is set equal to  $\text{Min}(\text{maxX}, x + (\text{haveAboveRight} ? 2 * w : w) - 1)$ .
  - $\text{AboveRow}[i]$  is set equal to  $\text{CurrFrame}[\text{plane}][y - 1][\text{Min}(\text{aboveLimit}, x + i)]$ .

The array  $\text{LeftCol}[i]$  for  $i = 0..w + h - 1$  is derived as follows:

- If  $\text{haveLeft}$  is equal to 0 and  $\text{haveAbove}$  is equal to 1,  $\text{LeftCol}[i]$  is set equal to  $\text{CurrFrame}[\text{plane}][y - 1][x]$ .
- Otherwise, if  $\text{haveLeft}$  is equal to 0 and  $\text{haveAbove}$  is equal to 0,  $\text{LeftCol}[i]$  is set equal to  $(1 \ll (\text{BitDepth} - 1)) + 1$ .
- Otherwise, the following applies:
  - The variable  $\text{leftLimit}$  is set equal to  $\text{Min}(\text{maxY}, y + (\text{haveBelowLeft} ? 2 * h : h) - 1)$ .
  - $\text{LeftCol}[i]$  is set equal to  $\text{CurrFrame}[\text{plane}][\text{Min}(\text{leftLimit}, y + i)][x - 1]$ .

The array  $\text{AboveRow}[i]$  for  $i = -1$  is specified by:

- If  $\text{haveAbove}$  is equal to 1 and  $\text{haveLeft}$  is equal to 1,  $\text{AboveRow}[-1]$  is set equal to  $\text{CurrFrame}[\text{plane}][y - 1][x - 1]$ .
- Otherwise if  $\text{haveAbove}$  is equal to 1,  $\text{AboveRow}[-1]$  is set equal to  $\text{CurrFrame}[\text{plane}][y - 1][x]$ .
- Otherwise if  $\text{haveLeft}$  is equal to 1,  $\text{AboveRow}[-1]$  is set equal to  $\text{CurrFrame}[\text{plane}][y][x - 1]$ .

- 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 7.10.1.2](#) 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 7.10.1.3](#) 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 7.10.1.5](#) 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 7.10.1.4](#) is invoked with haveLeft, haveAbove, log2W, log2H, w, and h as inputs and the output is assigned to pred.
- Otherwise, the basic intra prediction process specified in [section 7.10.1.1](#) 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  $j = 0..w-1$ .

### 7.10.1.1. Basic Intra Prediction Process

The inputs to this process are:

- a variable 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.

The output of this process is a 2D array named pred containing the intra predicted samples.

The process uses simple filters to generate filtered samples from the samples in LeftCol and AboveRow as follows:

- If mode is equal to V\_PRED, pred[ i ][ j ] is set equal to AboveRow[ j ] with  $j = 0..w-1$  and  $i = 0..h-1$  (each row of the block is filled with a copy of AboveRow).
- Otherwise if mode is equal to H\_PRED, pred[ i ][ j ] is set equal to LeftCol[ i ] with  $j = 0..w-1$  and  $i = 0..h-1$  (each column of the block is filled with a copy of LeftCol).
- Otherwise if mode is equal to D203\_PRED, the following applies for  $i = 0..h-1$ :

1.  $\text{pred}[i][j] = \text{Round2}(\text{LeftCol}[i + j/2] + \text{LeftCol}[i + 1 + j/2], 1)$  for even values of  $j$  in the range  $0..w-2$
  2.  $\text{pred}[i][j] = \text{Round2}(\text{LeftCol}[i + j/2] + 2 * \text{LeftCol}[i + 1 + j/2] + \text{LeftCol}[i + 2 + j/2], 2)$  for odd values of  $j$  in the range  $0..w-1$
- Otherwise if mode is equal to D45\_PRED, the following applies for  $i = 0..h-1$ , for  $j = 0..w-1$ :
    1.  $\text{pred}[i][j] = \text{Round2}(\text{AboveRow}[i + j] + 2 * \text{AboveRow}[i + j + 1] + \text{AboveRow}[i + j + 1], 2)$  if  $(i + j + 2 \geq w + h)$
    2.  $\text{pred}[i][j] = \text{Round2}(\text{AboveRow}[i + j] + 2 * \text{AboveRow}[i + j + 1] + \text{AboveRow}[i + j + 2], 2)$  if  $(i + j + 2 < w + h)$
  - Otherwise if mode is equal to D67\_PRED, the following applies for  $i = 0..h-1$ , for  $j = 0..w-1$ :
    1.  $\text{pred}[i][j] = \text{Round2}(\text{AboveRow}[i/2 + j] + \text{AboveRow}[i/2 + j + 1], 1)$  for even values of  $i$  in the range  $0..h-2$
    2.  $\text{pred}[i][j] = \text{Round2}(\text{AboveRow}[i/2 + j] + 2 * \text{AboveRow}[i/2 + j + 1] + \text{AboveRow}[i/2 + j + 2], 2)$  for odd values of  $i$  in the range  $0..h-1$ .
  - Otherwise if mode is equal to D113\_PRED, the following applies:
    1.  $\text{pred}[0][j] = \text{Round2}(\text{AboveRow}[j - 1] + \text{AboveRow}[j], 1)$  for  $j = 0..w-1$
    2.  $\text{pred}[1][0] = \text{Round2}(\text{LeftCol}[0] + 2 * \text{AboveRow}[-1] + \text{AboveRow}[0], 2)$
    3.  $\text{pred}[1][j] = \text{Round2}(\text{AboveRow}[j - 2] + 2 * \text{AboveRow}[j - 1] + \text{AboveRow}[j], 2)$  for  $j = 1..w-1$
    4.  $\text{pred}[2][0] = \text{Round2}(\text{AboveRow}[-1] + 2 * \text{LeftCol}[0] + \text{LeftCol}[1], 2)$
    5.  $\text{pred}[i][0] = \text{Round2}(\text{LeftCol}[i - 3] + 2 * \text{LeftCol}[i - 2] + \text{LeftCol}[i - 1], 2)$  for  $i = 3..h-1$
    6.  $\text{pred}[i][j] = \text{pred}[i - 2][j - 1]$  for  $i = 2..h-1$ , for  $j = 1..w-1$
  - Otherwise if mode is equal to D135\_PRED, the following applies:
    1.  $\text{pred}[0][0] = \text{Round2}(\text{LeftCol}[0] + 2 * \text{AboveRow}[-1] + \text{AboveRow}[0], 2)$
    2.  $\text{pred}[0][j] = \text{Round2}(\text{AboveRow}[j - 2] + 2 * \text{AboveRow}[j - 1] + \text{AboveRow}[j], 2)$  for  $j = 1..w-1$
    3.  $\text{pred}[1][0] = \text{Round2}(\text{AboveRow}[-1] + 2 * \text{LeftCol}[0] + \text{LeftCol}[1], 2)$
    4.  $\text{pred}[i][0] = \text{Round2}(\text{LeftCol}[i - 2] + 2 * \text{LeftCol}[i - 1] + \text{LeftCol}[i], 2)$  for  $i = 2..h-1$
    5.  $\text{pred}[i][j] = \text{pred}[i - 1][j - 1]$  for  $i = 1..h-1$ , for  $j = 1..w-1$
  - Otherwise if mode is equal to D157\_PRED, the following applies:
    1.  $\text{pred}[0][0] = \text{Round2}(\text{LeftCol}[0] + \text{AboveRow}[-1], 1)$

2.  $\text{pred}[i][0] = \text{Round2}(\text{LeftCol}[i-1] + \text{LeftCol}[i], 1)$  for  $i = 1..h-1$
  3.  $\text{pred}[0][1] = \text{Round2}(\text{LeftCol}[0] + 2 * \text{AboveRow}[-1] + \text{AboveRow}[0], 2)$
  4.  $\text{pred}[1][1] = \text{Round2}(\text{AboveRow}[-1] + 2 * \text{LeftCol}[0] + \text{LeftCol}[1], 2)$
  5.  $\text{pred}[i][1] = \text{Round2}(\text{LeftCol}[i-2] + 2 * \text{LeftCol}[i-1] + \text{LeftCol}[i], 2)$  for  $i = 2..h-1$
  6.  $\text{pred}[0][j] = \text{Round2}(\text{AboveRow}[j-3] + 2 * \text{AboveRow}[j-2] + \text{AboveRow}[j-1], 2)$  for  $j = 2..w-1$
  7.  $\text{pred}[i][j] = \text{pred}[i-1][j-2]$  for  $i = 1..h-1$ , for  $j = 2..w-1$
- Otherwise (mode is equal to TM\_PRED), the following ordered steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
    1. The variable base is set equal to  $\text{AboveRow}[j] + \text{LeftCol}[i] - \text{AboveRow}[-1]$ .
    2. The variable pLeft is set equal to  $\text{Abs}(\text{base} - \text{LeftCol}[i])$ .
    3. The variable pTop is set equal to  $\text{Abs}(\text{base} - \text{AboveRow}[j])$ .
    4. The variable pTopLeft is set equal to  $\text{Abs}(\text{base} - \text{AboveRow}[-1])$ .
    5. If  $p\text{Left} \leq p\text{Top}$  and  $p\text{Left} \leq p\text{TopLeft}$ ,  $\text{pred}[i][j]$  is set equal to  $\text{LeftCol}[i]$ .
    6. Otherwise, if  $p\text{Top} \leq p\text{TopLeft}$ ,  $\text{pred}[i][j]$  is set equal to  $\text{AboveRow}[j]$ .
    7. Otherwise,  $\text{pred}[i][j]$  is set equal to  $\text{AboveRow}[-1]$ .

The output of the process is the array pred.

### 7.10.1.2. 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 4x2 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  $\text{AboveRow}[(j4 \ll 2) + i - 1]$ .
  - Otherwise ( $i2$  is not equal to 0),  $p[i]$  is set equal to  $\text{pred}[(i2 \ll 1) - 1][(j4 \ll 2) + i - 1]$ .
- Otherwise ( $i$  is greater than or equal to 5),  $p[i]$  is derived as follows:
  - If  $j4$  is equal to 0,  $p[i]$  is set equal to  $\text{LeftCol}[(i2 \ll 1) + i - 5]$ .
  - Otherwise ( $j4$  is not equal to 0),  $p[i]$  is set equal to  $\text{pred}[(i2 \ll 1) + i - 5][(j4 \ll 2) - 1]$ .
- The following steps apply for  $i1 = 0..1$ ,  $j1 = 0..3$ :
  - The variable  $pr$  is set equal to 0.
  - The variable  $pr$  is incremented by  $\text{Intra\_Filter\_Taps}[\text{filter\_intra\_mode}][(i1 \ll 2) + j1][i] * p[i]$  for  $i = 0..6$ .
  - $\text{pred}[(i2 \ll 1) + i1][(j4 \ll 2) + j1]$  is set equal to  $\text{Clip1}(\text{Round2Signed}(pr, \text{INTRA\_FILTER\_SCALE\_BITS}))$ .

The output of the process is the array  $\text{pred}$ .

### 7.10.1.3. 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 7.10.1.6](#) is invoked and the output assigned to both LeftCol[ -1 ] and AboveRow[ -1 ].
    - The intra filter type process specified in [section 7.10.1.7](#) 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 7.10.1.8](#) 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 7.10.1.11](#) 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 7.10.1.8](#) 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 7.10.1.11](#) is invoked with the parameters numPx, strength, and 1 as inputs.
  - The intra edge upsample selection process specified in [section 7.10.1.9](#) 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  $(w + (pAngle < 90 ? h : 0))$ .
- If upsampleAbove is equal to 1, the intra edge upsample process specified in [section 7.10.1.10](#) is invoked with the parameters numPx and 0 as inputs.
- The intra edge upsample selection process specified in [section 7.10.1.9](#) 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 7.10.1.10](#) is invoked with the parameters numPx and 1 as inputs.

5. The variable dx is derived as follows:

- If pAngle is greater than 0 and 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 and less than 270, dy is set equal to Dr\_Intra\_Derivative[ 270 - pAngle ].
- Otherwise, dy is undefined.

7. If pAngle is greater than 0 and 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  $(idx >> (6 - upsampleAbove)) + (j << upsampleAbove)$ .
- The variable shift is set equal to  $((idx << upsampleAbove) >> 1) \& 0x1F$ .
- The variable maxBaseX is set equal to  $(w + h - 1) << upsampleAbove$ .
- 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  $idx \gg (6 - upsampleAbove)$ .
  - If  $base$  is greater than or equal to  $-(1 \ll upsampleAbove)$ , the following steps apply:
    - The variable  $shift$  is set equal to  $((idx \ll upsampleAbove) \gg 1) \& 0x1F$ .
    - $pred[i][j]$  is set equal to  $Clip1( Round2( AboveRow[base] * (32 - shift) + AboveRow[base + 1] * shift, 5 ))$ .
  - Otherwise ( $base$  is less than  $-(1 \ll upsampleAbove)$ ), the following steps apply:
    - The variable  $idx$  is set equal to  $(i \ll 6) - (j + 1) * dy$ .
    - The variable  $base$  is set equal to  $idx \gg (6 - upsampleLeft)$ .
    - The variable  $shift$  is set equal to  $((idx \ll upsampleLeft) \gg 1) \& 0x1F$ .
    - $pred[i][j]$  is set equal to  $Clip1( Round2( LeftCol[base] * (32 - shift) + LeftCol[base + 1] * shift, 5 ))$ .
9. Otherwise, if  $pAngle$  is greater than 180 and  $pAngle$  is less than 270, the following steps apply for  $i = 0..h-1$ , for  $j = 0..w-1$ :
- The variable  $idx$  is set equal to  $(j + 1) * dy$ .
  - The variable  $base$  is set equal to  $(idx \gg (6 - upsampleLeft)) + (i \ll upsampleLeft)$ .
  - The variable  $shift$  is set equal to  $((idx \ll upsampleLeft) \gg 1) \& 0x1F$ .
  - The variable  $maxBaseY$  is set equal to  $(w + h - 1) \ll upsampleLeft$ .
  - If  $base$  is less than  $maxBaseY$ ,  $pred[i][j]$  is set equal to  $Clip1( Round2( LeftCol[base] * (32 - shift) + LeftCol[base + 1] * shift, 5 ))$ .
  - Otherwise ( $base$  is greater than or equal to  $maxBaseY$ ),  $pred[i][j]$  is set equal to  $LeftCol[maxBaseY]$ .
10. Otherwise, if  $pAngle$  is equal to 90,  $pred[i][j]$  is set equal to  $AboveRow[j]$  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,  $pred[i][j]$  is set equal to  $LeftCol[i]$  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  $pred$ .

#### 7.10.1.4. 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),  $\text{pred}[i][j]$  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`.

### 7.10.1.5. 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:

log2H	smWeightsY
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

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. pred[ i ][ j ] is set equal to Round2( 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 smWeights is set dependent on the value of log2H according to the following table:

log2H	smWeights
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 smoothPred is set as follows:

```
smoothPred = smWeights[ i ] * AboveRow[ j ] +
              ( 256 - smWeights[ i ] ) * LeftCol[ h - 1 ]
```

3. pred[ i ][ j ] is set equal to Round2( smoothPred, 8 ).

- Otherwise if 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 smWeights is set dependent on the value of log2W according to the following table:

log2W	smWeights
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 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.

### 7.10.1.6. 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).

### 7.10.1.7. 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)
}

```

### 7.10.1.8. 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) {
        if (d >= 1) strength = 1
        if (d >= 4) strength = 2
        if (d >= 32) strength = 3
    } else {
        if (d >= 1) 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 {
        if (d >= 1) strength = 3
    }
}

```

### 7.10.1.9. 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)
}
```

#### 7.10.1.10. 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]
}

```

### 7.10.1.11. Intra Edge Filter Process

The inputs to this process are:

- a size sz,
- 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.

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 ] \* ( left ? LeftCol[ k - 1 ] : AboveRow[ k - 1 ] ).
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 }
}
```

## 7.10.2. Inter Prediction Process

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 variable refList is set equal to 0.
2. The variable useWarp 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, useWarp is set equal to 1.
  - Otherwise, if (YMode == GLOBALMV || YMode == GLOBAL\_GLOBALMV) and GmType[ RefFrame[ refList ] ] > TRANSLATION, useWarp is set equal to 1.
  - Otherwise, useWarp is set equal to 0.
3. The rounding variables derivation process specified in [section 7.10.2.1](#) is invoked with the variable isCompound as input.

4. If setting useWarp equal to 1 would result in the bitstream not being conformant (due to either the requirement in the warp estimation process in [section 7.10.2.7](#) about the determinant being non-zero, or the requirement in the setup shear process in [section 7.10.2.5](#) about the size of the shear) then useWarp is set equal to 0.
5. If plane is equal to 0 and motion\_mode is equal to LOCALWARP and useWarp is equal to 1, the warp estimation process in [section 7.10.2.7](#) is invoked.
6. The motion vector array mv is set equal to  $Mvs[ candRow ][ candCol ][ refList ]$ .
7. 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  $ref\_frame\_idx[ RefFrame[ refList ] - LAST\_FRAME ]$ .
  - 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.
8. The motion vector scaling process in [section 7.10.2.2](#) 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.
9. If useWarp is equal to 1, the block warp process in [section 7.10.2.4](#) is invoked with plane, refList, x, y, i8, j8, w, h as inputs and the output is merged into the 2D array  $preds[ refList ]$  for  $i8 = 0..((h-1) >> 3)$  and for  $j8 = 0..((w-1) >> 3)$ . (Each invocation fills in a block of output of size w by h at x offset  $j8 * 8$  and y offset  $i8 * 8$ .)
10. If useWarp is equal to 0, the block inter prediction process in [section 7.10.2.3](#) is invoked with plane, refIdx, startX, startY, stepX, stepY, w, h, candRow, candCol as inputs and the output is assigned to the 2D array  $preds[ refList ]$ .
11. If isCompound is equal to 1, then the variable refList is set equal to 1 and steps 4 to 9 are repeated to form the prediction for the second reference.

If IsInterIntra is equal to 1, the following ordered steps apply:

1. The variable mode is set as follows:
  - If interintra\_mode is equal to II\_DC\_PRED, mode is set equal to DC\_PRED.
  - Otherwise if interintra\_mode is equal to II\_V\_PRED, mode is set equal to V\_PRED,
  - Otherwise if interintra\_mode is equal to II\_H\_PRED, mode is set equal to H\_PRED,
  - Otherwise (interintra\_mode is equal to II\_SMOOTH\_PRED), mode is set equal to SMOOTH\_PRED.
2. The variable haveLeft is set equal to AvailL
3. The variable haveAbove is set equal to AvailU
4. The variable haveAboveRight is set equal to 0

5. The variable `haveBelowLeft` is set equal to 0
6. The variable `log2W` is set equal to  $\log_2(w)$
7. The variable `log2H` is set equal to  $\log_2(h)$
8. The intra prediction process specified in [section 7.10.1](#) is invoked with `plane`, `x`, `y`, `haveLeft`, `haveAbove`, `haveAboveRight`, `haveBelowLeft`, `mode`, `log2W`, `log2H` as inputs. (This will write intra predicted samples into `CurrFrame`.)

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 7.10.2.10](#) is invoked with `w`, `h` as inputs.
- Otherwise if `compound_type` is equal to `COMPOUND_INTRA`, the inter intra mask process in [section 7.10.2.12](#) is invoked with `preds`, `plane`, `x`, `y`, `w`, `h` as inputs.
- Otherwise if `compound_type` is equal to `COMPOUND_SEG` and `plane` is equal to 0, the segment mask process in [section 7.10.2.11](#) 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 7.10.2.14](#) 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 7.10.2.13](#) is invoked with `preds`, `plane`, `x`, `y`, `w`, `h` as inputs.

If `motion_mode` is equal to `OBMC`, the overlapped motion compensation in [section 7.10.2.8](#) is invoked with `plane`, `x`, `y`, `w`, `h` as inputs.

### 7.10.2.1. 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.

### 7.10.2.2. 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 pixel, 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  $((x \ll \text{SUBPEL\_BITS}) + ((2 * \text{mv}[1]) \gg \text{subX}) + \text{halfSample})$ .

The variable `origY` is set equal to  $((y \ll \text{SUBPEL\_BITS}) + ((2 * \text{mv}[0]) \gg \text{subY}) + \text{halfSample})$ .

(`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.)

### 7.10.2.3. 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 EIGHTTAP\_SMOOTH filter.

#### 7.10.2.4. Block Warp Process

The inputs to this process are:

- 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 ( (RefUpscaledWidth[ refIdx ] + subX) >> 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  $(j8 * 8 + 4) \ll \text{subX}$ .

The variable srcY is set equal to  $(i8 * 8 + 4) \ll \text{subY}$ .

(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 motion\_mode is equal to LOCALWARP, warpParams is set equal to LocalWarpParams.
- Otherwise (motion\_mode is not equal to LOCALWARP), warpParams is set equal to gm\_params[ RefFrame[ 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 7.10.2.5](#) is invoked with warpParams as input, and the outputs are assigned to alpha, beta, gamma, and delta.

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 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 }
}
```

### 7.10.2.5. Setup Shear Process

The input for this process is an array warpParams representing an affine transformation.

The outputs from this process are 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 7.10.2.6](#) 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) ).

It is a requirement of bitstream conformance that  $4 * \text{Abs}(\text{alpha0}) + 7 * \text{Abs}(\text{beta0})$  is less than  $(1 \ll \text{WARPEDMODEL\_PREC\_BITS})$ .

It is a requirement of bitstream conformance that  $4 * \text{Abs}(\text{gamma0}) + 4 * \text{Abs}(\text{delta0})$  is less than  $(1 \ll \text{WARPEDMODEL\_PREC\_BITS})$ .

The output variables 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
```

### 7.10.2.6. 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  $\text{Abs}(d) - (1 \ll n)$ .

The variable  $f$  is set as follows:

- If  $n > \text{DIV\_LUT\_BITS}$ ,  $f$  is set equal to  $\text{Round2}(e, n - \text{DIV\_LUT\_BITS})$ .
- Otherwise,  $f$  is set equal to  $e \ll (\text{DIV\_LUT\_BITS} - n)$ .

The output variable  $\text{divShift}$  is set equal to  $(n + \text{DIV\_LUT\_PREC\_BITS})$ .

The output variable  $\text{divFactor}$  is set as follows:

- If  $d$  is less than 0,  $\text{divFactor}$  is set equal to  $-\text{Div\_Lut}[f]$ .
- Otherwise,  $\text{divFactor}$  is set equal to  $\text{Div\_Lut}[f]$ .

The lookup table  $\text{Div\_Lut}$  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
}
```

### 7.10.2.7. Warp Estimation Process

This process produces the array  $\text{LocalWarpParams}$  based on  $\text{NumSamples}$  candidates in  $\text{CandList}$  by performing a least squares fit.

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  $A[0][0] * A[1][1] - A[0][1] * A[0][1]$ .

It is a requirement of bitstream conformance that det is not equal to 0.

The resolve divisor process specified in [section 7.10.2.6](#) 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))
}
```

### 7.10.2.8. 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 neighbour 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[ candRow ][ candCol ][ 0 ].
2. The variable refIdx is set equal to ref\_frame\_idx[ RefFrames[ candRow ][ candCol ][ 0 ] - LAST\_FRAME ].
3. The variable predX is set equal to (x4 \* 4) >> subX.
4. The variable predY is set equal to (y4 \* 4) >> subY.
5. The motion vector scaling process in [section 7.10.2.2](#) 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 7.10.2.3](#) 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. obmcPred[ i ][ j ] is set equal to Clip1( obmcPred[ i ][ j ] ) for i = 0..predH-1 and j = 0..predW-1.
8. The blending process in [section 7.10.2.9](#) 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 == 1) {
        return Obmc_Mask_1
    } else 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 if (length == 32) {
        return Obmc_Mask_32
    }
}

```

The blending masks are defined as follows:

```

Obmc_Mask_1[1] = { 64 }

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 }

```

### 7.10.2.9. 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 neighbouring 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  $\text{mask}[i]$ .
  - Otherwise (pass is equal to 1 meaning blend from left),  $m$  is set equal to  $\text{mask}[j]$ .
2.  $\text{CurrFrame}[\text{plane}][\text{predY} + i][\text{predX} + j]$  is set equal to  $\text{Round2}(m * \text{CurrFrame}[\text{plane}][\text{predY} + i][\text{predX} + j] + (64 - m) * \text{obmcPred}[i][j], 6)$

### 7.10.2.10. 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 specified as:

```

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, 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

### 7.10.2.11. Segment 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
  }
}

```

## 7.10.2.12. Inter Intra 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, 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, 1, 1
}
```

## 7.10.2.13. 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 ( interintra && (!subX && !subY) ) {
      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( preds[ 0 ][ y ][ x ] )
      pred1 = CurrFrame[plane][x+dstX][y+dstY]
      CurrFrame[plane][x+dstX][y+dstY] = Round2( m * pred1 + (64 - m) * pred0, InterPostRound )
    } else {
      pred0 = preds[ 0 ][ y ][ x ]
      pred1 = preds[ 1 ][ y ][ x ]
      CurrFrame[plane][x+dstX][y+dstY] = Clip1( Round2( m * pred0 + (64 - m) * pred1, 6 +
InterPostRound ) )
    }
  }
}

```

#### 7.10.2.14. 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 }
}

```

### 7.10.3. 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.

## 7.10.4. 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 luma pixels with 3 fractional bits of precision) and lumaAvg (representing the average luma intensity with 3 fractional bits of precision) is specified as:

```

lumaAvg = 0
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        lumaX = (startX + j) << subX
        lumaY = (startY + i) << subY
        lumaX = Min( lumaX, MaxLumaW - (1 << subX) )
        lumaY = Min( lumaY, MaxLumaH - (1 << subY) )
        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 pixels 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 )
    }
}

```

## 7.11. Reconstruction and Dequantization

This section details the process of reconstructing a block of coefficients using dequantization and inverse transforms.

### 7.11.1. 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  $Dc\_Qlookup[(BitDepth-8) \gg 1][Clip3(0, 255, b)]$  where  $Dc\_Qlookup$  is defined as follows:

Draft Document

```

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 data equal to `base_q_idx + data`
  3. Return `Clip3( 0, 255, data )`.
- Otherwise, if `ignoreDeltaQ` is equal to 0 and `delta_q_present` is equal to 1, return `CurrentQIndex`.
- Otherwise, return `base_q_idx`.

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 )`.

## 7.11.2. 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 5.9.33](#).

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 transpose is set equal to `( Tx_Width[ txSz ] > Tx_Height[ txSz ] )`.

The variable txSize is derived as follows:

- If transpose is equal to 1, txSize is set equal to `Transpose_Tx_Size[ txSz ]`.
- Otherwise (transpose is equal to 0), txSize is set equal to txSz.

The variable txType is derived as follows:

- If transpose is equal to 1, txType is set equal to `Transpose_Tx_Type[ PlaneTxType ]`.
- Otherwise (transpose is equal to 0), txType is set equal to `PlaneTxType`.

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 txType 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 txType 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  $\text{get\_dc\_quant}(\text{plane})$ .
    - Otherwise (i, j or both are not equal to 0), the variable q is set equal to  $\text{get\_ac\_quant}(\text{plane})$ .
  - 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  $\text{Round2}(q * \text{Quantizer\_Matrix}[\text{SegQMLLevel}[\text{plane}][\text{segment\_id}]] [\text{plane} > 0][\text{Qm\_Offset}[\text{txSz}] + i * tw + j], 4)$ .
    - Otherwise, q2 is set equal to q.
  - c.  $\text{Dequant}[i][j]$  is set equal to  $\text{Clip3}(- (1 \ll (7 + \text{BitDepth})), (1 \ll (7 + \text{BitDepth})) - 1, (\text{Quant}[i * tw + j] * q2) / dqDenom)$ .

2. Invoke the 2D inverse transform block process defined in [section 7.12.2](#) with the variables txSize, txType and transpose 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  $\text{Clip1}(\text{CurrFrame}[\text{plane}][y + yy][x + xx] + (\text{transpose} ? \text{Residual}[j][i] : \text{Residual}[i][j]))$ .

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  $8 + \text{BitDepth}$  bits.

## 7.12. Inverse Transform Process

This section details the inverse transforms used during the reconstruction processes detailed in [section 7.11](#).

### 7.12.1. 1D Transforms

#### 7.12.1.1. 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 `cos128( angle )` is specified for integer values of the input angle by the following ordered steps:

1. Set a variable `angle2` equal to `angle & 255`.
2. If `angle2` is greater than or equal to 0 and less than or equal to 64, return `Cos128_Lookup[ angle2 ]`.
3. If `angle2` is greater than 64 and less than or equal to 128, return `Cos128_Lookup[ 128 - angle2 ] * -1`.
4. If `angle2` is greater than 128 and less than or equal to 192, return `Cos128_Lookup[ angle2 - 128 ] * -1`.
5. Otherwise (if `angle2` is greater than 192 and less than 256), return `Cos128_Lookup[ 256 - angle2 ]`.

Where `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 `sin128( angle )` is defined to be `cos128( angle - 64 )`.

**Note:** The `cos128` function implements the expression  $4096 * \cos( \text{angle} * \pi / 128 )$  rounded to the nearest integer. The `sin128` function implements the expression  $4096 * \sin( \text{angle} * \pi / 128 )$  rounded to the nearest integer.

When the 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 `cos128( 32 + 64 * k )` is always equal to that of `sin128( 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  $(\text{angle} \& 64) ? -T[a] + T[b] : T[a] + T[b]$ .
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 `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.

### 7.12.1.2. Inverse DCT Array Permutation Process

This process performs an in-place permutation of the array  $T$  of length  $2^n$  for  $2 \leq n \leq 5$  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  $i = 0..((1 \ll n) - 1)$ .

### 7.12.1.3. 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 7.12.1.2](#) 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  $i = 0..31$ .

#### 7.12.1.4. 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  $3 \leq n \leq 5$ .

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  $T$ .

If  $n$  is less than 5, the following steps apply for  $i = 0..(n0-1)$ :

- The variable  $a$  is set equal to  $(i + 1) \& 1$ .
- The variable  $b$  is set equal to  $a \wedge ((i \gg 1) \& 1)$ .
- The variable  $c$  is set equal to  $a \wedge ((i \gg 2) \& 1)$ .
- The variable  $d$  is set equal to  $a \wedge ((i \gg 3) \& 1)$ .
- The variable  $idx$  is set equal to  $((d \ll 3)(c \ll 2)(b \ll 1)a) \& (n0 - 1)$ .
- $T[i]$  is set equal to  $copyT[idx]$ .

Otherwise ( $n$  is equal to 5), the following steps apply for  $i = 0..(n0-1)$ :

- The variable  $a$  is set equal to  $i \& 1$ .
- The variable  $b$  is set equal to  $a \wedge ((i \gg 1) \& 1)$ .
- The variable  $c$  is set equal to  $b \wedge ((i \gg 2) \& 1)$ .
- The variable  $d$  is set equal to  $c \wedge ((i \gg 3) \& 1)$ .
- The variable  $e$  is set equal to  $d \wedge ((i \gg 4) \& 1)$ .



- The variable  $idx$  is set equal to  $(a \ll 4) \mid (b \ll 3) \mid (c \ll 2) \mid (d \ll 1) \mid e$ .
- $T[i]$  is set equal to  $e ? (-copyT[idx]) : copyT[idx]$ .

### 7.12.1.5. 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  $3 \leq n \leq 5$ .

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  $T$ .

If  $n$  is less than 5, 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  $(i \& 1) ? (-copyT[idx]) : copyT[idx]$ .

Otherwise ( $n$  is equal to 5), the following steps apply for  $i = 0..(n0-1)$ :

- $T[i]$  is set equal to  $(i \& 1) ? copyT[n0 - i - 1] : copyT[i + 1]$ .

### 7.12.1.6. 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.

### 7.12.1.7. 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 7.12.1.4](#) 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 7.12.1.5](#) with the input variable  $n$  set to 3.

### 7.12.1.8. 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 7.12.1.4](#) 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  $B(4 + 8 * j + 3 * i, 5 + 8 * j + i, 48 - 32 * i, 1, r)$  and  $B(12 + 3 * i, 13 + i, 48 - 32 * i, 1, r)$  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 7.12.1.5](#) with the input variable  $n$  set to 4.

### 7.12.1.9. 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 7.12.1.6](#), with the input variable r.
- Otherwise, if n is equal to 3, invoke the inverse ADST8 process in [section 7.12.1.7](#), with the input variable r.
- Otherwise (n is equal to 4), invoke the inverse ADST16 process in [section 7.12.1.8](#), with the input variable r.

### 7.12.1.10. 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

```

### 7.12.1.11. 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 )

```

### 7.12.1.12. Inverse Identity Transform 8 Process

The process does an in-place transform of the array T (of length 8) by the following calculation for  $i = 0..7$ :

$$T[i] = T[i] * 2$$

### 7.12.1.13. Inverse Identity Transform 16 Process

The process does an in-place transform of the array T (of length 16) by the following calculation for  $i = 0..15$ :

$$T[i] = \text{Round2}(T[i] * 11586, 12)$$

### 7.12.1.14. Inverse Identity Transform 32 Process

The process does an in-place transform of the array T (of length 32) by the following calculation for  $i = 0..31$ :

$$T[i] = T[i] * 4$$

### 7.12.1.15. Inverse Identity Transform 64 Process

The process does an in-place transform of the array T (of length 64) by the following calculation for  $i = 0..63$ :

$$T[i] = \text{Round2}(T[i] * 23172, 12)$$

### 7.12.1.16. 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  $2 \leq n \leq 6$ .

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 7.12.1.11](#).
- Otherwise, if n is equal to 3, invoke the inverse identity transform 8 process in [section 7.12.1.12](#).
- Otherwise, if n is equal to 4, invoke the inverse identity transform 16 process in [section 7.12.1.13](#).
- Otherwise if n is equal to 5, invoke the inverse identity transform 32 process in [section 7.12.1.14](#).
- Otherwise (n is equal to 6), invoke the inverse identity transform 64 process in [section 7.12.1.15](#).

## 7.12.2. 2D Inverse Transform

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 inputs to this process are:

- a variable txSz that specifies the transform size,
- a variable txType that specifies the transform type,
- a variable transpose, which, if equal to 1, signifies that the transform should be transposed.

Set the variable log2W equal to Tx\_Width\_Log2[ txSz ].

Set the variable log2H equal to 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 whMax equal to Max(w, h).

Set the variable rowShift equal to Transform\_Row\_Shift[ txSz ].

Set the variable colShift equal to 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 ( transpose ? Dequant[ j ][ i ] : Dequant[ i ][ j ] ).
  - Otherwise,  $T[j]$  is set equal to 0.
- If Abs( log2W - log2H ) 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 7.12.1.10](#) with shift equal to 2.
- Otherwise, if txType is equal to one of DCT\_DCT, ADST\_DCT, FLIPADST\_DCT or H\_DCT, invoke the inverse DCT process as specified in [section 7.12.1.3](#) with the input variable n equal to log2W and the input variable r equal to rowClampRange.

- Otherwise, if txType 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 7.12.1.9](#) with input variable n equal to log2W and the input variable r equal to rowClampRange.
- Otherwise, invoke the inverse identity transform process specified in [section 7.12.1.16](#) with the input variable n equal to log2W.
- 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 << ( colClampRange - 1 ) ), ( 1 << ( 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 7.12.1.10](#) with shift equal to 0.
- Otherwise, if txType is equal to one of DCT\_DCT, DCT\_ADST, DCT\_FLIPADST or V\_DCT, invoke the inverse DCT process as specified in [section 7.12.1.3](#) with the input variable n equal to log2H and the input variable r equal to colClampRange.
- Otherwise, if txType 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 7.12.1.9](#) 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 7.12.1.16](#) 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
}
```

## 7.13. Loop Filter Process

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[ 0 ] || loop_filter_level[ 1 ] ) ) ||
        ( 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 7.13.1](#) 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.

**Note:** The loop filter applies after the macroblocks have been “reconstructed” (i.e., had their prediction summed with their residual); correct decoding is predicated on the fact that already-constructed portions of the current frame referenced via intra prediction are not yet filtered.

## 7.13.1. 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 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 FrameWidth, onScreen is set equal to 0.
- Otherwise, if y is greater than or equal to FrameHeight, onScreen is set equal to 0.
- Otherwise, if pass is equal to 0 and x is equal to 0, onScreen is set equal to 0.
- Otherwise, if pass is equal to 1 and y is equal to 0, onScreen is set equal to 0.
- Otherwise, onScreen is set equal to 1.

If 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 prevRow and prevCol (containing the location of the mode info block on the other side of the boundary) are derived as follows:

- Set prevRow equal to  $\text{row} - (\text{dy} \ll \text{subY})$
- Set prevCol equal to  $\text{col} - (\text{dx} \ll \text{subX})$

Set the variable MiSize equal to  $\text{MiSizes}[\text{row}][\text{col}]$ .

Set the variable txSz equal to  $\text{LoopfilterTxSizes}[\text{plane}][\text{row}][\text{col}]$ .

Set the variable planeSize equal to  $\text{get\_plane\_residual\_size}(\text{MiSize}, \text{plane})$

Set the variable skip equal to `Skips[ row ][ col ]`.

Set the variable `isIntra` equal to `RefFrames[ row ][ col ][ 0 ] <= INTRA_FRAME`.

Set the variable `prevTxSz` equal to `get_tx_size( plane, TxSizes[ prevRow ][ prevCol ] )`.

Set the variable `prevSkip` equal to `Skips[ prevRow ][ prevCol ]`.

Set the variable `prevIsIntra` equal to `RefFrames[ prevRow ][ prevCol ][ 0 ] <= INTRA_FRAME`.

The variable `isBlockEdge` (equal to 1 if the samples cross a prediction block edge) is derived as follows:

- If `pass` is equal to 0 and `xP` is an exact multiple of `Block_Width[ planeSize ]`, `isBlockEdge` is set equal to 1.
- Otherwise, if `pass` is equal to 1 and `yP` is an exact multiple of `Block_Height[ planeSize ]`, `isBlockEdge` is set equal to 1.
- Otherwise, `isBlockEdge` is set equal to 0.

The variable `isTxEdge` (equal to 1 if the samples cross a transform block edge) is derived as follows:

- If `pass` is equal to 0 and `xP` is an exact multiple of `Tx_Width[ txSz ]`, `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 or `prevSkip` is equal to 0 or `prevIsIntra` is equal to 1, `applyFilter` is set equal to 1.
- Otherwise `applyFilter` is set equal to 0.

The filter size process specified in [section 7.13.2](#) 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 7.13.3](#) 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 7.13.3](#) 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 7.13.5](#) 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.

## 7.13.2. 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.

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 set equal to  $\text{Min}(\text{prevTxSz}, \text{txSz})$ .

The output variable filterSize is derived as follows:

- If plane is equal to 0, filterSize is set equal to  $\text{Min}(\text{TX\_16X16}, \text{baseSize})$ ,
- Otherwise, (plane is greater than 0), filterSize is set equal to  $\text{Min}(\text{TX\_8X8}, \text{baseSize})$ .

## 7.13.3. 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  $\text{SegmentIds}[\text{row}][\text{col}]$ .

- 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 equal to `NEARESTMV` or `NEARMV` or `NEWMV`, `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 7.13.4](#) 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 `Clip3( 1, 9 - loop_filter_sharpness, lvl >> shift )`.
- Otherwise, `limit` is set equal to `Max( 1, lvl >> shift )`.

The output variable `blimit` is set equal to `2 * (lvl + 2) + limit`.

The output variable `thresh` is set equal to `lvl >> 4`.

## 7.13.4. 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 ( plane == 0 ) ? pass : ( plane + 1 ).

The variable baseFilterLevel is set equal to Clip3( 0, MAX\_LOOP\_FILTER, deltaLF + loop\_filter\_level[ 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. lvlSeg is set equal to FeatureData[ segment ][ feature ] + lvlSeg.
  - b. lvlSeg is set equal to Clip3( 0, MAX\_LOOP\_FILTER, lvlSeg ).
4. If loop\_filter\_delta\_enabled is equal to 1, then the following ordered steps apply:
  - a. The variable nShift is set equal to lvlSeg >> 5.
  - b. If ref is equal to INTRA\_FRAME, then lvlSeg is set equal to lvlSeg + ( loop\_filter\_ref\_deltas[ INTRA\_FRAME ] << nShift ).
  - c. Otherwise, if ref is not equal to INTRA\_FRAME, then lvlSeg is set equal to lvlSeg + ( loop\_filter\_ref\_deltas[ ref ] << nShift ) + ( loop\_filter\_mode\_deltas[ modeType ] << nShift ).
  - d. lvlSeg is set equal to Clip3(0, MAX\_LOOP\_FILTER, lvlSeg).
5. Return lvlSeg.

### 7.13.5. Sample Filtering 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 of this process are modified values in the array CurrFrame.

First the filter mask process specified in [section 7.13.5.1](#) 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 TX\_4X4 or flatMask is equal to 0, the narrow filter process specified in [section 7.13.5.2](#) is invoked with the additional input variable hevMask.
- Otherwise, if filterSize is equal to TX\_8X8 or flatMask2 is equal to 0, the wide filter process specified in [section 7.13.5.3](#) is invoked with the additional input variable log2Size set to 3.
- Otherwise, the wide filter process specified in [section 7.13.5.3](#) is invoked with the additional input variable log2Size set to 4.

### 7.13.5.1. 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 >= TX\_8X8),
- flatMask2 (only used if filterSize >= TX\_16X16).

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 TX\_16X16.

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 TX\_4X4, 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 TX\_8X8, 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
if ( filterLen >= 4 ) {
    mask |= (Abs( p1 - p0 ) > limitBd)
    mask |= (Abs( q1 - q0 ) > limitBd)
    mask |= (Abs( p0 - q0 ) * 2 + Abs( p1 - q1 ) / 2 > blimitBd)
} else if ( filterLen >= 6 ) {
    mask |= (Abs( p2 - p1 ) > limitBd)
    mask |= (Abs( q2 - q1 ) > limitBd)
} else 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 >= TX\_8X8. 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 >= TX_8X8 ) {
    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 >= TX\_16X16. 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 >= TX_16X16 ) {
    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)
}

```

### 7.13.5.2. 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 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
}

```

### 7.13.5.3. 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.

## 7.14. 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 pixels.

The CDEF filter is applied on each 8 by 8 block of pixels 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 7.14.1](#) is invoked with r, c, and idx as inputs.

## 7.14.1. 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 ]
    }
}
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* ].

If *skip* is equal to 0, the CDEF direction process specified in [section 7.14.2](#) 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  $(\text{priStr} == 0) ? 0 : \text{yDir}$ .
4. The variable `varStr` is set equal to  $(\text{var} \gg 6) ? \text{Min}(\text{FloorLog2}(\text{var} \gg 6), 12) : 0$ .
5. The variable `priStr` is set equal to  $(\text{var} ? (\text{priStr} * (4 + \text{varStr}) + 8) \gg 4 : 0)$ .
6. The variable `damping` is set equal to `CdefDamping + coeffShift`.
7. The CDEF filter process specified in [section 7.14.3](#) 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  $(\text{priStr} == 0) ? 0 : \text{Cdef\_Uv\_Dir}[\text{subsampling\_x}][\text{subsampling\_y}][\text{yDir}]$ .
12. The variable `damping` is set equal to `CdefDamping + coeffShift - 1`.
13. The CDEF filter process specified in [section 7.14.3](#) is invoked with plane equal to 1, `r`, `c`, `priStr`, `secStr`, `damping`, and `dir` as input.
14. The CDEF filter process specified in [section 7.14.3](#) 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} }
}
```

## 7.14.2. 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:

Draft Document

```

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
}

```

### 7.14.3. 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)
                sum += Cdef_Pri_Taps[(priStr >> coeffShift) & 1][k] * constrain(p - x, priStr,
damping)

                if (p != CDEF_VERY_LARGE) {
                    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)
                    sum += Cdef_Sec_Taps[(priStr >> coeffShift) & 1][k] * constrain(s - x, secStr,
damping)

                    if (s != CDEF_VERY_LARGE) {
                        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) )
}

```

and `cdef_get_at` fetches a pixel from `CurrFrame` if it is available, or returns `CDEF_VERY_LARGE` otherwise, and is defined as:

```

cdef_get_at(plane, x0, y0, i, j, dir, k, sign, subX, subY) {
    if ( subX != subY )
        dir = subX ? Cdef_422Directions[ dir ] : Cdef_440Directions[ dir ]
    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 ) )
        return CurrFrame[ plane ][ y ][ x ]
    else
        return CDEF_VERY_LARGE
}

```

where `Cdef_Directions`, `Cdef_422Directions` and `Cdef_440Directions` are constant lookup tables 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 } }
}

Cdef_422Directions[8] = { 7, 0, 2, 4, 5, 6, 6, 6 }

Cdef_440Directions[8] = { 1, 2, 2, 2, 3, 4, 6, 0 }

```

## 7.15. Upscaling Process

Input to this process is an array `inputFrame` of width `FrameWidth` and height `FrameHeight`.

The output of this process is a horizontally upscaled 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:

Draft Document

```

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
    for (i = 0; i < TileCols; i++) {
        downscaledX0 = MiColStarts[i] << (MI_SIZE_LOG2 - subX)
        upscaledX0 = (downscaledX0 * SuperresDenom) / SUPERRES_NUM
        if (i == TileCols - 1) {
            fullTileX1 = miW * MI_SIZE
            upscaledX1 = upscaledPlaneW
        } else {
            fullTileX1 = MiColStarts[i+1] << (MI_SIZE_LOG2 - subX)
            upscaledX1 = (fullTileX1 * SuperresDenom) / SUPERRES_NUM
        }
        minX = 0
        maxX = miW * MI_SIZE - 1
        for (y = 0; y < planeH; y++) {
            for (x = upscaledX0; x < upscaledX1; 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][sampleX][y]
                    sum += px * Upscale_Filter[srcXSubpel][k]
                }
                outputFrame[plane][x][y] = 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.

## 7.16. 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 pixels 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 UpscaledCdefFrame. (The contents of LrFrame will later be overwritten for blocks that require restoration filtering.)

If allow\_intrabc is equal to 1, then the process returns immediately after performing this copy.

Otherwise, loop restoration is applied for each each tile as follows:

```
for ( TileNum = 0; TileNum < NumTiles; TileNum++ ) {
    TileRow = TileNum / TileCols
    TileCol = TileNum % TileCols
    MiRowStart = MiRowStarts[ TileRow ]
    MiRowEnd = MiRowStarts[ TileRow + 1 ]
    MiColStart = MiColStarts[ TileCol ]
    MiColEnd = MiColStarts[ TileCol + 1 ]
    tileLeft = upscale_mi_to_pos(MiColStart, 0)
    tileRight = upscale_mi_to_pos(MiColEnd, 0)
    tileTop = MiRowStart * MI_SIZE
    tileBottom = MiRowEnd * MI_SIZE
    for ( y = tileTop; y < tileBottom; y += MI_SIZE ) {
        for ( x = tileLeft; x < tileRight; x += MI_SIZE ) {
            for ( plane = 0; plane < NumPlanes; plane++ ) {
                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 7.16.1](#) is invoked with plane, row, and col as inputs.

### 7.16.1. 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  $(\text{lumaY} + 8) / 64$ .

**Note:** The stripes are offset upwards by 8 luma pixels 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 `LoopRestorationSize[ plane ]`.

The variable `unitRows` (specifying the number of restoration units down the tile) is set equal to  $\text{count\_units\_in\_tile}(\text{unitSize}, (\text{MiRowEnd} - \text{MiRowStart}) * \text{MI\_SIZE} \gg \text{subY})$ .

The variable `tileLeft` (specifying the left edge of the tile in units of samples in the current plane) is set equal to `upscale_mi_to_pos(MiColStart, plane)`.

The variable `tileRight` (specifying the right edge of the tile in units of samples in the current plane) is set equal to `upscale_mi_to_pos(MiColEnd, plane)`.

The variable `unitCols` (specifying the number of restoration units across the tile) is set equal to  $\text{count\_units\_in\_tile}(\text{unitSize}, \text{tileRight} - \text{tileLeft})$ .

**Note:** The number of restoration units in a tile can be different for chroma and luma.

The variable `unitRow` (specifying the vertical index of the current loop restoration unit) is set equal to  $\text{Min}(\text{unitRows} - 1, ((\text{row} - \text{MiRowStart}) * \text{MI\_SIZE} + 8) \gg \text{subY}) / \text{unitSize}$ .

The variable `unitCol` (specifying the horizontal index of the current loop restoration unit) is set equal to  $\text{Min}(\text{unitCols} - 1, (\text{col} * \text{MI\_SIZE} - \text{tileLeft}) / \text{unitSize})$ .

The horizontal extent of the space allowed for filtering is specified as follows:

- `TileStartX` is set equal to 0
- `TileEndX` is set equal to  $\text{Round2}(\text{UpscaledWidth}, \text{subX}) - 1$

The vertical extent of the space allowed for filtering is specified as follows:

- `TileStartY` is set equal to 0
- `TileEndY` 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  $(\text{MI\_SIZE} \gg \text{subX})$ .

The variable `h` is set equal to  $(\text{MI\_SIZE} \gg \text{subY})$ .

(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 `LrType[ TileNum ][ plane ][ unitRow ][ unitCol ]`.

The filter to used depends on `rType` as follows:

- If `rType` is equal to `RESTORE_WIENER`, the Wiener filter process specified in [section 7.16.4](#) is invoked with `plane`, `unitRow`, `unitCol`, `x`, `y`, `w`, and `h` as inputs.
- Otherwise, if `rType` is equal to `RESTORE_SGRPROJ`, the self guided filter process specified in [section 7.16.2](#) is invoked with `plane`, `unitRow`, `unitCol`, `x`, `y`, `w`, and `h` as inputs.
- Otherwise (`rType` is equal to `RESTORE_NONE`), no filtering is applied.

## 7.16.2. Self Guided 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 offset of the loop restoration unit with the current tile,



- 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 flt0 and flt1 are prepared by the following ordered steps:

1. The variable set is set equal to LrSgrSet[ TileNum ][ plane ][ unitRow ][ unitCol ].
2. The variable pass is set equal to 0.
3. The box filter process specified in [section 7.16.3](#) is invoked with plane, x, y, w, h, set, and pass as inputs, and the output assigned to flt0.
4. The variable pass is set equal to 1.
5. The box filter process specified in [section 7.16.3](#) is invoked with plane, x, y, w, h, set, and pass as inputs, and the output assigned to flt1.

The restoration process is then applied for each sample as follows:

```

xqd0 = LrSgrXqd[ TileNum ][ plane ][ unitRow ][ unitCol ][ 0 ]
xqd1 = LrSgrXqd[ TileNum ][ plane ][ unitRow ][ unitCol ][ 1 ]
xq0 = xqd0
r0 = Sgr_Params[ set ][ 0 ]
r1 = Sgr_Params[ set ][ 2 ]
if ( r1 == 0 ) {
    xq1 = 0
} else {
    xq1 = (1 << SGRPROJ_PRJ_BITS) - xq0 - xqd1
}
for ( i = 0; i < h; i++ ) {
    for ( j = 0; j < w; j++ ) {
        u = UpscaledCdefFrame[ plane ][ y + i ][ x + j ] << SGRPROJ_RST_BITS
        v = u << SGRPROJ_PRJ_BITS
        if ( r0 )
            v += xq0 * ( flt0[ i ][ j ] - u )
        if ( r1 )
            v += xq1 * ( flt1[ i ][ j ] - u )
        w = Round2( v, SGRPROJ_RST_BITS + SGRPROJ_PRJ_BITS )
        LrFrame[ plane ][ y + i ][ x + j ] = Clip1( w )
    }
}

```

### 7.16.3. 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) )
        b = Round2( b, BitDepth - 8 )
        p = Max( 0, a * n - b * b )
        z = Round2( p * s, SGRPROJ_MTABLE_BITS )
        if ( z >= 255 )
            a2 = 256
        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 7.16.6](#) 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 }
}

```

## 7.16.4. 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 offset of the loop restoration unit with the current tile,
- 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 pixels 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 7.10.2.1](#) is invoked with the input variable isCompound set equal to 0.

The Wiener coefficient process specified in [section 7.16.5](#) is invoked with an input of LrWiener[ TileNum ][ plane ][ unitRow ][ unitCol ][ 0 ] and the output assigned to vfilter.

The Wiener coefficient process specified in [section 7.16.5](#) is invoked with an input of LrWiener[ TileNum ][ 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 7.16.6](#) should be invoked.

**Note:** The intermediate result is clipped so that  $(\text{intermediate}[r][c] + \text{offset})$  fits in an unsigned variable with  $(\text{BitDepth} + 3)$  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 )
    }
}

```

## 7.16.5. 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.

## 7.16.6. 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(TileEndX, x)
x = Max(TileStartX, x)
y = Min(TileEndY, y)
y = Max(TileStartY, 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.

## 7.17. Output Process

This process is invoked to output a frame from the decoding process.

First 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 7.17.1](#) is invoked with inputs of  $w$ ,  $h$ ,  $\text{subX}$ , and  $\text{subY}$ . (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.

**Note:** Even when spatial scalability is being used, this output process will output all frames within the operating point. This matches the reference decoder implementation and is useful for conformance testing, but applications may choose to only use a subset of the output frames. For example, applications that are displaying the decoded video are expected to only display the frames from the highest spatial layer within the operating point.

### 7.17.1. Film Grain Synthesis Process

The inputs to this process are:

- variables  $w$  and  $h$  specifying the width and height of the frame,
- variables  $\text{subX}$  and  $\text{subY}$  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 7.17.1.2](#) is invoked.
6. The scaling lookup initialization process specified in [section 7.17.1.3](#) is invoked.
7. The add noise process specified in [section 7.17.1.4](#) is invoked with w, h, subX, and subY as inputs.

### 7.17.1.1. 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.

### 7.17.1.2. 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 7.17.1.1](#).

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, Round2( s, shift ) )
  }
}

```

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 ^ 0xd849
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
        }
        pos++
      }
    }
    CbGrain[ y ][ x ] = Clip3( GrainMin, GrainMax, Round2( s0, shift ) )
    CrGrain[ y ][ x ] = Clip3( GrainMin, GrainMax, 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.

### 7.17.1.3. 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 ]
}

```

#### 7.17.1.4. 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  $(34 \gg \text{subY})$  samples high and  $(w \gg \text{subX})$  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 ) {
                                v = old * 27 + g * 17
                            } else {
                                v = 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 ]
                        v = 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 >> ( 4 + planeSubY )
    i = y - ( lumaNum << ( 4 + 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 ) {
            v = old * 27 + g * 17
          } else {
            v = 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 ]
          v = 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)
    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 ) )
            }
            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 ) )
            }
            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 )
    }
}
}

```

## 7.18. 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`:

```

for ( list = 0; list < 2; list++ ) {
    MfRefFrames[ row ][ col ][ list ] = NONE
    MfMvs[ row ][ col ][ list ][ 0 ] = 0
    MfMvs[ row ][ col ][ list ][ 1 ] = 0
}
for ( list = 0; list < 2; list++ ) {
    r = RefFrames[ row ][ col ][ list ]
    if ( r > INTRA_FRAME ) {
        dist = get_relative_dist( RefOrderHint[ r ], OrderHint )
        if ( dist != 0 ) {
            refIdx = dist > 0 ? 1 : 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 ][ refIdx ] = r
                MfMvs[ row ][ col ][ refIdx ][ 0 ] = mvRow
                MfMvs[ row ][ col ][ refIdx ][ 1 ] = mvCol
            }
        }
    }
}
}

```

## 7.19. 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.
- `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 ]` is set equal to `RefOrderHint[ j ]` for  $j = 0..NUM\_REF\_FRAMES-1$ .
- `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 ][ list ]` is set equal to `MfRefFrames[ row ][ col ][ list ]` for  $list = 0..1$ , for  $row = 0..MiRows-1$ , for  $col = 0..MiCols-1$ .
- `SavedMvs[ i ][ row ][ col ][ list ][ comp ]` is set equal to `MfMvs[ row ][ col ][ list ][ comp ]` for  $comp = 0..1$ , for  $list = 0..1$ , 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.

## 7.20. Reference Frame Loading Process

This process is the reverse of the reference frame update process specified in [section 7.19](#). 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.

- 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 ].
- RefOrderHint[ j ] is set equal to SavedOrderHints[ frame\_to\_show\_map\_idx ][ j ] for j = 0..NUM\_REF\_FRAMES-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.

- $MfRefFrames[ row ][ col ][ list ]$  is set equal to  $SavedRefFrames[ frame\_to\_show\_map\_idx ][ row ][ col ][ list ]$  for  $list = 0..1$ , for  $row = 0..MiRows-1$ , for  $col = 0..MiCols-1$ .
- $MfMvs[ row ][ col ][ list ][ comp ]$  is set equal to  $SavedMvs[ frame\_to\_show\_map\_idx ][ row ][ col ][ list ][ comp ]$  for  $comp = 0..1$ , for  $list = 0..1$ , 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 6.7.20](#)).
- 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

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( readPadding ), read\_symbol( cdf ), and read\_bool( ) are used in this Specification to indicate the entropy decoding operation.

#### 8.2.1. 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 UpdateCdfs (representing if this is the biggest tile seen so far) is set equal to (sz > MaxTileSize).

The variable MaxTileSize (that tracks the largest tile size seen so far) is set equal to Max( MaxTileSize, sz ).

The variable buf is read using the f(15) parsing process.

The variable SymbolValue is set to ((1 << 15) - 1) ^ buf.

The variable SymbolRange is set to 1 << 15.

The variable SymbolMaxBits is set to  $8 * sz - 15$ .

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

## 8.2.2. 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 8.2.4](#).

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.

### 8.2.3. Exit Process for Symbol Decoder

This process is invoked when the function `exit_symbol( readPadding )` is called from the syntax structure.

It is a requirement of bitstream conformance that `SymbolMaxBits` is greater than or equal to 0 whenever this process is invoked.

If `readPadding` is equal to 1, the padding syntax element is read using the `f(SymbolMaxBits)` parsing process.

If `readPadding` is equal to 1, it is a requirement of bitstream conformance that padding is equal to 0.

**Note:** The padding is required to make the bit position byte aligned. It is legal for frames to end with more than one byte of padding.

If `readPadding` is equal to 0 (this happens for the last tile in a tile group), no reading of padding is performed at this level of the syntax. The remaining bits will be read by `trailing_bits` at the end of the OBU.

If `UpdateCdfs` is equal to 1, 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`.

### 8.2.4. 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.

## 8.2.5. 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  $1 \ll 15$ .

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 read from the bitstream.
2. The variable `SymbolRange` is set to `SymbolRange << bits`.

3. The variable `newData` is read using the `f(bits)` parsing process.
4. The variable `SymbolValue` is set to  $\text{newData} \wedge ((\text{SymbolValue} + 1) \ll \text{bits}) - 1$ .
5. The variable `SymbolMaxBits` is set to `SymbolMaxBits - 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

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 8.3.1 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 )`.

### 8.3.1. 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 Default\_Intra\_Frame\_Y\_Mode\_Cdf[ 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 TileUVModeCdfAllowedCdf[ YMode ].
- Otherwise, if Lossless is equal to 0 and Max( Block\_Width[ MiSize ], Block\_Height[ MiSize ] ) <= 32, the cdf is given by TileUVModeCdfAllowedCdf[ YMode ].
- Otherwise, the cdf is given by TileUVModeCdfNotAllowedCdf[ 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.

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.

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[ MiCol ][ MiRow - 1 ] ) {
    aboveW = Block_Width[ MiSizes[ MiCol ][ MiRow - 1 ] ]
} else if ( AvailU ) {
    aboveW = get_above_tx_width( MiRow, MiCol )
} else {
    aboveW = 0
}

if ( AvailL && IsInters[ MiCol - 1 ][ MiRow ] ) {
    leftH = Block_Height[ MiSizes[ MiCol - 1 ][ MiRow ] ]
} 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  = Max( Block_Width[ MiSize ], Block_Height[ MiSize ] )
maxTxSz = find_tx_size( size, size )
txSzSqrUp = Tx_Size_Sqr_Up[ txSz ]
ctx = (txSzSqrUp != maxTxSz && maxTxSz > TX_8X8) * 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 ( !AvailU ) {
        return 64
    } else if ( Skips[ row - 1 ][ col ] && IsInters[ row - 1 ][ col ] ) {
        return Block_Width[ MiSizes[ row - 1 ][ col ] ]
    } else {
        return Tx_Width[ InterTxSizes[ row - 1 ][ col ] ]
    }
}

get_left_tx_height( row, col ) {
    if ( !AvailL ) {
        return 64
    } else if ( Skips[ row ][ col - 1 ] && IsInters[ row ][ col - 1 ] ) {
        return Block_Height[ MiSizes[ row ][ col - 1 ] ]
    } else {
        return Tx_Height[ InterTxSizes[ row ][ col - 1 ] ]
    }
}

```

**segment\_id:** The cdf is given by `TileSegmentIdCdf[ ctx ]` where `ctx` is computed by:

```

if (prevUL < 0 || prevU < 0 || prevL < 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 = (AboveRefFrame[ 0 ] == CompFixedRef)
            ^ (LeftRefFrame[ 0 ] == CompFixedRef)
    else if ( AboveSingle )
        ctx = 2 + (AboveRefFrame[ 0 ] == CompFixedRef || AboveIntra)
    else if ( LeftSingle )
        ctx = 2 + (LeftRefFrame[ 0 ] == CompFixedRef || LeftIntra)
    else
        ctx = 4
} else if ( AvailU ) {
    if ( AboveSingle )
        ctx = AboveRefFrame[ 0 ] == CompFixedRef
    else
        ctx = 3
} else if ( AvailL ) {
    if ( LeftSingle )
        ctx = LeftRefFrame[ 0 ] == CompFixedRef
    else
        ctx = 3
} else {
    ctx = 1
}

```

**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]

bw = Block_Width[ MiSize ]
bh = Block_Height[ MiSize ]
mask = use_128x128_superblock ? 255 : 127

if ( plane > 0 ) {
    bw = bw >> subsampling_x
    bh = bh >> subsampling_y
    w4 = w4 >> subsampling_x
    h4 = h4 >> subsampling_y
}
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 ) & mask ] )
    }
    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 ) & mask ]
        left |= LeftDcContext[ plane ][ ( y4 + i ) & mask ]
    }
}
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, startX, startY )
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 ][ ctx ]`, where `ctx` is computed as for `eob_pt_16`.

**eob\_pt\_1024:** the cdf is given by `TileEobPt1024Cdf[ ptype ][ ctx ]`, where `ctx` is computed as for `eob_pt_16`.

**eob\_extra:** the cdf is given by `TileEobExtraCdf[ txSzCtx ][ ptype ][ eobPt ]`.

**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++
        }
    }
}

mask = use_128x128_superblock ? 255 : 127
for ( k = 0; k < h4; k++ ) {
    if ( y4 + k < maxY4 ) {
        sign = LeftDcContext[ plane ][ ( y4 + k ) & mask ]
        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 ( txClass == 2 ) {
            if ( row == 0 ) {
                ctx = mag + 7
            } else {
                ctx = mag + 14
            }
        } 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	<code>TilePaletteSize2YColorCdf[ ctx ]</code>
3	<code>TilePaletteSize3YColorCdf[ ctx ]</code>
4	<code>TilePaletteSize4YColorCdf[ ctx ]</code>
5	<code>TilePaletteSize5YColorCdf[ ctx ]</code>
6	<code>TilePaletteSize6YColorCdf[ ctx ]</code>
7	<code>TilePaletteSize7YColorCdf[ ctx ]</code>
8	<code>TilePaletteSize8YColorCdf[ ctx ]</code>

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_Size_Sqr[ txSz ] ]
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) {
    if (ref0 <= INTRA_FRAME || ref1 <= INTRA_FRAME)
        return 0

    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 ]
```

**interintra\_mode:** The cdf is given by TileInterIntraModeCdf[ ctx ], where ctx is computed as follows:

```
ctx = Size_Group[ MiSize ]
```

**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 computed as follows:

```
ctx = (signU - 1) * 3 + signV
```

**Note:** 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 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.

Draft Document

## 9. Additional Tables

This section contains tables that do not naturally fit in the main sections of the Specification.

### 9.1. 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_16x32	Mrow_Scan_32x16
Mcol_Scan_32x32	Mrow_Scan_32x32



Table	Transpose
Mcol_Scan_16x4	Mrow_Scan_4x16
Mcol_Scan_32x8	Mrow_Scan_8x32
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_16x32	Mcol_Scan_32x16
Mrow_Scan_32x32	Mcol_Scan_32x32
Mrow_Scan_16x4	Mcol_Scan_4x16
Mrow_Scan_32x8	Mcol_Scan_8x32

```
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
}

```

```

Mcol_Scan_16x32[ 512 ] = {
    0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224,
    240, 256, 272, 288, 304, 320, 336, 352, 368, 384, 400, 416, 432, 448, 464,
    480, 496, 1, 17, 33, 49, 65, 81, 97, 113, 129, 145, 161, 177, 193,
    209, 225, 241, 257, 273, 289, 305, 321, 337, 353, 369, 385, 401, 417, 433,
    449, 465, 481, 497, 2, 18, 34, 50, 66, 82, 98, 114, 130, 146, 162,
    178, 194, 210, 226, 242, 258, 274, 290, 306, 322, 338, 354, 370, 386, 402,
    418, 434, 450, 466, 482, 498, 3, 19, 35, 51, 67, 83, 99, 115, 131,
    147, 163, 179, 195, 211, 227, 243, 259, 275, 291, 307, 323, 339, 355, 371,
    387, 403, 419, 435, 451, 467, 483, 499, 4, 20, 36, 52, 68, 84, 100,
    116, 132, 148, 164, 180, 196, 212, 228, 244, 260, 276, 292, 308, 324, 340,
    356, 372, 388, 404, 420, 436, 452, 468, 484, 500, 5, 21, 37, 53, 69,
    85, 101, 117, 133, 149, 165, 181, 197, 213, 229, 245, 261, 277, 293, 309,
    325, 341, 357, 373, 389, 405, 421, 437, 453, 469, 485, 501, 6, 22, 38,
    54, 70, 86, 102, 118, 134, 150, 166, 182, 198, 214, 230, 246, 262, 278,
    294, 310, 326, 342, 358, 374, 390, 406, 422, 438, 454, 470, 486, 502, 7,
    23, 39, 55, 71, 87, 103, 119, 135, 151, 167, 183, 199, 215, 231, 247,
    263, 279, 295, 311, 327, 343, 359, 375, 391, 407, 423, 439, 455, 471, 487,
    503, 8, 24, 40, 56, 72, 88, 104, 120, 136, 152, 168, 184, 200, 216,
    232, 248, 264, 280, 296, 312, 328, 344, 360, 376, 392, 408, 424, 440, 456,
    472, 488, 504, 9, 25, 41, 57, 73, 89, 105, 121, 137, 153, 169, 185,
    201, 217, 233, 249, 265, 281, 297, 313, 329, 345, 361, 377, 393, 409, 425,
    441, 457, 473, 489, 505, 10, 26, 42, 58, 74, 90, 106, 122, 138, 154,
    170, 186, 202, 218, 234, 250, 266, 282, 298, 314, 330, 346, 362, 378, 394,
    410, 426, 442, 458, 474, 490, 506, 11, 27, 43, 59, 75, 91, 107, 123,
    139, 155, 171, 187, 203, 219, 235, 251, 267, 283, 299, 315, 331, 347, 363,
    379, 395, 411, 427, 443, 459, 475, 491, 507, 12, 28, 44, 60, 76, 92,
    108, 124, 140, 156, 172, 188, 204, 220, 236, 252, 268, 284, 300, 316, 332,
    348, 364, 380, 396, 412, 428, 444, 460, 476, 492, 508, 13, 29, 45, 61,
    77, 93, 109, 125, 141, 157, 173, 189, 205, 221, 237, 253, 269, 285, 301,
    317, 333, 349, 365, 381, 397, 413, 429, 445, 461, 477, 493, 509, 14, 30,
    46, 62, 78, 94, 110, 126, 142, 158, 174, 190, 206, 222, 238, 254, 270,
    286, 302, 318, 334, 350, 366, 382, 398, 414, 430, 446, 462, 478, 494, 510,
    15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239,
    255, 271, 287, 303, 319, 335, 351, 367, 383, 399, 415, 431, 447, 463, 479,
    495, 511
}

```



```

Mrow_Scan_16x32[ 512 ] = {
    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, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269,
    270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
    285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299,
    300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314,
    315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329,
    330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344,
    345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359,
    360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,
    375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
    390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404,
    405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419,
    420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434,
    435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449,
    450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
    465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479,
    480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494,
    495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509,
    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
}

```

```

Mcol_Scan_32x16[ 512 ] = {
    0,  32, 64, 96,  128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480,
    1,  33, 65, 97,  129, 161, 193, 225, 257, 289, 321, 353, 385, 417, 449, 481,
    2,  34, 66, 98,  130, 162, 194, 226, 258, 290, 322, 354, 386, 418, 450, 482,
    3,  35, 67, 99,  131, 163, 195, 227, 259, 291, 323, 355, 387, 419, 451, 483,
    4,  36, 68, 100, 132, 164, 196, 228, 260, 292, 324, 356, 388, 420, 452, 484,
    5,  37, 69, 101, 133, 165, 197, 229, 261, 293, 325, 357, 389, 421, 453, 485,
    6,  38, 70, 102, 134, 166, 198, 230, 262, 294, 326, 358, 390, 422, 454, 486,
    7,  39, 71, 103, 135, 167, 199, 231, 263, 295, 327, 359, 391, 423, 455, 487,
    8,  40, 72, 104, 136, 168, 200, 232, 264, 296, 328, 360, 392, 424, 456, 488,
    9,  41, 73, 105, 137, 169, 201, 233, 265, 297, 329, 361, 393, 425, 457, 489,
    10, 42, 74, 106, 138, 170, 202, 234, 266, 298, 330, 362, 394, 426, 458, 490,
    11, 43, 75, 107, 139, 171, 203, 235, 267, 299, 331, 363, 395, 427, 459, 491,
    12, 44, 76, 108, 140, 172, 204, 236, 268, 300, 332, 364, 396, 428, 460, 492,
    13, 45, 77, 109, 141, 173, 205, 237, 269, 301, 333, 365, 397, 429, 461, 493,
    14, 46, 78, 110, 142, 174, 206, 238, 270, 302, 334, 366, 398, 430, 462, 494,
    15, 47, 79, 111, 143, 175, 207, 239, 271, 303, 335, 367, 399, 431, 463, 495,
    16, 48, 80, 112, 144, 176, 208, 240, 272, 304, 336, 368, 400, 432, 464, 496,
    17, 49, 81, 113, 145, 177, 209, 241, 273, 305, 337, 369, 401, 433, 465, 497,
    18, 50, 82, 114, 146, 178, 210, 242, 274, 306, 338, 370, 402, 434, 466, 498,
    19, 51, 83, 115, 147, 179, 211, 243, 275, 307, 339, 371, 403, 435, 467, 499,
    20, 52, 84, 116, 148, 180, 212, 244, 276, 308, 340, 372, 404, 436, 468, 500,
    21, 53, 85, 117, 149, 181, 213, 245, 277, 309, 341, 373, 405, 437, 469, 501,
    22, 54, 86, 118, 150, 182, 214, 246, 278, 310, 342, 374, 406, 438, 470, 502,
    23, 55, 87, 119, 151, 183, 215, 247, 279, 311, 343, 375, 407, 439, 471, 503,
    24, 56, 88, 120, 152, 184, 216, 248, 280, 312, 344, 376, 408, 440, 472, 504,
    25, 57, 89, 121, 153, 185, 217, 249, 281, 313, 345, 377, 409, 441, 473, 505,
    26, 58, 90, 122, 154, 186, 218, 250, 282, 314, 346, 378, 410, 442, 474, 506,
    27, 59, 91, 123, 155, 187, 219, 251, 283, 315, 347, 379, 411, 443, 475, 507,
    28, 60, 92, 124, 156, 188, 220, 252, 284, 316, 348, 380, 412, 444, 476, 508,
    29, 61, 93, 125, 157, 189, 221, 253, 285, 317, 349, 381, 413, 445, 477, 509,
    30, 62, 94, 126, 158, 190, 222, 254, 286, 318, 350, 382, 414, 446, 478, 510,
    31, 63, 95, 127, 159, 191, 223, 255, 287, 319, 351, 383, 415, 447, 479, 511
}

```

```

Mrow_Scan_32x16[ 512 ] = {
    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, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269,
    270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
    285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299,
    300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314,
    315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329,
    330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344,
    345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359,
    360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,
    375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
    390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404,
    405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419,
    420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434,
    435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449,
    450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
    465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479,
    480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494,
    495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509,
    510, 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

}

```

Mcol_Scan_32x32[ 1024 ] = {
    0, 32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416,
    448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864,
    896, 928, 960, 992, 1, 33, 65, 97, 129, 161, 193, 225, 257, 289,
    321, 353, 385, 417, 449, 481, 513, 545, 577, 609, 641, 673, 705, 737,
    769, 801, 833, 865, 897, 929, 961, 993, 2, 34, 66, 98, 130, 162,
    194, 226, 258, 290, 322, 354, 386, 418, 450, 482, 514, 546, 578, 610,
    642, 674, 706, 738, 770, 802, 834, 866, 898, 930, 962, 994, 3, 35,
    67, 99, 131, 163, 195, 227, 259, 291, 323, 355, 387, 419, 451, 483,
    515, 547, 579, 611, 643, 675, 707, 739, 771, 803, 835, 867, 899, 931,
    963, 995, 4, 36, 68, 100, 132, 164, 196, 228, 260, 292, 324, 356,
    388, 420, 452, 484, 516, 548, 580, 612, 644, 676, 708, 740, 772, 804,
    836, 868, 900, 932, 964, 996, 5, 37, 69, 101, 133, 165, 197, 229,
    261, 293, 325, 357, 389, 421, 453, 485, 517, 549, 581, 613, 645, 677,
    709, 741, 773, 805, 837, 869, 901, 933, 965, 997, 6, 38, 70, 102,
    134, 166, 198, 230, 262, 294, 326, 358, 390, 422, 454, 486, 518, 550,
    582, 614, 646, 678, 710, 742, 774, 806, 838, 870, 902, 934, 966, 998,
    7, 39, 71, 103, 135, 167, 199, 231, 263, 295, 327, 359, 391, 423,
    455, 487, 519, 551, 583, 615, 647, 679, 711, 743, 775, 807, 839, 871,
    903, 935, 967, 999, 8, 40, 72, 104, 136, 168, 200, 232, 264, 296,
    328, 360, 392, 424, 456, 488, 520, 552, 584, 616, 648, 680, 712, 744,
    776, 808, 840, 872, 904, 936, 968, 1000, 9, 41, 73, 105, 137, 169,
    201, 233, 265, 297, 329, 361, 393, 425, 457, 489, 521, 553, 585, 617,
    649, 681, 713, 745, 777, 809, 841, 873, 905, 937, 969, 1001, 10, 42,
    74, 106, 138, 170, 202, 234, 266, 298, 330, 362, 394, 426, 458, 490,
    522, 554, 586, 618, 650, 682, 714, 746, 778, 810, 842, 874, 906, 938,
    970, 1002, 11, 43, 75, 107, 139, 171, 203, 235, 267, 299, 331, 363,
    395, 427, 459, 491, 523, 555, 587, 619, 651, 683, 715, 747, 779, 811,
    843, 875, 907, 939, 971, 1003, 12, 44, 76, 108, 140, 172, 204, 236,
    268, 300, 332, 364, 396, 428, 460, 492, 524, 556, 588, 620, 652, 684,
    716, 748, 780, 812, 844, 876, 908, 940, 972, 1004, 13, 45, 77, 109,
    141, 173, 205, 237, 269, 301, 333, 365, 397, 429, 461, 493, 525, 557,
    589, 621, 653, 685, 717, 749, 781, 813, 845, 877, 909, 941, 973, 1005,
    14, 46, 78, 110, 142, 174, 206, 238, 270, 302, 334, 366, 398, 430,
    462, 494, 526, 558, 590, 622, 654, 686, 718, 750, 782, 814, 846, 878,
    910, 942, 974, 1006, 15, 47, 79, 111, 143, 175, 207, 239, 271, 303,
    335, 367, 399, 431, 463, 495, 527, 559, 591, 623, 655, 687, 719, 751,
    783, 815, 847, 879, 911, 943, 975, 1007, 16, 48, 80, 112, 144, 176,
    208, 240, 272, 304, 336, 368, 400, 432, 464, 496, 528, 560, 592, 624,
    656, 688, 720, 752, 784, 816, 848, 880, 912, 944, 976, 1008, 17, 49,
    81, 113, 145, 177, 209, 241, 273, 305, 337, 369, 401, 433, 465, 497,
    529, 561, 593, 625, 657, 689, 721, 753, 785, 817, 849, 881, 913, 945,
    977, 1009, 18, 50, 82, 114, 146, 178, 210, 242, 274, 306, 338, 370,
    402, 434, 466, 498, 530, 562, 594, 626, 658, 690, 722, 754, 786, 818,
    850, 882, 914, 946, 978, 1010, 19, 51, 83, 115, 147, 179, 211, 243,
    275, 307, 339, 371, 403, 435, 467, 499, 531, 563, 595, 627, 659, 691,
    723, 755, 787, 819, 851, 883, 915, 947, 979, 1011, 20, 52, 84, 116,
    148, 180, 212, 244, 276, 308, 340, 372, 404, 436, 468, 500, 532, 564,
    596, 628, 660, 692, 724, 756, 788, 820, 852, 884, 916, 948, 980, 1012,
    21, 53, 85, 117, 149, 181, 213, 245, 277, 309, 341, 373, 405, 437,

```

469, 501, 533, 565, 597, 629, 661, 693, 725, 757, 789, 821, 853, 885,  
917, 949, 981, 1013, 22, 54, 86, 118, 150, 182, 214, 246, 278, 310,  
342, 374, 406, 438, 470, 502, 534, 566, 598, 630, 662, 694, 726, 758,  
790, 822, 854, 886, 918, 950, 982, 1014, 23, 55, 87, 119, 151, 183,  
215, 247, 279, 311, 343, 375, 407, 439, 471, 503, 535, 567, 599, 631,  
663, 695, 727, 759, 791, 823, 855, 887, 919, 951, 983, 1015, 24, 56,  
88, 120, 152, 184, 216, 248, 280, 312, 344, 376, 408, 440, 472, 504,  
536, 568, 600, 632, 664, 696, 728, 760, 792, 824, 856, 888, 920, 952,  
984, 1016, 25, 57, 89, 121, 153, 185, 217, 249, 281, 313, 345, 377,  
409, 441, 473, 505, 537, 569, 601, 633, 665, 697, 729, 761, 793, 825,  
857, 889, 921, 953, 985, 1017, 26, 58, 90, 122, 154, 186, 218, 250,  
282, 314, 346, 378, 410, 442, 474, 506, 538, 570, 602, 634, 666, 698,  
730, 762, 794, 826, 858, 890, 922, 954, 986, 1018, 27, 59, 91, 123,  
155, 187, 219, 251, 283, 315, 347, 379, 411, 443, 475, 507, 539, 571,  
603, 635, 667, 699, 731, 763, 795, 827, 859, 891, 923, 955, 987, 1019,  
28, 60, 92, 124, 156, 188, 220, 252, 284, 316, 348, 380, 412, 444,  
476, 508, 540, 572, 604, 636, 668, 700, 732, 764, 796, 828, 860, 892,  
924, 956, 988, 1020, 29, 61, 93, 125, 157, 189, 221, 253, 285, 317,  
349, 381, 413, 445, 477, 509, 541, 573, 605, 637, 669, 701, 733, 765,  
797, 829, 861, 893, 925, 957, 989, 1021, 30, 62, 94, 126, 158, 190,  
222, 254, 286, 318, 350, 382, 414, 446, 478, 510, 542, 574, 606, 638,  
670, 702, 734, 766, 798, 830, 862, 894, 926, 958, 990, 1022, 31, 63,  
95, 127, 159, 191, 223, 255, 287, 319, 351, 383, 415, 447, 479, 511,  
543, 575, 607, 639, 671, 703, 735, 767, 799, 831, 863, 895, 927, 959,  
991, 1023

}



```

Mrow_Scan_32x32[ 1024 ] = {
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, 256, 257, 258, 259,
260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,
312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,
572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584,
585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597,
598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610,
611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623,
624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636,

```

```

637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649,
650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662,
663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675,
676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701,
702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727,
728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740,
741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753,
754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766,
767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779,
780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792,
793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805,
806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818,
819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831,
832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,
845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857,
858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870,
871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883,
884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896,
897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909,
910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922,
923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948,
949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961,
962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974,
975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987,
988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000,
1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013,
1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 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
}

```

```

Mcol_Scan_8x32[ 256 ] = {
    0,  8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112,
    120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232,
    240, 248, 1,  9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97,
    105, 113, 121, 129, 137, 145, 153, 161, 169, 177, 185, 193, 201, 209, 217,
    225, 233, 241, 249, 2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82,
    90, 98, 106, 114, 122, 130, 138, 146, 154, 162, 170, 178, 186, 194, 202,
    210, 218, 226, 234, 242, 250, 3, 11, 19, 27, 35, 43, 51, 59, 67,
    75, 83, 91, 99, 107, 115, 123, 131, 139, 147, 155, 163, 171, 179, 187,
    195, 203, 211, 219, 227, 235, 243, 251, 4, 12, 20, 28, 36, 44, 52,
    60, 68, 76, 84, 92, 100, 108, 116, 124, 132, 140, 148, 156, 164, 172,
    180, 188, 196, 204, 212, 220, 228, 236, 244, 252, 5, 13, 21, 29, 37,
    45, 53, 61, 69, 77, 85, 93, 101, 109, 117, 125, 133, 141, 149, 157,
    165, 173, 181, 189, 197, 205, 213, 221, 229, 237, 245, 253, 6, 14, 22,
    30, 38, 46, 54, 62, 70, 78, 86, 94, 102, 110, 118, 126, 134, 142,
    150, 158, 166, 174, 182, 190, 198, 206, 214, 222, 230, 238, 246, 254, 7,
    15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119, 127,
    135, 143, 151, 159, 167, 175, 183, 191, 199, 207, 215, 223, 231, 239, 247,
    255
}

```

```

Mrow_Scan_8x32[ 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_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
}

```

```

Mcol_Scan_32x8[ 256 ] = {
    0,  32, 64, 96,  128, 160, 192, 224, 1,  33, 65, 97,  129, 161, 193, 225,
    2,  34, 66, 98,  130, 162, 194, 226, 3,  35, 67, 99,  131, 163, 195, 227,
    4,  36, 68, 100, 132, 164, 196, 228, 5,  37, 69, 101, 133, 165, 197, 229,
    6,  38, 70, 102, 134, 166, 198, 230, 7,  39, 71, 103, 135, 167, 199, 231,
    8,  40, 72, 104, 136, 168, 200, 232, 9,  41, 73, 105, 137, 169, 201, 233,
    10, 42, 74, 106, 138, 170, 202, 234, 11, 43, 75, 107, 139, 171, 203, 235,
    12, 44, 76, 108, 140, 172, 204, 236, 13, 45, 77, 109, 141, 173, 205, 237,
    14, 46, 78, 110, 142, 174, 206, 238, 15, 47, 79, 111, 143, 175, 207, 239,
    16, 48, 80, 112, 144, 176, 208, 240, 17, 49, 81, 113, 145, 177, 209, 241,
    18, 50, 82, 114, 146, 178, 210, 242, 19, 51, 83, 115, 147, 179, 211, 243,
    20, 52, 84, 116, 148, 180, 212, 244, 21, 53, 85, 117, 149, 181, 213, 245,
    22, 54, 86, 118, 150, 182, 214, 246, 23, 55, 87, 119, 151, 183, 215, 247,
    24, 56, 88, 120, 152, 184, 216, 248, 25, 57, 89, 121, 153, 185, 217, 249,
    26, 58, 90, 122, 154, 186, 218, 250, 27, 59, 91, 123, 155, 187, 219, 251,
    28, 60, 92, 124, 156, 188, 220, 252, 29, 61, 93, 125, 157, 189, 221, 253,
    30, 62, 94, 126, 158, 190, 222, 254, 31, 63, 95, 127, 159, 191, 223, 255
}

```

```

Mrow_Scan_32x8[ 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
}

```

## 9.2. 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  $x$ ,  $Max\_Tx\_Size[x]$  returns the largest square transform size that can be used for blocks of size  $x$ .

```

Max_Tx_Size[ BLOCK_SIZES ] = {
    TX_4X4, TX_4X4, TX_4X4, TX_8X8,
    TX_8X8, TX_8X8, TX_16X16, TX_16X16,
    TX_16X16, TX_32X32, TX_32X32, TX_32X32,
    TX_64X64, TX_64X64, TX_64X64, TX_64X64,
    TX_4X4, TX_4X4, TX_8X8, TX_8X8,
    TX_16X16, TX_16X16
}

```

For a luma block size  $x$ ,  $\text{Max\_Tx\_Size\_Rect}[x]$  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  $p$ ,  $\text{Partition\_Subsize}[p][x]$  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, or for a 4x4 block size.



```

Partition_Subsize[ 10 ][ BLOCK_SIZES ] = {
    {
        BLOCK_INVALID,
        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_128X32,
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_32X128,
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[ MB_MODE_COUNT ] = {
    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, // NEARESTMV
    DCT_DCT, // NEARMV
    DCT_DCT, // GLOBALMV
    DCT_DCT // NEWMV
}

```

```

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, 111, 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
}
```

For a transform set *s* and transform type *t*, returns whether the transform type is allowed in set *s*.

```
Tx_Type_In_Set[ TX_SET_TYPES ][ TX_TYPES ] = {
    {
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    },
    {
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
    },
    {
        1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 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, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    },
    {
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    },
}
```

```
Wedge_Bits[ BLOCK_SIZES ] = {
    0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0,
    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
}

```

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
}

```

## 9.3. 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 }
}
```

```

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 }
}

```

```

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 ] = {
    { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
    { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
    { 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 ] = {
    { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
    { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
    { 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 }  
}
```

Draft Document

```

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 ] = {
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 } },
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 } },
{ { 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[ 4 ][ 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 }
    },
    {
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 }
    },
    {
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
        { 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },
    }
}

```

```
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
{ 4681, 9362, 14043, 18725, 23406, 28087, 32768, 0 },  
}  
}
```

Draft Document

```

Default_Intra_Tx_Type_Set2_Cdf[ 4 ][ 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 },
},
{
    { 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 }
  }
}

```

```

Default_Inter_Tx_Type_Set1_Cdf[ 4 ][ 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[ 4 ][ 13 ] = {
  { 2731, 5461, 8192, 10923, 13653, 16384, 19115, 21845,
    24576, 27307, 30037, 32768, 0 },
  { 2731, 5461, 8192, 10923, 13653, 16384, 19115, 21845,
    24576, 27307, 30037, 32768, 0 },
  { 770, 2421, 5225, 12907, 15819, 18927, 21561, 24089, 26595,
    28526, 30529, 32768, 0 },
  { 2731, 5461, 8192, 10923, 13653, 16384, 19115, 21845,
    24576, 27307, 30037, 32768, 0 }
}

```

```

Default_Inter_Tx_Type_Set3_Cdf[ 4 ][ 3 ] = {
  { 16384, 32768, 0 },
  { 4167, 32768, 0 },
  { 1998, 32768, 0 },
  { 748, 32768, 0 }
}

```

```

Default_Compound_IdxCdf[ 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_IdxCdf[ 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 },
    { 11820, 32768, 0 },
    { 7701, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 }
}

```

```
Default_Inter_Intra_Cdf[ BLOCK_SIZE_GROUPS ][ 3 ] = {  
    { 16384, 32768, 0 },  
    { 26887, 32768, 0 },  
    { 27597, 32768, 0 },  
    { 30237, 32768, 0 }  
}
```

```
Default_Inter_Intra_Mode_Cdf[ BLOCK_SIZE_GROUPS ][ INTERINTRA_MODES + 1 ] = {  
    { 8192, 16384, 24576, 32768, 0 },  
    { 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 }
}

```

[illegible]

```

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 }
};

```

```

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
}

```

```

DefaultTxbSkipCdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ TXB_SKIP_CONTEXTS ][ 3 ] = {
{
  { { 32605, 32768, 0 },
    { 10627, 32768, 0 },
    { 13167, 32768, 0 },
    { 22671, 32768, 0 },
    { 18710, 32768, 0 },
    { 26810, 32768, 0 },
    { 32126, 32768, 0 },
    { 3637, 32768, 0 },
    { 16905, 32768, 0 },
    { 28677, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32613, 32768, 0 },
    { 12454, 32768, 0 },
    { 14380, 32768, 0 },
    { 25061, 32768, 0 },
    { 18405, 32768, 0 },
    { 27262, 32768, 0 },
    { 32308, 32768, 0 },
    { 2819, 32768, 0 },
    { 15663, 32768, 0 },
    { 28353, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32209, 32768, 0 },
    { 10135, 32768, 0 },
    { 14508, 32768, 0 },
    { 23776, 32768, 0 },
    { 21415, 32768, 0 },
    { 25031, 32768, 0 },
    { 31713, 32768, 0 },
    { 5176, 32768, 0 },
    { 16175, 32768, 0 },
    { 29220, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 26786, 32768, 0 },
    { 11656, 32768, 0 },
    { 17377, 32768, 0 },
    { 26137, 32768, 0 },
    { 16384, 32768, 0 },
    { 26600, 32768, 0 },
    { 31359, 32768, 0 },
    { 11087, 32768, 0 },
    { 25919, 32768, 0 },

```



```

    { 31989, 32768, 0 },
    { 15766, 32768, 0 },
    { 26756, 32768, 0 },
    { 29848, 32768, 0 } },
  { { 21611, 32768, 0 },
    { 5461, 32768, 0 },
    { 28087, 32768, 0 },
    { 25817, 32768, 0 },
    { 16384, 32768, 0 },
    { 28672, 32768, 0 },
    { 32208, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
},
{
  { { 31574, 32768, 0 },
    { 10880, 32768, 0 },
    { 13298, 32768, 0 },
    { 20932, 32768, 0 },
    { 17307, 32768, 0 },
    { 25362, 32768, 0 },
    { 31521, 32768, 0 },
    { 4062, 32768, 0 },
    { 15737, 32768, 0 },
    { 27189, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32595, 32768, 0 },
    { 11226, 32768, 0 },
    { 12558, 32768, 0 },
    { 22364, 32768, 0 },
    { 17483, 32768, 0 },
    { 26136, 32768, 0 },
    { 32081, 32768, 0 },
    { 3509, 32768, 0 },
    { 14439, 32768, 0 },
    { 26145, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32485, 32768, 0 },
    { 12729, 32768, 0 },
    { 14812, 32768, 0 },
    { 22261, 32768, 0 },
    { 20672, 32768, 0 },

```

```

    { 25408, 32768, 0 },
    { 31463, 32768, 0 },
    { 3565, 32768, 0 },
    { 15559, 32768, 0 },
    { 27508, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 30478, 32768, 0 },
    { 10847, 32768, 0 },
    { 15746, 32768, 0 },
    { 25044, 32768, 0 },
    { 23164, 32768, 0 },
    { 25746, 32768, 0 },
    { 31778, 32768, 0 },
    { 7642, 32768, 0 },
    { 22163, 32768, 0 },
    { 30328, 32768, 0 },
    { 8536, 32768, 0 },
    { 24702, 32768, 0 },
    { 30147, 32768, 0 } },
  { { 26684, 32768, 0 },
    { 3208, 32768, 0 },
    { 17644, 32768, 0 },
    { 27445, 32768, 0 },
    { 16384, 32768, 0 },
    { 30341, 32768, 0 },
    { 32556, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
},
{ { { 29941, 32768, 0 },
    { 9962, 32768, 0 },
    { 11511, 32768, 0 },
    { 19393, 32768, 0 },
    { 14863, 32768, 0 },
    { 23680, 32768, 0 },
    { 30957, 32768, 0 },
    { 3960, 32768, 0 },
    { 15120, 32768, 0 },
    { 25656, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
  { { 32541, 32768, 0 },
    { 9637, 32768, 0 },

```

```

{ 11428, 32768, 0 },
{ 20348, 32768, 0 },
{ 16737, 32768, 0 },
{ 24878, 32768, 0 },
{ 31599, 32768, 0 },
{ 3938, 32768, 0 },
{ 13268, 32768, 0 },
{ 23916, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 } },
{ { 32560, 32768, 0 },
{ 14332, 32768, 0 },
{ 14899, 32768, 0 },
{ 21998, 32768, 0 },
{ 22306, 32768, 0 },
{ 27670, 32768, 0 },
{ 31776, 32768, 0 },
{ 4071, 32768, 0 },
{ 15952, 32768, 0 },
{ 26342, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 } },
{ { 31747, 32768, 0 },
{ 11539, 32768, 0 },
{ 15576, 32768, 0 },
{ 24995, 32768, 0 },
{ 25295, 32768, 0 },
{ 25947, 32768, 0 },
{ 32154, 32768, 0 },
{ 6818, 32768, 0 },
{ 21562, 32768, 0 },
{ 28731, 32768, 0 },
{ 3421, 32768, 0 },
{ 21167, 32768, 0 },
{ 30760, 32768, 0 } },
{ { 30272, 32768, 0 },
{ 7198, 32768, 0 },
{ 22483, 32768, 0 },
{ 27619, 32768, 0 },
{ 27483, 32768, 0 },
{ 29545, 32768, 0 },
{ 32353, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 },
{ 16384, 32768, 0 } } },

```

```

{ { { 30198, 32768, 0 },
    { 7807, 32768, 0 },
    { 9940, 32768, 0 },
    { 19103, 32768, 0 },
    { 14088, 32768, 0 },
    { 22936, 32768, 0 },
    { 30320, 32768, 0 },
    { 3227, 32768, 0 },
    { 11686, 32768, 0 },
    { 21633, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
{ { 32322, 32768, 0 },
    { 6667, 32768, 0 },
    { 9014, 32768, 0 },
    { 18042, 32768, 0 },
    { 14207, 32768, 0 },
    { 22732, 32768, 0 },
    { 30904, 32768, 0 },
    { 3422, 32768, 0 },
    { 10341, 32768, 0 },
    { 19457, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
{ { 32463, 32768, 0 },
    { 10932, 32768, 0 },
    { 14618, 32768, 0 },
    { 21939, 32768, 0 },
    { 20793, 32768, 0 },
    { 26929, 32768, 0 },
    { 31366, 32768, 0 },
    { 4351, 32768, 0 },
    { 12704, 32768, 0 },
    { 22868, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
{ { 31987, 32768, 0 },
    { 9885, 32768, 0 },
    { 21616, 32768, 0 },
    { 23702, 32768, 0 },
    { 23316, 32768, 0 },
    { 28245, 32768, 0 },
    { 32112, 32768, 0 },
    { 5654, 32768, 0 },
    { 18625, 32768, 0 },
    { 25878, 32768, 0 },
    { 1762, 32768, 0 },

```

```

    { 14824, 32768, 0 },
    { 26877, 32768, 0 } },
  { { 32374, 32768, 0 },
    { 14704, 32768, 0 },
    { 24688, 32768, 0 },
    { 28913, 32768, 0 },
    { 29407, 32768, 0 },
    { 30985, 32768, 0 },
    { 32453, 32768, 0 },
    { 16384, 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 ] = {
  { { { 1063, 1368, 2747, 5584, 32768, 0 },
    { 269, 442, 1012, 4118, 32768, 0 } } },
    { { { 5514, 7645, 11953, 19700, 32768, 0 },
    { 3979, 6565, 11760, 20491, 32768, 0 } } } },
    { { { 2689, 3569, 6487, 11023, 32768, 0 },
    { 469, 787, 1694, 6678, 32768, 0 } } },
    { { { 9194, 11895, 17248, 24761, 32768, 0 },
    { 6407, 10248, 16706, 25074, 32768, 0 } } } },
    { { { 4637, 6257, 10483, 16764, 32768, 0 },
    { 686, 1307, 2637, 9683, 32768, 0 } } },
    { { { 12745, 15999, 21878, 28227, 32768, 0 },
    { 9092, 14278, 20810, 28556, 32768, 0 } } } },
    { { { 6544, 9316, 15031, 22497, 32768, 0 },
    { 1059, 2140, 4290, 12545, 32768, 0 } } },
    { { { 20492, 23668, 28204, 31451, 32768, 0 },
    { 14289, 21097, 25960, 31272, 32768, 0 } } } }
}

```

```

Default_Eob_Pt_32_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 7 ] = {
  { { { 722, 898, 1491, 2584, 6481, 32768, 0 },
        { 199, 308, 781, 2096, 6618, 32768, 0 } } },
  { { { 6788, 8626, 12634, 17791, 26086, 32768, 0 },
        { 4005, 6267, 10129, 15919, 29241, 32768, 0 } } } },
  { { { 1699, 2138, 3410, 5506, 11583, 32768, 0 },
        { 261, 436, 1430, 3761, 10049, 32768, 0 } } },
  { { { 11733, 13930, 18373, 23020, 29777, 32768, 0 },
        { 7461, 12117, 19218, 25087, 31599, 32768, 0 } } } },
  { { { 2969, 3841, 5727, 9044, 16543, 32768, 0 },
        { 407, 736, 1930, 5443, 13281, 32768, 0 } } },
  { { { 15519, 18234, 22613, 26662, 31670, 32768, 0 },
        { 9677, 15468, 23735, 29377, 32496, 32768, 0 } } } },
  { { { 4072, 5629, 8217, 12353, 21017, 32768, 0 },
        { 755, 1245, 3058, 7733, 17443, 32768, 0 } } },
  { { { 24075, 26423, 29061, 30983, 32407, 32768, 0 },
        { 14789, 20299, 30158, 31898, 32478, 32768, 0 } } } }
}

```

```

Default_Eob_Pt_64_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 8 ] = {
  { { { 373, 465, 678, 1108, 2134, 5056, 32768, 0 },
        { 173, 275, 474, 870, 2784, 9771, 32768, 0 } } },
  { { { 6280, 8262, 11236, 14340, 18747, 26775, 32768, 0 },
        { 2057, 3515, 5746, 8425, 13442, 29671, 32768, 0 } } } },
  { { { 1149, 1395, 2137, 3544, 5779, 10343, 32768, 0 },
        { 294, 474, 838, 1682, 4499, 13540, 32768, 0 } } },
  { { { 12776, 15052, 18630, 22072, 25650, 29879, 32768, 0 },
        { 6210, 10569, 17310, 23755, 27570, 30928, 32768, 0 } } } },
  { { { 3317, 4392, 6615, 10239, 14971, 21621, 32768, 0 },
        { 588, 1079, 1973, 4055, 8456, 19237, 32768, 0 } } },
  { { { 18292, 20058, 23703, 26608, 28853, 31769, 32768, 0 },
        { 9388, 16293, 23561, 28952, 31072, 32586, 32768, 0 } } } },
  { { { 6759, 8501, 11620, 16030, 21813, 28436, 32768, 0 },
        { 1208, 2280, 4139, 7332, 13311, 24022, 32768, 0 } } },
  { { { 25358, 27110, 30078, 31694, 32475, 32724, 32768, 0 },
        { 13458, 18725, 24576, 29842, 31598, 32183, 32768, 0 } } } }
}

```

```

Default_Eob_Pt_128_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 9 ] = {
  { { { 693, 939, 1310, 2060, 3151, 5602, 12620, 32768, 0 },
      { 114, 206, 391, 1290, 3317, 6071, 12537, 32768, 0 } },
    { { { 9705, 11363, 17511, 19445, 21573, 23355, 28542, 32768, 0 },
      { 4902, 7998, 12514, 18448, 20512, 21802, 28253, 32768, 0 } } },
    { { { 1329, 1818, 2514, 3824, 6179, 9616, 17895, 32768, 0 },
      { 216, 356, 709, 2307, 6058, 10199, 19064, 32768, 0 } },
      { { { 15653, 17962, 22305, 25114, 27331, 29004, 30960, 32768, 0 },
        { 6679, 11479, 18889, 27341, 29950, 30785, 31516, 32768, 0 } } },
      { { { 2600, 3508, 5304, 8477, 12607, 17377, 24531, 32768, 0 },
        { 300, 668, 1470, 4938, 12570, 18869, 25835, 32768, 0 } },
        { { { 18739, 21483, 24858, 27514, 29455, 30605, 31968, 32768, 0 },
          { 5461, 11099, 18322, 27483, 31535, 32416, 32592, 32768, 0 } } },
          { { { 4917, 6467, 9434, 14108, 18783, 23990, 29846, 32768, 0 },
            { 682, 1653, 3738, 11800, 19886, 25614, 30629, 32768, 0 } },
            { { { 24445, 27240, 30036, 31548, 32264, 32555, 32716, 32768, 0 },
              { 7022, 11703, 21065, 28087, 29257, 30427, 31598, 32768, 0 } } } } } }
}

```

```

Default_Eob_Pt_256_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 10 ] = {
  { { { { 1911, 2651, 3878, 5131, 6362, 8149, 10305, 15539, 32768, 0 },
        { 482, 948, 1582, 2688, 6755, 9357, 13182, 18521, 32768, 0 } },
      { { { 6563, 9048, 13129, 14730, 15789, 16556, 17584, 26263, 32768, 0 },
        { 3040, 4392, 7094, 8445, 12499, 15539, 16215, 22971, 32768, 0 } } },
      { { { 2341, 2977, 4715, 6995, 10042, 13542, 17070, 22695, 32768, 0 },
        { 524, 1279, 2241, 3709, 8952, 14201, 20125, 25634, 32768, 0 } },
        { { { 13249, 15850, 19810, 22294, 24265, 25502, 26775, 29692, 32768, 0 },
          { 6353, 9697, 17053, 22403, 26749, 27084, 27418, 30093, 32768, 0 } } },
          { { { 3832, 5041, 7899, 11629, 16180, 20378, 24470, 28392, 32768, 0 },
            { 950, 2850, 4165, 5815, 11094, 18188, 25953, 29718, 32768, 0 } },
            { { { 17762, 19980, 23289, 25947, 27924, 28882, 29747, 31546, 32768, 0 },
              { 7509, 12971, 21163, 25941, 30037, 30720, 31403, 32085, 32768, 0 } } } },
              { { { 7219, 9681, 13792, 18500, 22814, 26581, 29566, 31887, 32768, 0 },
                { 1895, 4116, 6730, 9344, 15584, 23196, 28717, 31690, 32768, 0 } },
                { { { 24807, 26197, 28542, 30832, 31949, 32296, 32476, 32625, 32768, 0 },
                  { 4468, 14895, 23831, 25321, 26810, 28300, 29789, 31279, 32768, 0 } } } } } } } }
}

```

```

Default_EobPt_512_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 11 ] = {
  { { { 1022, 2121, 5275, 8179, 10996, 13027, 14783, 17418, 22377, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } },
    { { { 5208, 6744, 9847, 11629, 12582, 13150, 13580, 13949, 30694, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } } },
    { { { 3211, 4958, 7991, 10408, 13976, 16848, 19587, 22712, 27627, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } },
    { { { 12009, 15222, 20460, 23718, 25624, 26532, 26913, 27440, 32577, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } } },
    { { { 4840, 6256, 8940, 12031, 16592, 19797, 23212, 26986, 30599, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } },
    { { { 15907, 18994, 24689, 27469, 29319, 30136, 30351, 30760, 32655, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } } },
    { { { 7344, 9242, 12843, 16504, 21289, 24542, 27923, 30785, 32028, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } },
    { { { 24120, 26247, 28940, 30682, 32039, 32322, 32464, 32525, 32700, 32768, 0 },
      { 3277, 6554, 9830, 13107, 16384, 19661, 22938, 26214, 29491, 32768, 0 } } } }
}

```

```

Default_EobPt_1024_Cdf[ COEFF_CDF_Q_CTXS ][ PLANE_TYPES ][ 2 ][ 12 ] = {
  { { { 349, 769, 3512, 8048, 13380, 15861, 17645, 18394, 19942, 21726, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } },
    { { { 2394, 2967, 5141, 7838, 9658, 10737, 11041, 11142, 11277, 29734, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } } },
    { { { 2546, 3630, 9183, 13423, 18156, 21419, 23412, 24314, 25571, 27368, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } },
    { { { 6416, 7552, 12474, 16116, 18183, 19811, 20393, 20536, 20661, 32240, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } } },
    { { { 5102, 6253, 11888, 15776, 20221, 23734, 26178, 27339, 28740, 30236, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } },
    { { { 12604, 14163, 20142, 22246, 24028, 25598, 26065, 26199, 26243, 32479, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } } },
    { { { 9285, 10807, 15092, 19816, 23951, 27300, 29275, 30308, 31311, 32020, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } },
    { { { 24507, 25696, 28784, 30112, 30961, 31718, 31903, 31965, 31996, 32692, 32768, 0 },
      { 2979, 5958, 8937, 11916, 14895, 17873, 20852, 23831, 26810, 29789, 32768, 0 } } } }
}

```



```

Default_Eob_Extra_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ EOB_COEF_CONTEXTS ][ 3 ] = {
  { { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 14583, 32768, 0 }, { 14609, 32768, 0 }, { 6039, 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 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
    { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 18056, 32768, 0 }, { 16946, 32768, 0 }, { 15356, 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 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16960, 32768, 0 }, { 18623, 32768, 0 }, { 17045, 32768, 0 },
    { 11983, 32768, 0 }, { 9010, 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } },
    { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 21542, 32768, 0 }, { 20073, 32768, 0 }, { 16775, 32768, 0 },
    { 20287, 32768, 0 }, { 21990, 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 21749, 32768, 0 }, { 18940, 32768, 0 }, { 16287, 32768, 0 },
    { 18244, 32768, 0 }, { 14240, 32768, 0 }, { 15856, 32768, 0 },
    { 16732, 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 } },
    { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 22957, 32768, 0 }, { 18467, 32768, 0 }, { 18271, 32768, 0 },
    { 18538, 32768, 0 }, { 13782, 32768, 0 }, { 23742, 32768, 0 },
    { 20798, 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 } } },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },

```

```

{ 20661, 32768, 0 }, { 22431, 32768, 0 }, { 22648, 32768, 0 },
{ 20514, 32768, 0 }, { 19787, 32768, 0 }, { 13765, 32768, 0 },
{ 19496, 32768, 0 }, { 11160, 32768, 0 }, { 11234, 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 },
{ 16384, 32768, 0 } },
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 25541, 32768, 0 }, { 25051, 32768, 0 }, { 21781, 32768, 0 },
{ 20439, 32768, 0 }, { 21370, 32768, 0 }, { 18569, 32768, 0 },
{ 13370, 32768, 0 }, { 27280, 32768, 0 }, { 27699, 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 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 19181, 32768, 0 }, { 16733, 32768, 0 }, { 18422, 32768, 0 },
{ 15124, 32768, 0 }, { 22686, 32768, 0 }, { 20549, 32768, 0 },
{ 14489, 32768, 0 }, { 15881, 32768, 0 }, { 18639, 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 },
{ 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 }, { 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 14911, 32768, 0 }, { 15414, 32768, 0 }, { 9035, 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 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 19376, 32768, 0 }, { 18353, 32768, 0 }, { 19497, 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 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 17395, 32768, 0 }, { 18844, 32768, 0 }, { 17566, 32768, 0 },
{ 14742, 32768, 0 }, { 10578, 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 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } },
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 20738, 32768, 0 }, { 20524, 32768, 0 }, { 17537, 32768, 0 },
{ 24074, 32768, 0 }, { 26643, 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 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 19326, 32768, 0 }, { 18988, 32768, 0 }, { 16704, 32768, 0 },
{ 19532, 32768, 0 }, { 15510, 32768, 0 }, { 19016, 32768, 0 },
{ 20232, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } },
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 22906, 32768, 0 }, { 22105, 32768, 0 }, { 21782, 32768, 0 },
{ 20700, 32768, 0 }, { 14921, 32768, 0 }, { 23591, 32768, 0 },
{ 24978, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 25305, 32768, 0 }, { 23310, 32768, 0 }, { 18178, 32768, 0 },
{ 19735, 32768, 0 }, { 20520, 32768, 0 }, { 15769, 32768, 0 },
{ 23096, 32768, 0 }, { 13287, 32768, 0 }, { 12667, 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 },
{ 16384, 32768, 0 } } },
{ { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 26633, 32768, 0 }, { 22458, 32768, 0 }, { 22068, 32768, 0 },
{ 19405, 32768, 0 }, { 23887, 32768, 0 }, { 8856, 32768, 0 },
{ 16328, 32768, 0 }, { 31572, 32768, 0 }, { 26214, 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 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 26354, 32768, 0 }, { 20723, 32768, 0 }, { 20647, 32768, 0 },
{ 15541, 32768, 0 }, { 21027, 32768, 0 }, { 22805, 32768, 0 },
{ 12712, 32768, 0 }, { 17246, 32768, 0 }, { 21475, 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 },
    { 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 }, { 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
  { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 15912, 32768, 0 }, { 16777, 32768, 0 }, { 13160, 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 },
    { 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 }, { 16384, 32768, 0 },
    { 19931, 32768, 0 }, { 19777, 32768, 0 }, { 22966, 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 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
  { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 18112, 32768, 0 }, { 19084, 32768, 0 }, { 18295, 32768, 0 },
    { 16862, 32768, 0 }, { 15713, 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 21534, 32768, 0 }, { 20739, 32768, 0 }, { 19526, 32768, 0 },
    { 22958, 32768, 0 }, { 28379, 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
  { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 17715, 32768, 0 }, { 18575, 32768, 0 }, { 17600, 32768, 0 },
    { 19592, 32768, 0 }, { 17971, 32768, 0 }, { 19944, 32768, 0 },
    { 19606, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } }

```

```

    { 16384, 32768, 0 } },
  { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 23762, 32768, 0 }, { 25444, 32768, 0 }, { 23182, 32768, 0 },
    { 21878, 32768, 0 }, { 13577, 32768, 0 }, { 21873, 32768, 0 },
    { 29739, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
  { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 25025, 32768, 0 }, { 22643, 32768, 0 }, { 17867, 32768, 0 },
    { 20472, 32768, 0 }, { 22017, 32768, 0 }, { 19198, 32768, 0 },
    { 24447, 32768, 0 }, { 15542, 32768, 0 }, { 18678, 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 },
    { 16384, 32768, 0 } },
    { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 25073, 32768, 0 }, { 23406, 32768, 0 }, { 21609, 32768, 0 },
    { 22615, 32768, 0 }, { 23636, 32768, 0 }, { 6827, 32768, 0 },
    { 20312, 32768, 0 }, { 31618, 32768, 0 }, { 21845, 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 },
    { 16384, 32768, 0 } },
    { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 27530, 32768, 0 }, { 20747, 32768, 0 }, { 21651, 32768, 0 },
    { 18812, 32768, 0 }, { 17457, 32768, 0 }, { 22176, 32768, 0 },
    { 12059, 32768, 0 }, { 18204, 32768, 0 }, { 20602, 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 },
    { 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 }, { 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } },
  { { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 15381, 32768, 0 }, { 17118, 32768, 0 }, { 18874, 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 },
    { 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 }, { 16384, 32768, 0 },

```

```

{ 20759, 32768, 0 }, { 23346, 32768, 0 }, { 26502, 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 },
{ 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 }, { 16384, 32768, 0 },
{ 17675, 32768, 0 }, { 19214, 32768, 0 }, { 18992, 32768, 0 },
{ 20161, 32768, 0 }, { 21785, 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 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 24438, 32768, 0 }, { 25137, 32768, 0 }, { 24678, 32768, 0 },
{ 24700, 32768, 0 }, { 20852, 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 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 20315, 32768, 0 }, { 18453, 32768, 0 }, { 18924, 32768, 0 },
{ 20072, 32768, 0 }, { 19729, 32768, 0 }, { 23392, 32768, 0 },
{ 25556, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 24534, 32768, 0 }, { 28135, 32768, 0 }, { 22390, 32768, 0 },
{ 21958, 32768, 0 }, { 18725, 32768, 0 }, { 24273, 32768, 0 },
{ 31403, 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 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 22457, 32768, 0 }, { 23403, 32768, 0 }, { 19857, 32768, 0 },
{ 20241, 32768, 0 }, { 22129, 32768, 0 }, { 20539, 32768, 0 },
{ 20621, 32768, 0 }, { 21215, 32768, 0 }, { 26935, 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 },
{ 16384, 32768, 0 } } },
{ { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
{ 24305, 32768, 0 }, { 24004, 32768, 0 }, { 20950, 32768, 0 },
{ 25486, 32768, 0 }, { 18971, 32768, 0 }, { 23406, 32768, 0 },

```

```

    { 17873, 32768, 0 }, { 30840, 32768, 0 }, { 10923, 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 },
    { 16384, 32768, 0 } } },
  { { { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 25699, 32768, 0 }, { 21787, 32768, 0 }, { 21964, 32768, 0 },
    { 17686, 32768, 0 }, { 21433, 32768, 0 }, { 16384, 32768, 0 },
    { 18530, 32768, 0 }, { 21633, 32768, 0 }, { 14398, 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 },
    { 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 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 }, { 16384, 32768, 0 }, { 16384, 32768, 0 },
    { 16384, 32768, 0 } } } } }
}

```

```

DefaultDcSignCdf[ 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
] = {
  { { { { 20597, 31183, 32768, 0 },
        { 31273, 32627, 32768, 0 },
        { 31825, 32680, 32768, 0 },
        { 27224, 30407, 32768, 0 } },
    { { 28394, 31807, 32768, 0 },
      { 31660, 32638, 32768, 0 },
      { 32058, 32667, 32768, 0 },
      { 31564, 32518, 32768, 0 } } },
  { { { { 15187, 30433, 32768, 0 },
        { 32209, 32689, 32768, 0 },
        { 32628, 32745, 32768, 0 },
        { 29289, 32300, 32768, 0 } },
    { { 26024, 31469, 32768, 0 },
      { 31866, 32605, 32768, 0 },
      { 32328, 32726, 32768, 0 },
      { 32366, 32703, 32768, 0 } } },
  { { { { 5796, 25887, 32768, 0 },
        { 32280, 32684, 32768, 0 },
        { 32718, 32756, 32768, 0 },
        { 32341, 32689, 32768, 0 } },
    { { 17040, 25504, 32768, 0 },
      { 32367, 32732, 32768, 0 },
      { 32104, 32702, 32768, 0 },
      { 32536, 32731, 32768, 0 } } },
  { { { { 10486, 20733, 32768, 0 },
        { 31320, 32449, 32768, 0 },
        { 32303, 32722, 32768, 0 },
        { 32517, 32704, 32768, 0 } },
    { { 27125, 29437, 32768, 0 },
      { 31058, 31898, 32768, 0 },
      { 31343, 32056, 32768, 0 },
      { 31555, 32742, 32768, 0 } } },
  { { { { 16384, 19661, 32768, 0 },
        { 32276, 32589, 32768, 0 },
        { 30554, 32547, 32768, 0 },
        { 32549, 32713, 32768, 0 } },
    { { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 },
      { 10923, 21845, 32768, 0 } } } },
  { { { { 20771, 31640, 32768, 0 },
        { 31727, 32593, 32768, 0 },
        { 31842, 32688, 32768, 0 },
        { 29451, 31534, 32768, 0 } },
    { { 27842, 31527, 32768, 0 },
      { 31658, 32647, 32768, 0 },
      { 32108, 32699, 32768, 0 },
      { 32038, 32662, 32768, 0 } } },

```

```

{ { { 17666, 30666, 32768, 0 },
    { 32305, 32679, 32768, 0 },
    { 32591, 32751, 32768, 0 },
    { 31284, 32594, 32768, 0 } },
  { { 28021, 31540, 32768, 0 },
    { 31750, 32595, 32768, 0 },
    { 32403, 32725, 32768, 0 },
    { 32425, 32722, 32768, 0 } } },
{ { { 12056, 29503, 32768, 0 },
    { 32342, 32686, 32768, 0 },
    { 32719, 32756, 32768, 0 },
    { 32590, 32738, 32768, 0 } },
  { { 21009, 28705, 32768, 0 },
    { 31851, 32617, 32768, 0 },
    { 32262, 32743, 32768, 0 },
    { 32547, 32721, 32768, 0 } } },
{ { { 3890, 15934, 32768, 0 },
    { 30573, 31743, 32768, 0 },
    { 32509, 32749, 32768, 0 },
    { 32625, 32744, 32768, 0 } },
  { { 22429, 27807, 32768, 0 },
    { 30492, 31850, 32768, 0 },
    { 31374, 32419, 32768, 0 },
    { 31007, 32717, 32768, 0 } } },
{ { { 4830, 5875, 32768, 0 },
    { 29974, 31257, 32768, 0 },
    { 31928, 32628, 32768, 0 },
    { 31925, 32734, 32768, 0 } },
  { { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 } } } },
{ { { { 22884, 31609, 32768, 0 },
    { 31723, 32529, 32768, 0 },
    { 31905, 32699, 32768, 0 },
    { 30839, 32165, 32768, 0 } },
  { { 27149, 31080, 32768, 0 },
    { 31720, 32693, 32768, 0 },
    { 32371, 32721, 32768, 0 },
    { 32387, 32743, 32768, 0 } } } },
{ { { 21572, 31546, 32768, 0 },
    { 32053, 32643, 32768, 0 },
    { 32497, 32733, 32768, 0 },
    { 32270, 32655, 32768, 0 } },
  { { 27952, 31297, 32768, 0 },
    { 31833, 32635, 32768, 0 },
    { 32444, 32747, 32768, 0 },
    { 32513, 32742, 32768, 0 } } } },
{ { { 16700, 29042, 32768, 0 },
    { 32185, 32665, 32768, 0 },

```

```

    { 32667, 32740, 32768, 0 },
    { 32619, 32747, 32768, 0 } },
  { { 25356, 30314, 32768, 0 },
    { 31958, 32606, 32768, 0 },
    { 32451, 32715, 32768, 0 },
    { 32602, 32747, 32768, 0 } } },
  { { { 12218, 22903, 32768, 0 },
    { 31421, 32164, 32768, 0 },
    { 32454, 32748, 32768, 0 },
    { 32174, 32570, 32768, 0 } } },
    { { 22272, 28451, 32768, 0 },
    { 30802, 32293, 32768, 0 },
    { 30093, 32099, 32768, 0 },
    { 30899, 32725, 32768, 0 } } },
  { { { 5670, 16547, 32768, 0 },
    { 28947, 30241, 32768, 0 },
    { 32105, 32635, 32768, 0 },
    { 32316, 32693, 32768, 0 } } },
    { { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 },
    { 10923, 21845, 32768, 0 } } } },
{ { { { 23556, 31195, 32768, 0 },
    { 31953, 32674, 32768, 0 },
    { 32129, 32703, 32768, 0 },
    { 31963, 32607, 32768, 0 } } },
    { { 27624, 31497, 32768, 0 },
    { 32234, 32756, 32768, 0 },
    { 32558, 32740, 32768, 0 },
    { 32640, 32747, 32768, 0 } } } },
  { { { 23114, 31301, 32768, 0 },
    { 32277, 32729, 32768, 0 },
    { 32435, 32723, 32768, 0 },
    { 32395, 32706, 32768, 0 } } },
    { { 27864, 31264, 32768, 0 },
    { 31905, 32658, 32768, 0 },
    { 32506, 32731, 32768, 0 },
    { 32618, 32730, 32768, 0 } } } },
  { { { 21525, 30206, 32768, 0 },
    { 32186, 32691, 32768, 0 },
    { 32572, 32724, 32768, 0 },
    { 32656, 32744, 32768, 0 } } },
    { { 27055, 30657, 32768, 0 },
    { 32025, 32586, 32768, 0 },
    { 31279, 32272, 32768, 0 },
    { 31858, 32313, 32768, 0 } } } },
  { { { 21773, 28489, 32768, 0 },
    { 32495, 32732, 32768, 0 },
    { 32332, 32728, 32768, 0 },
    { 30806, 32184, 32768, 0 } } },

```

```
{ { 27061, 30422, 32768, 0 },  
  { 32361, 32731, 32768, 0 },  
  { 21845, 27307, 32768, 0 },  
  { 31694, 32231, 32768, 0 } } },  
{ { { 12922, 23047, 32768, 0 },  
    { 31948, 32657, 32768, 0 },  
    { 32165, 32567, 32768, 0 },  
    { 31591, 32376, 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 ] = {
  { { { { 4462, 11148, 15537, 32768, 0 },
        { 18264, 30755, 32341, 32768, 0 },
        { 14122, 27788, 31245, 32768, 0 },
        { 9572, 22213, 28095, 32768, 0 },
        { 6129, 16137, 22879, 32768, 0 },
        { 2396, 6864, 10704, 32768, 0 },
        { 19658, 30736, 32079, 32768, 0 },
        { 14361, 27434, 30896, 32768, 0 },
        { 8665, 20636, 26666, 32768, 0 },
        { 5279, 14035, 20396, 32768, 0 },
        { 2691, 7559, 11944, 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 },
        { 16555, 28785, 31272, 32768, 0 },
        { 12438, 25266, 29768, 32768, 0 },
        { 5704, 14954, 21411, 32768, 0 },
        { 3851, 9759, 15119, 32768, 0 },
        { 2651, 6786, 10985, 32768, 0 },
        { 19197, 30991, 32196, 32768, 0 },
        { 15094, 28306, 31286, 32768, 0 },
        { 8944, 21144, 26942, 32768, 0 },
        { 4582, 12237, 18765, 32768, 0 },
        { 1710, 5074, 8401, 32768, 0 },
        { 22491, 31741, 32377, 32768, 0 },
        { 15856, 28468, 31274, 32768, 0 },
        { 8171, 19494, 25316, 32768, 0 },
        { 3625, 9950, 15314, 32768, 0 },
        { 1842, 5482, 9080, 32768, 0 },
        { 23278, 31180, 32046, 32768, 0 },
        { 15698, 27666, 30720, 32768, 0 },
        { 6556, 15447, 21080, 32768, 0 },
        { 2286, 6697, 11415, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 },
        { 8192, 16384, 24576, 32768, 0 } },
    { { { 8739, 19251, 23911, 32768, 0 },
        { 22607, 31690, 32533, 32768, 0 },
        { 16370, 29189, 31705, 32768, 0 },
        { 9688, 22103, 27905, 32768, 0 },
        { 5443, 14551, 21369, 32768, 0 },
        { 3070, 8678, 13719, 32768, 0 },
        { 24156, 31889, 32566, 32768, 0 },

```

```

{ 16734, 29422, 31842, 32768, 0 },
{ 9666, 22190, 28116, 32768, 0 },
{ 5574, 15559, 22623, 32768, 0 },
{ 3657, 10421, 16371, 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 },
{ 22073, 31374, 32456, 32768, 0 },
{ 16997, 29526, 31873, 32768, 0 },
{ 8534, 20607, 27087, 32768, 0 },
{ 4771, 14468, 21426, 32768, 0 },
{ 4089, 11614, 18725, 32768, 0 },
{ 25249, 32051, 32625, 32768, 0 },
{ 18701, 30256, 32113, 32768, 0 },
{ 10557, 23235, 28557, 32768, 0 },
{ 6309, 15377, 22265, 32768, 0 },
{ 3217, 9564, 15395, 32768, 0 },
{ 24835, 31937, 32612, 32768, 0 },
{ 18659, 30280, 32129, 32768, 0 },
{ 10696, 22992, 28323, 32768, 0 },
{ 6242, 15495, 22322, 32768, 0 },
{ 4446, 9569, 15369, 32768, 0 },
{ 24935, 31936, 32608, 32768, 0 },
{ 19146, 30559, 32249, 32768, 0 },
{ 8802, 20397, 26710, 32768, 0 },
{ 5435, 10094, 17393, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9856, 18284, 21827, 32768, 0 },
{ 26059, 32237, 32565, 32768, 0 },
{ 19806, 30095, 31706, 32768, 0 },
{ 11294, 23774, 28530, 32768, 0 },
{ 6440, 16494, 22901, 32768, 0 },
{ 2819, 7893, 12211, 32768, 0 },
{ 26592, 32578, 32720, 32768, 0 },
{ 22042, 31691, 32466, 32768, 0 },
{ 14456, 27549, 30942, 32768, 0 },
{ 8199, 19878, 26163, 32768, 0 },
{ 3415, 9266, 14132, 32768, 0 },
{ 23936, 31846, 32470, 32768, 0 },
{ 16040, 28366, 31207, 32768, 0 },
{ 9041, 20996, 26680, 32768, 0 },
{ 5275, 13918, 20301, 32768, 0 },

```

```

{ 2350, 6567, 10463, 32768, 0 },
{ 22539, 31631, 32400, 32768, 0 },
{ 16042, 28534, 31190, 32768, 0 },
{ 9241, 21034, 26794, 32768, 0 },
{ 5301, 14055, 20372, 32768, 0 },
{ 2391, 6563, 10442, 32768, 0 },
{ 25940, 32342, 32690, 32768, 0 },
{ 22414, 31695, 32502, 32768, 0 },
{ 15880, 28009, 31099, 32768, 0 },
{ 9070, 20468, 26695, 32768, 0 },
{ 4237, 11225, 16915, 32768, 0 },
{ 25588, 32380, 32632, 32768, 0 },
{ 19426, 30827, 32189, 32768, 0 },
{ 11665, 25052, 29617, 32768, 0 },
{ 6475, 16768, 23598, 32768, 0 },
{ 2744, 7688, 12023, 32768, 0 },
{ 27451, 32604, 32723, 32768, 0 },
{ 21118, 31472, 32401, 32768, 0 },
{ 12896, 26366, 30360, 32768, 0 },
{ 7372, 18709, 24965, 32768, 0 },
{ 3313, 9311, 14250, 32768, 0 },
{ 27040, 32497, 32670, 32768, 0 },
{ 20325, 31075, 32278, 32768, 0 },
{ 11948, 25084, 29512, 32768, 0 },
{ 7027, 17855, 24267, 32768, 0 },
{ 3489, 9731, 14876, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 11745, 23067, 26778, 32768, 0 },
{ 25071, 32147, 32596, 32768, 0 },
{ 19926, 30514, 31941, 32768, 0 },
{ 13206, 25647, 29530, 32768, 0 },
{ 7380, 17863, 24275, 32768, 0 },
{ 3289, 9424, 13962, 32768, 0 },
{ 26415, 32555, 32732, 32768, 0 },
{ 20972, 31488, 32486, 32768, 0 },
{ 13664, 26742, 30609, 32768, 0 },
{ 8351, 19980, 26188, 32768, 0 },
{ 4411, 11626, 17082, 32768, 0 },
{ 28142, 32455, 32681, 32768, 0 },
{ 19947, 30749, 32187, 32768, 0 },
{ 11516, 24090, 29048, 32768, 0 },
{ 5912, 15814, 22505, 32768, 0 },
{ 3284, 8872, 13758, 32768, 0 },
{ 24754, 32134, 32609, 32768, 0 },
{ 18838, 30537, 32187, 32768, 0 },
{ 12430, 25578, 30040, 32768, 0 },
{ 8208, 19343, 25605, 32768, 0 },
{ 4177, 10963, 16351, 32768, 0 },
{ 28000, 32534, 32721, 32768, 0 },
{ 21768, 31544, 32500, 32768, 0 },

```

```

{ 13635, 26637, 30601, 32768, 0 },
{ 7563, 19062, 25978, 32768, 0 },
{ 4390, 11935, 18307, 32768, 0 },
{ 25000, 32431, 32703, 32768, 0 },
{ 20932, 31526, 32518, 32768, 0 },
{ 14314, 27946, 31325, 32768, 0 },
{ 10851, 23360, 28921, 32768, 0 },
{ 5897, 15520, 22133, 32768, 0 },
{ 23474, 32064, 32692, 32768, 0 },
{ 20234, 31075, 32480, 32768, 0 },
{ 14033, 26970, 31249, 32768, 0 },
{ 12194, 24453, 29462, 32768, 0 },
{ 7420, 16583, 21962, 32768, 0 },
{ 22977, 31982, 32658, 32768, 0 },
{ 19435, 30913, 32456, 32768, 0 },
{ 11713, 25342, 30752, 32768, 0 },
{ 8749, 20236, 27921, 32768, 0 },
{ 4383, 13804, 20439, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 10755, 20127, 23881, 32768, 0 },
{ 24354, 31770, 32585, 32768, 0 },
{ 17309, 29579, 31791, 32768, 0 },
{ 11556, 24090, 28614, 32768, 0 },
{ 6193, 16236, 22637, 32768, 0 },
{ 2455, 6871, 10508, 32768, 0 },
{ 28585, 32627, 32733, 32768, 0 },
{ 21783, 31704, 32512, 32768, 0 },
{ 14206, 27735, 31142, 32768, 0 },
{ 8579, 20749, 26779, 32768, 0 },
{ 3344, 9122, 13657, 32768, 0 },
{ 29568, 32604, 32711, 32768, 0 },
{ 21312, 31184, 32268, 32768, 0 },
{ 12770, 25999, 30141, 32768, 0 },
{ 7652, 19125, 25430, 32768, 0 },
{ 3286, 9392, 14319, 32768, 0 },
{ 27758, 32539, 32700, 32768, 0 },
{ 20811, 31421, 32413, 32768, 0 },
{ 14047, 27753, 31196, 32768, 0 },
{ 9015, 21723, 27608, 32768, 0 },
{ 3879, 10482, 15686, 32768, 0 },
{ 29707, 32711, 32748, 32768, 0 },
{ 23194, 32086, 32635, 32768, 0 },
{ 15310, 29109, 31927, 32768, 0 },
{ 10241, 23995, 29594, 32768, 0 },
{ 5171, 13799, 19770, 32768, 0 },
{ 28022, 32588, 32705, 32768, 0 },
{ 20949, 31502, 32369, 32768, 0 },
{ 12876, 25966, 30093, 32768, 0 },
{ 7989, 19482, 25860, 32768, 0 },
{ 3673, 10233, 15494, 32768, 0 },

```



```

{ 29706, 32686, 32747, 32768, 0 },
{ 23091, 32054, 32605, 32768, 0 },
{ 14800, 28751, 31652, 32768, 0 },
{ 9850, 23230, 28722, 32768, 0 },
{ 4536, 12001, 17689, 32768, 0 },
{ 29400, 32698, 32748, 32768, 0 },
{ 23285, 32109, 32653, 32768, 0 },
{ 15287, 29288, 32016, 32768, 0 },
{ 10640, 24648, 29857, 32768, 0 },
{ 5243, 13730, 19559, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 9521, 21806, 26977, 32768, 0 },
{ 27936, 32497, 32698, 32768, 0 },
{ 18881, 30706, 32224, 32768, 0 },
{ 13866, 27418, 30762, 32768, 0 },
{ 9859, 22155, 26794, 32768, 0 },
{ 2721, 7754, 12153, 32768, 0 },
{ 28128, 32634, 32742, 32768, 0 },
{ 21600, 31722, 32542, 32768, 0 },
{ 15041, 28744, 31513, 32768, 0 },
{ 9793, 23312, 28915, 32768, 0 },
{ 4200, 10874, 15808, 32768, 0 },
{ 30338, 32581, 32730, 32768, 0 },
{ 19260, 30457, 32025, 32768, 0 },
{ 11154, 23294, 28333, 32768, 0 },
{ 6379, 16048, 22696, 32768, 0 },
{ 3584, 8558, 13751, 32768, 0 },
{ 28828, 32660, 32748, 32768, 0 },
{ 22611, 32073, 32640, 32768, 0 },
{ 18081, 30085, 32173, 32768, 0 },
{ 13957, 26831, 30432, 32768, 0 },
{ 10019, 20420, 24576, 32768, 0 },
{ 30576, 32671, 32748, 32768, 0 },
{ 22696, 31593, 32485, 32768, 0 },
{ 15674, 28435, 31466, 32768, 0 },
{ 11781, 24480, 29491, 32768, 0 },
{ 6809, 15880, 21918, 32768, 0 },
{ 21357, 31843, 32614, 32768, 0 },
{ 14847, 30213, 32361, 32768, 0 },
{ 13543, 28177, 31907, 32768, 0 },
{ 12702, 26785, 31111, 32768, 0 },
{ 7193, 16784, 24456, 32768, 0 },
{ 21005, 31928, 32348, 32768, 0 },
{ 19183, 30720, 32358, 32768, 0 },
{ 17324, 29421, 31652, 32768, 0 },
{ 13107, 27088, 30693, 32768, 0 },
{ 8001, 19813, 25338, 32768, 0 },
{ 19583, 31038, 32508, 32768, 0 },
{ 15796, 29040, 32034, 32768, 0 },
{ 11833, 25767, 31112, 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 } } } },
{ { { { 4124, 11059, 15633, 32768, 0 },
{ 18394, 30703, 32248, 32768, 0 },
{ 13398, 27103, 30987, 32768, 0 },
{ 8665, 20604, 26909, 32768, 0 },
{ 5309, 14230, 21005, 32768, 0 },
{ 2359, 6454, 10409, 32768, 0 },
{ 21026, 31227, 32281, 32768, 0 },
{ 14136, 27385, 31026, 32768, 0 },
{ 7712, 19722, 26214, 32768, 0 },
{ 4688, 13105, 19899, 32768, 0 },
{ 2726, 7811, 12613, 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 },
{ 17825, 29738, 31751, 32768, 0 },
{ 12572, 25682, 30276, 32768, 0 },
{ 5725, 15610, 22799, 32768, 0 },
{ 4377, 10950, 17689, 32768, 0 },
{ 3170, 8066, 12758, 32768, 0 },
{ 21492, 31555, 32375, 32768, 0 },
{ 15141, 28588, 31403, 32768, 0 },
{ 8790, 20480, 26487, 32768, 0 },
```

```

{ 3765, 10836, 17481, 32768, 0 },
{ 1671, 4757, 8206, 32768, 0 },
{ 24691, 32136, 32502, 32768, 0 },
{ 16369, 28517, 31232, 32768, 0 },
{ 8516, 19478, 25157, 32768, 0 },
{ 3340, 9884, 15638, 32768, 0 },
{ 2039, 5664, 9589, 32768, 0 },
{ 26997, 31917, 32381, 32768, 0 },
{ 17402, 28370, 31028, 32768, 0 },
{ 7668, 17256, 23130, 32768, 0 },
{ 2476, 7667, 13098, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 7807, 18182, 23504, 32768, 0 },
  { 22227, 31588, 32513, 32768, 0 },
  { 14304, 28166, 31501, 32768, 0 },
  { 8350, 20565, 27310, 32768, 0 },
  { 5048, 14009, 21308, 32768, 0 },
  { 3477, 9406, 15089, 32768, 0 },
  { 24745, 32070, 32638, 32768, 0 },
  { 15554, 29413, 32047, 32768, 0 },
  { 9228, 22656, 28919, 32768, 0 },
  { 5706, 16412, 24045, 32768, 0 },
  { 4691, 12165, 18561, 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 },
  { 23283, 31916, 32660, 32768, 0 },
  { 17021, 30258, 32297, 32768, 0 },
  { 10345, 23603, 29452, 32768, 0 },
  { 3968, 17792, 25088, 32768, 0 },
  { 2260, 12429, 24858, 32768, 0 },
  { 26285, 32108, 32658, 32768, 0 },
  { 18293, 29887, 32051, 32768, 0 },
  { 9282, 21866, 27783, 32768, 0 },
  { 4479, 12129, 20937, 32768, 0 },
  { 2453, 8830, 14226, 32768, 0 },
  { 25781, 32080, 32618, 32768, 0 },
  { 18779, 29929, 32165, 32768, 0 },
  { 10923, 22578, 27809, 32768, 0 },
  { 5712, 12927, 21044, 32768, 0 },
  { 3955, 12429, 18079, 32768, 0 },
  { 25820, 31951, 32604, 32768, 0 },

```

```

{ 19699, 30537, 32299, 32768, 0 },
{ 9862, 19724, 27360, 32768, 0 },
{ 4681, 9362, 14043, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 6566, 14440, 19033, 32768, 0 },
{ 22455, 31575, 32407, 32768, 0 },
{ 15831, 28483, 31329, 32768, 0 },
{ 9693, 22225, 27827, 32768, 0 },
{ 5703, 15240, 22125, 32768, 0 },
{ 2543, 7444, 11683, 32768, 0 },
{ 25817, 32497, 32694, 32768, 0 },
{ 19968, 31041, 32317, 32768, 0 },
{ 12383, 25712, 30150, 32768, 0 },
{ 6777, 17647, 24575, 32768, 0 },
{ 3137, 8905, 13866, 32768, 0 },
{ 24846, 31915, 32452, 32768, 0 },
{ 15632, 28234, 31232, 32768, 0 },
{ 8423, 20400, 26656, 32768, 0 },
{ 5175, 13906, 20512, 32768, 0 },
{ 2479, 7115, 11521, 32768, 0 },
{ 23211, 31770, 32458, 32768, 0 },
{ 15633, 28161, 31186, 32768, 0 },
{ 7969, 19692, 26070, 32768, 0 },
{ 4835, 13326, 20018, 32768, 0 },
{ 2515, 6981, 11176, 32768, 0 },
{ 26779, 32565, 32704, 32768, 0 },
{ 22101, 31714, 32494, 32768, 0 },
{ 14543, 27410, 30903, 32768, 0 },
{ 7657, 19345, 26054, 32768, 0 },
{ 3947, 11234, 17307, 32768, 0 },
{ 25261, 32068, 32576, 32768, 0 },
{ 17540, 29996, 31979, 32768, 0 },
{ 10592, 23809, 28945, 32768, 0 },
{ 5871, 15705, 22341, 32768, 0 },
{ 2602, 7262, 11582, 32768, 0 },
{ 28230, 32583, 32709, 32768, 0 },
{ 20517, 31175, 32331, 32768, 0 },
{ 12846, 26061, 30040, 32768, 0 },
{ 7118, 18243, 24556, 32768, 0 },
{ 3129, 9001, 14174, 32768, 0 },
{ 28850, 32619, 32713, 32768, 0 },
{ 20949, 31348, 32384, 32768, 0 },
{ 12391, 25575, 29806, 32768, 0 },
{ 6856, 17474, 23996, 32768, 0 },
{ 3312, 9642, 14989, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9723, 20276, 24593, 32768, 0 },
{ 24514, 31812, 32479, 32768, 0 },
{ 18048, 29628, 31519, 32768, 0 },

```

```

{ 11589, 23830, 28596, 32768, 0 },
{ 6357, 16468, 23195, 32768, 0 },
{ 3134, 8999, 13786, 32768, 0 },
{ 26258, 32407, 32712, 32768, 0 },
{ 19022, 30642, 32264, 32768, 0 },
{ 10810, 24078, 29283, 32768, 0 },
{ 5904, 16192, 23183, 32768, 0 },
{ 3581, 10547, 16335, 32768, 0 },
{ 27887, 32385, 32671, 32768, 0 },
{ 17809, 29882, 32038, 32768, 0 },
{ 9580, 21976, 27833, 32768, 0 },
{ 5880, 15087, 21588, 32768, 0 },
{ 3255, 8941, 14428, 32768, 0 },
{ 24471, 32058, 32603, 32768, 0 },
{ 17012, 29990, 32090, 32768, 0 },
{ 10112, 23199, 29008, 32768, 0 },
{ 5836, 16020, 23207, 32768, 0 },
{ 3503, 9946, 15663, 32768, 0 },
{ 28320, 32550, 32732, 32768, 0 },
{ 20066, 31287, 32503, 32768, 0 },
{ 11948, 25973, 30537, 32768, 0 },
{ 7074, 18828, 26476, 32768, 0 },
{ 4582, 12635, 19990, 32768, 0 },
{ 25121, 32151, 32666, 32768, 0 },
{ 18222, 30616, 32285, 32768, 0 },
{ 11542, 25440, 30517, 32768, 0 },
{ 8306, 20537, 26685, 32768, 0 },
{ 4800, 15445, 22124, 32768, 0 },
{ 22309, 31899, 32690, 32768, 0 },
{ 18121, 30818, 32484, 32768, 0 },
{ 12285, 26556, 30977, 32768, 0 },
{ 10198, 21734, 27920, 32768, 0 },
{ 4228, 13213, 20612, 32768, 0 },
{ 21103, 31836, 32696, 32768, 0 },
{ 18836, 31086, 32533, 32768, 0 },
{ 12013, 26071, 30877, 32768, 0 },
{ 8326, 17996, 26322, 32768, 0 },
{ 4599, 12072, 18396, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9550, 17624, 21538, 32768, 0 },
{ 24882, 31701, 32513, 32768, 0 },
{ 15356, 28381, 31130, 32768, 0 },
{ 9747, 21795, 27083, 32768, 0 },
{ 5300, 14553, 21009, 32768, 0 },
{ 2247, 6514, 10325, 32768, 0 },
{ 28009, 32592, 32719, 32768, 0 },
{ 20378, 31014, 32360, 32768, 0 },
{ 11639, 24941, 29911, 32768, 0 },
{ 6649, 17900, 24697, 32768, 0 },
{ 2812, 7993, 12458, 32768, 0 },

```



```

{ 29842, 32574, 32710, 32768, 0 },
{ 20150, 30644, 32145, 32768, 0 },
{ 11176, 23964, 28983, 32768, 0 },
{ 6463, 16904, 23591, 32768, 0 },
{ 2983, 8898, 13695, 32768, 0 },
{ 27690, 32557, 32697, 32768, 0 },
{ 21125, 31239, 32357, 32768, 0 },
{ 12439, 25456, 30046, 32768, 0 },
{ 6951, 18075, 24861, 32768, 0 },
{ 3043, 8862, 13681, 32768, 0 },
{ 30454, 32729, 32748, 32768, 0 },
{ 24373, 32272, 32661, 32768, 0 },
{ 15239, 28610, 31667, 32768, 0 },
{ 8752, 21867, 28297, 32768, 0 },
{ 4481, 12487, 18574, 32768, 0 },
{ 27738, 32478, 32675, 32768, 0 },
{ 19407, 30740, 32198, 32768, 0 },
{ 11116, 23705, 28702, 32768, 0 },
{ 5753, 15450, 21848, 32768, 0 },
{ 2846, 8210, 12891, 32768, 0 },
{ 29962, 32670, 32736, 32768, 0 },
{ 22991, 31878, 32529, 32768, 0 },
{ 14477, 27576, 30994, 32768, 0 },
{ 7999, 19669, 26297, 32768, 0 },
{ 3670, 10611, 16337, 32768, 0 },
{ 30784, 32735, 32748, 32768, 0 },
{ 24678, 32435, 32701, 32768, 0 },
{ 16718, 29725, 32014, 32768, 0 },
{ 9547, 22615, 28356, 32768, 0 },
{ 4355, 12438, 18375, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 7427, 17188, 21890, 32768, 0 },
  { 26814, 32222, 32608, 32768, 0 },
  { 16461, 28592, 31381, 32768, 0 },
  { 9778, 22398, 27133, 32768, 0 },
  { 5815, 14867, 20328, 32768, 0 },
  { 2178, 6347, 9744, 32768, 0 },
  { 28035, 32440, 32725, 32768, 0 },
  { 18970, 30651, 32293, 32768, 0 },
  { 11805, 25835, 29912, 32768, 0 },
  { 7871, 19918, 26147, 32768, 0 },
  { 3369, 9073, 13929, 32768, 0 },
  { 29403, 32382, 32627, 32768, 0 },
  { 15885, 28037, 31077, 32768, 0 },
  { 8006, 19228, 25221, 32768, 0 },
  { 4977, 13204, 19080, 32768, 0 },
  { 2887, 8504, 12795, 32768, 0 },
  { 28805, 32541, 32734, 32768, 0 },
  { 19886, 30860, 32365, 32768, 0 },
  { 13722, 26501, 30560, 32768, 0 },

```

```

{ 8960, 21018, 26432, 32768, 0 },
{ 3707, 10612, 16331, 32768, 0 },
{ 30766, 32712, 32748, 32768, 0 },
{ 21700, 31760, 32587, 32768, 0 },
{ 14844, 28410, 31602, 32768, 0 },
{ 8942, 21893, 27890, 32768, 0 },
{ 4622, 12781, 19033, 32768, 0 },
{ 24181, 32218, 32631, 32768, 0 },
{ 11995, 30876, 32087, 32768, 0 },
{ 13073, 27335, 30900, 32768, 0 },
{ 12156, 25897, 31711, 32768, 0 },
{ 8192, 15019, 20480, 32768, 0 },
{ 17695, 31457, 32113, 32768, 0 },
{ 15750, 30414, 32406, 32768, 0 },
{ 15278, 28546, 31160, 32768, 0 },
{ 10169, 27683, 30508, 32768, 0 },
{ 19661, 26214, 29491, 32768, 0 },
{ 20361, 31225, 32649, 32768, 0 },
{ 18018, 31367, 32631, 32768, 0 },
{ 13879, 28503, 32362, 32768, 0 },
{ 11108, 28325, 31380, 32768, 0 },
{ 10923, 26526, 31208, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 5556, 12459, 16671, 32768, 0 },
{ 19510, 28933, 30949, 32768, 0 },
{ 10782, 22318, 26991, 32768, 0 },
{ 6425, 17337, 23417, 32768, 0 },
{ 3353, 9760, 15789, 32768, 0 },
{ 1378, 3974, 6683, 32768, 0 },
{ 27885, 32206, 32661, 32768, 0 },
{ 16506, 28800, 31566, 32768, 0 },
{ 8880, 21100, 26867, 32768, 0 },
{ 4925, 14039, 20589, 32768, 0 },
{ 2003, 5812, 9458, 32768, 0 },
{ 29532, 32323, 32578, 32768, 0 },
{ 18719, 30254, 31943, 32768, 0 },
{ 10881, 23591, 28851, 32768, 0 },
{ 6147, 16595, 23036, 32768, 0 },
{ 2871, 7918, 12170, 32768, 0 },
{ 29145, 32568, 32707, 32768, 0 },
{ 20594, 31091, 32274, 32768, 0 },
{ 12739, 25878, 29947, 32768, 0 },
{ 7180, 17988, 24861, 32768, 0 },
{ 3177, 8838, 13418, 32768, 0 },
{ 31541, 32728, 32748, 32768, 0 },
{ 22662, 32095, 32666, 32768, 0 },
{ 15674, 29774, 32154, 32768, 0 },
{ 10785, 24760, 29924, 32768, 0 },
{ 4240, 11619, 16898, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },

```

[illegible]



[illegible]

```

{ 4623, 13399, 20891, 32768, 0 },
{ 3414, 9185, 14944, 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 },
{ 19898, 30891, 32341, 32768, 0 },
{ 13312, 27172, 31225, 32768, 0 },
{ 6872, 17894, 25643, 32768, 0 },
{ 6330, 12884, 20655, 32768, 0 },
{ 5925, 11003, 14873, 32768, 0 },
{ 23872, 31922, 32413, 32768, 0 },
{ 14591, 28022, 31251, 32768, 0 },
{ 7800, 18666, 25337, 32768, 0 },
{ 2933, 9493, 15953, 32768, 0 },
{ 1377, 4555, 8608, 32768, 0 },
{ 27231, 32205, 32509, 32768, 0 },
{ 16691, 27977, 31050, 32768, 0 },
{ 8922, 19309, 25077, 32768, 0 },
{ 3629, 11017, 16520, 32768, 0 },
{ 2392, 6578, 10763, 32768, 0 },
{ 30006, 32359, 32570, 32768, 0 },
{ 18797, 28390, 31221, 32768, 0 },
{ 8526, 19522, 25646, 32768, 0 },
{ 2365, 8783, 15877, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 7361, 18287, 24405, 32768, 0 },
{ 22334, 31620, 32570, 32768, 0 },
{ 13357, 28299, 31763, 32768, 0 },
{ 8540, 21286, 27979, 32768, 0 },
{ 5260, 15486, 23131, 32768, 0 },
{ 3197, 8743, 15040, 32768, 0 },
{ 25589, 32327, 32707, 32768, 0 },
{ 15905, 30232, 32380, 32768, 0 },
{ 9900, 24007, 30031, 32768, 0 },
{ 6115, 16462, 24641, 32768, 0 },
{ 4145, 10879, 18392, 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 },
{ 25799, 32564, 32748, 32768, 0 },
{ 18861, 31353, 32622, 32768, 0 },
{ 11786, 26422, 31473, 32768, 0 },
{ 8192, 21845, 27307, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 27314, 32139, 32687, 32768, 0 },
{ 17455, 29104, 31894, 32768, 0 },
{ 8392, 19021, 26134, 32768, 0 },
{ 7142, 13863, 19745, 32768, 0 },
{ 4274, 5699, 9973, 32768, 0 },
{ 26484, 31988, 32579, 32768, 0 },
{ 18920, 29829, 32084, 32768, 0 },
{ 11118, 22821, 29842, 32768, 0 },
{ 5783, 11565, 19275, 32768, 0 },
{ 7282, 18204, 25486, 32768, 0 },
{ 26862, 31879, 32514, 32768, 0 },
{ 19780, 30438, 32421, 32768, 0 },
{ 2731, 15019, 27307, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 5306, 13172, 18023, 32768, 0 },
{ 22174, 31768, 32544, 32768, 0 },
{ 14504, 28207, 31587, 32768, 0 },
{ 9063, 21603, 27790, 32768, 0 },
{ 5747, 15142, 22018, 32768, 0 },
{ 2665, 7998, 12578, 32768, 0 },
{ 25826, 32442, 32695, 32768, 0 },
{ 17050, 29997, 32168, 32768, 0 },
{ 9931, 23538, 29315, 32768, 0 },
{ 6323, 16989, 24177, 32768, 0 },
{ 3544, 10012, 15786, 32768, 0 },
{ 24636, 31907, 32551, 32768, 0 },
{ 14485, 27834, 31354, 32768, 0 },
{ 8382, 20539, 26960, 32768, 0 },
{ 5196, 14452, 21368, 32768, 0 },
{ 2976, 8504, 13546, 32768, 0 },
{ 23187, 31744, 32515, 32768, 0 },
{ 14608, 27811, 31297, 32768, 0 },
{ 7810, 20056, 26742, 32768, 0 },
{ 5206, 14392, 21258, 32768, 0 },
{ 3037, 8791, 13853, 32768, 0 },
{ 27745, 32568, 32716, 32768, 0 },
{ 19537, 31025, 32427, 32768, 0 },
{ 11275, 25278, 30482, 32768, 0 },
{ 7104, 18978, 26240, 32768, 0 },

```

```

{ 4434, 12675, 19421, 32768, 0 },
{ 26891, 32123, 32590, 32768, 0 },
{ 16499, 29303, 31734, 32768, 0 },
{ 9128, 21581, 27348, 32768, 0 },
{ 4831, 13689, 20215, 32768, 0 },
{ 2299, 6583, 10905, 32768, 0 },
{ 29253, 32565, 32702, 32768, 0 },
{ 19375, 30467, 32129, 32768, 0 },
{ 11245, 24302, 29078, 32768, 0 },
{ 6046, 16407, 23539, 32768, 0 },
{ 3088, 9440, 15273, 32768, 0 },
{ 30400, 32645, 32733, 32768, 0 },
{ 20688, 30891, 32312, 32768, 0 },
{ 11007, 24065, 29305, 32768, 0 },
{ 6289, 17016, 24026, 32768, 0 },
{ 3628, 10533, 16695, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 8646, 19108, 23905, 32768, 0 },
  { 23739, 31943, 32479, 32768, 0 },
  { 16012, 29063, 31413, 32768, 0 },
  { 11382, 23805, 28343, 32768, 0 },
  { 5585, 15795, 22502, 32768, 0 },
  { 4021, 10454, 16552, 32768, 0 },
  { 25774, 32257, 32715, 32768, 0 },
  { 17392, 30186, 32209, 32768, 0 },
  { 9999, 23407, 29063, 32768, 0 },
  { 5889, 16415, 23982, 32768, 0 },
  { 3924, 11892, 18535, 32768, 0 },
  { 28039, 32395, 32654, 32768, 0 },
  { 15958, 29493, 31882, 32768, 0 },
  { 10150, 22545, 28351, 32768, 0 },
  { 6983, 16262, 22806, 32768, 0 },
  { 4736, 10923, 17689, 32768, 0 },
  { 25247, 32153, 32644, 32768, 0 },
  { 16540, 30064, 32169, 32768, 0 },
  { 9724, 23041, 28984, 32768, 0 },
  { 6087, 16012, 23510, 32768, 0 },
  { 3636, 11238, 18465, 32768, 0 },
  { 28641, 32636, 32745, 32768, 0 },
  { 19619, 31455, 32600, 32768, 0 },
  { 11576, 26365, 31068, 32768, 0 },
  { 6688, 19832, 28387, 32768, 0 },
  { 6425, 14537, 22327, 32768, 0 },
  { 24312, 32051, 32684, 32768, 0 },
  { 17971, 29948, 32148, 32768, 0 },
  { 11327, 23868, 29329, 32768, 0 },
  { 5174, 13797, 20696, 32768, 0 },
  { 3277, 13107, 22938, 32768, 0 },
  { 23396, 31785, 32571, 32768, 0 },
  { 19640, 31350, 32348, 32768, 0 },

```



```

{ 14043, 19505, 26526, 32768, 0 },
{ 4681, 14043, 18725, 32768, 0 },
{ 14043, 23406, 28087, 32768, 0 },
{ 22952, 32046, 32624, 32768, 0 },
{ 18053, 30666, 32521, 32768, 0 },
{ 14336, 24576, 28672, 32768, 0 },
{ 4681, 14043, 18725, 32768, 0 },
{ 6554, 13107, 19661, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 6447, 14363, 18951, 32768, 0 },
{ 23736, 31444, 32532, 32768, 0 },
{ 13480, 27014, 30790, 32768, 0 },
{ 8395, 20532, 26629, 32768, 0 },
{ 5255, 14331, 21040, 32768, 0 },
{ 2289, 7037, 11241, 32768, 0 },
{ 27036, 32527, 32712, 32768, 0 },
{ 17223, 30108, 32192, 32768, 0 },
{ 10225, 23887, 29577, 32768, 0 },
{ 6309, 17098, 24150, 32768, 0 },
{ 2919, 8483, 13233, 32768, 0 },
{ 29436, 32530, 32712, 32768, 0 },
{ 18119, 29975, 32059, 32768, 0 },
{ 10039, 22952, 28783, 32768, 0 },
{ 6075, 16199, 23020, 32768, 0 },
{ 3176, 9307, 14620, 32768, 0 },
{ 26945, 32388, 32672, 32768, 0 },
{ 17686, 29999, 32134, 32768, 0 },
{ 10104, 23162, 29005, 32768, 0 },
{ 6120, 16494, 23597, 32768, 0 },
{ 2941, 8550, 13739, 32768, 0 },
{ 30723, 32723, 32748, 32768, 0 },
{ 23261, 32041, 32636, 32768, 0 },
{ 13607, 27654, 31409, 32768, 0 },
{ 7944, 20700, 27589, 32768, 0 },
{ 4415, 12702, 19331, 32768, 0 },
{ 27993, 32319, 32650, 32768, 0 },
{ 16599, 29380, 31793, 32768, 0 },
{ 8660, 20044, 26077, 32768, 0 },
{ 4517, 12317, 18513, 32768, 0 },
{ 2357, 6808, 11259, 32768, 0 },
{ 30290, 32609, 32712, 32768, 0 },
{ 21059, 31180, 32308, 32768, 0 },
{ 11912, 24987, 29897, 32768, 0 },
{ 6737, 17747, 25009, 32768, 0 },
{ 3386, 10886, 16566, 32768, 0 },
{ 31525, 32732, 32748, 32768, 0 },
{ 24173, 32214, 32664, 32768, 0 },
{ 14606, 27824, 31289, 32768, 0 },
{ 7942, 19819, 26273, 32768, 0 },
{ 4666, 12492, 18643, 32768, 0 },

```

```

    { 8192, 16384, 24576, 32768, 0 } },
  { { 6767, 15371, 20330, 32768, 0 },
    { 25254, 31761, 32478, 32768, 0 },
    { 14651, 27133, 30714, 32768, 0 },
    { 8759, 19519, 25210, 32768, 0 },
    { 4575, 12767, 17980, 32768, 0 },
    { 2621, 7677, 12639, 32768, 0 },
    { 27714, 32397, 32714, 32768, 0 },
    { 18103, 30373, 32299, 32768, 0 },
    { 12109, 25700, 29566, 32768, 0 },
    { 7079, 18230, 24357, 32768, 0 },
    { 3549, 10632, 15972, 32768, 0 },
    { 28603, 32338, 32573, 32768, 0 },
    { 14501, 27887, 30936, 32768, 0 },
    { 7303, 18808, 25077, 32768, 0 },
    { 6311, 14321, 20146, 32768, 0 },
    { 3523, 8456, 15151, 32768, 0 },
    { 28690, 32538, 32731, 32768, 0 },
    { 18322, 30307, 32174, 32768, 0 },
    { 12254, 24410, 29125, 32768, 0 },
    { 6788, 16946, 24027, 32768, 0 },
    { 3495, 10962, 17834, 32768, 0 },
    { 30608, 32739, 32748, 32768, 0 },
    { 21866, 31864, 32610, 32768, 0 },
    { 14076, 27533, 31249, 32768, 0 },
    { 7955, 20570, 27356, 32768, 0 },
    { 5068, 14634, 21751, 32768, 0 },
    { 25372, 31738, 32487, 32768, 0 },
    { 15534, 31068, 32150, 32768, 0 },
    { 7123, 14247, 18521, 32768, 0 },
    { 6554, 13107, 26214, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 21845, 28399, 30583, 32768, 0 },
    { 6951, 29789, 31775, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 10923, 21845, 27307, 32768, 0 },
    { 13107, 19661, 26214, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 },
    { 8192, 16384, 24576, 32768, 0 } } },
  { { { 4381, 10541, 14753, 32768, 0 },
    { 22048, 30479, 32313, 32768, 0 },
    { 11535, 24774, 29392, 32768, 0 },
    { 5954, 16992, 24035, 32768, 0 },
    { 4190, 11244, 17334, 32768, 0 },
    { 1780, 5000, 8139, 32768, 0 },
    { 25953, 32034, 32608, 32768, 0 },

```







[illegible]

```

{ 2427, 10923, 15777, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 7947, 20544, 26943, 32768, 0 },
{ 24652, 32335, 32654, 32768, 0 },
{ 14826, 29854, 32210, 32768, 0 },
{ 8809, 22288, 28971, 32768, 0 },
{ 5767, 15204, 23069, 32768, 0 },
{ 1365, 6827, 17749, 32768, 0 },
{ 27298, 32534, 32743, 32768, 0 },
{ 17987, 31415, 32629, 32768, 0 },
{ 12542, 25536, 30960, 32768, 0 },
{ 2731, 16384, 27307, 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 },
{ 27922, 32576, 32720, 32768, 0 },
{ 24881, 32435, 32657, 32768, 0 },
{ 6554, 19661, 26214, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 28613, 32174, 32649, 32768, 0 },
{ 16794, 28570, 32154, 32768, 0 },
{ 12288, 20480, 26624, 32768, 0 },
{ 16384, 21845, 27307, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 24030, 31130, 32222, 32768, 0 },
{ 20194, 30101, 32006, 32768, 0 },
{ 4681, 18725, 23406, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 22528, 28672, 30720, 32768, 0 },
{ 21845, 29127, 30948, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 5773, 14964, 20566, 32768, 0 },
{ 23520, 31988, 32666, 32768, 0 },
{ 14718, 28826, 31986, 32768, 0 },
{ 9050, 22915, 28999, 32768, 0 },
{ 6444, 16812, 23966, 32768, 0 },

```

```

{ 3596, 10705, 16731, 32768, 0 },
{ 25940, 32530, 32730, 32768, 0 },
{ 16613, 30412, 32406, 32768, 0 },
{ 10151, 24748, 30342, 32768, 0 },
{ 6936, 18162, 25694, 32768, 0 },
{ 4177, 12450, 19478, 32768, 0 },
{ 25578, 32184, 32670, 32768, 0 },
{ 15022, 28821, 31899, 32768, 0 },
{ 8460, 21731, 28482, 32768, 0 },
{ 5955, 15692, 22628, 32768, 0 },
{ 3501, 10096, 15945, 32768, 0 },
{ 25088, 32313, 32680, 32768, 0 },
{ 15564, 29262, 32026, 32768, 0 },
{ 9045, 22016, 28644, 32768, 0 },
{ 6182, 16554, 23714, 32768, 0 },
{ 3871, 11212, 17086, 32768, 0 },
{ 28257, 32655, 32746, 32768, 0 },
{ 18813, 31355, 32579, 32768, 0 },
{ 11383, 26157, 31207, 32768, 0 },
{ 7845, 20824, 28078, 32768, 0 },
{ 5232, 14795, 22747, 32768, 0 },
{ 28556, 32271, 32604, 32768, 0 },
{ 15888, 29004, 31684, 32768, 0 },
{ 8252, 20004, 26557, 32768, 0 },
{ 4242, 13050, 20055, 32768, 0 },
{ 1897, 6457, 11138, 32768, 0 },
{ 30420, 32590, 32725, 32768, 0 },
{ 18867, 30189, 32183, 32768, 0 },
{ 10567, 23955, 29534, 32768, 0 },
{ 6295, 17592, 25180, 32768, 0 },
{ 4707, 12220, 18873, 32768, 0 },
{ 31372, 32677, 32748, 32768, 0 },
{ 20156, 30965, 32445, 32768, 0 },
{ 10759, 24575, 30202, 32768, 0 },
{ 6213, 18391, 25613, 32768, 0 },
{ 5260, 13498, 20294, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { 7910, 18394, 24609, 32768, 0 },
{ 25137, 32189, 32696, 32768, 0 },
{ 15399, 29633, 32019, 32768, 0 },
{ 11449, 23688, 29807, 32768, 0 },
{ 3171, 16913, 23255, 32768, 0 },
{ 9638, 11565, 21203, 32768, 0 },
{ 28689, 32605, 32698, 32768, 0 },
{ 18748, 31201, 32380, 32768, 0 },
{ 9854, 26594, 31106, 32768, 0 },
{ 6275, 18127, 25099, 32768, 0 },
{ 3641, 10923, 23666, 32768, 0 },
{ 28173, 32548, 32727, 32768, 0 },
{ 17401, 30330, 32127, 32768, 0 },

```



```

{ 12663, 25228, 29868, 32768, 0 },
{ 12208, 25058, 28913, 32768, 0 },
{ 7123, 19946, 25645, 32768, 0 },
{ 28164, 32525, 32717, 32768, 0 },
{ 17988, 30216, 32294, 32768, 0 },
{ 11252, 24483, 30295, 32768, 0 },
{ 6199, 15941, 23912, 32768, 0 },
{ 4096, 16384, 24576, 32768, 0 },
{ 30688, 32728, 32748, 32768, 0 },
{ 21740, 32239, 32715, 32768, 0 },
{ 16132, 26719, 30752, 32768, 0 },
{ 7282, 18204, 25486, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 27982, 31295, 32400, 32768, 0 },
{ 16636, 30499, 32012, 32768, 0 },
{ 4681, 9362, 18725, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 18725, 28087, 30427, 32768, 0 },
{ 17644, 27727, 30247, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 16384, 21845, 27307, 32768, 0 },
{ 13107, 19661, 26214, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } },
{ { { 6447, 15470, 20891, 32768, 0 },
{ 23739, 31792, 32623, 32768, 0 },
{ 13987, 27588, 31524, 32768, 0 },
{ 7628, 20267, 26781, 32768, 0 },
{ 5218, 14836, 21280, 32768, 0 },
{ 2506, 7741, 13200, 32768, 0 },
{ 27828, 32533, 32744, 32768, 0 },
{ 17675, 30580, 32387, 32768, 0 },
{ 10378, 24402, 30056, 32768, 0 },
{ 7110, 18752, 25433, 32768, 0 },
{ 4204, 11816, 17754, 32768, 0 },
{ 29255, 32596, 32746, 32768, 0 },
{ 17208, 30043, 32288, 32768, 0 },
{ 10079, 23767, 29801, 32768, 0 },
{ 6547, 17026, 24483, 32768, 0 },
{ 3610, 11158, 16761, 32768, 0 },
{ 28656, 32539, 32743, 32768, 0 },
{ 17009, 30249, 32364, 32768, 0 },
{ 10196, 23531, 29486, 32768, 0 },
{ 6325, 17030, 24183, 32768, 0 },
{ 3724, 10255, 16668, 32768, 0 },

```

```

{ 30777, 32733, 32748, 32768, 0 },
{ 21665, 32002, 32651, 32768, 0 },
{ 13584, 28143, 31783, 32768, 0 },
{ 8635, 22029, 28686, 32768, 0 },
{ 5640, 16027, 23429, 32768, 0 },
{ 29265, 32394, 32668, 32768, 0 },
{ 15949, 28837, 31668, 32768, 0 },
{ 8609, 20421, 26119, 32768, 0 },
{ 4078, 11724, 17986, 32768, 0 },
{ 2264, 7149, 12035, 32768, 0 },
{ 31061, 32665, 32737, 32768, 0 },
{ 21073, 31231, 32395, 32768, 0 },
{ 12636, 25743, 29942, 32768, 0 },
{ 6090, 17544, 25953, 32768, 0 },
{ 3855, 11290, 18449, 32768, 0 },
{ 31680, 32745, 32749, 32768, 0 },
{ 22609, 31806, 32593, 32768, 0 },
{ 13415, 26899, 31068, 32768, 0 },
{ 5221, 20216, 25215, 32768, 0 },
{ 5041, 14564, 22125, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ { 5079, 14099, 20130, 32768, 0 },
{ 24576, 31177, 32414, 32768, 0 },
{ 11252, 26584, 30406, 32768, 0 },
{ 7509, 17067, 26169, 32768, 0 },
{ 5334, 14479, 22099, 32768, 0 },
{ 6242, 12483, 18725, 32768, 0 },
{ 28608, 32613, 32716, 32768, 0 },
{ 15918, 30477, 32535, 32768, 0 },
{ 7232, 23051, 30734, 32768, 0 },
{ 5120, 14336, 25600, 32768, 0 },
{ 5958, 14895, 20852, 32768, 0 },
{ 29156, 32252, 32621, 32768, 0 },
{ 15560, 28153, 31449, 32768, 0 },
{ 9242, 22686, 27727, 32768, 0 },
{ 4681, 14043, 23406, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 28863, 32306, 32697, 32768, 0 },
{ 15547, 29563, 31811, 32768, 0 },
{ 9125, 21154, 26961, 32768, 0 },
{ 5699, 14247, 21370, 32768, 0 },
{ 6554, 19661, 26214, 32768, 0 },
{ 32103, 32748, 32752, 32768, 0 },
{ 23758, 32223, 32744, 32768, 0 },
{ 16193, 30101, 32577, 32768, 0 },
{ 14564, 27307, 30948, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 18432, 28672, 30720, 32768, 0 },
{ 10128, 30385, 32172, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },

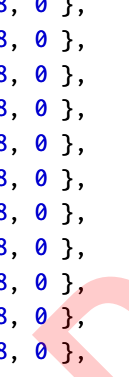
```

[illegible]



```
, 31843, 32768, 0 },  
    , 27918, 32768, 0 },  
    , 23453, 32768, 0 },  
    13602, 32768, 0 },  
    , 32680, 32768, 0 },  
    , 32335, 32768, 0 },  
    , 29660, 32768, 0 },  
    , 24835, 32768, 0 },  
    , 18525, 32768, 0 },  
    , 32749, 32768, 0 },  
    , 32604, 32768, 0 },  
    , 31206, 32768, 0 },  
    , 27937, 32768, 0 },  
    , 20751, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 },  
    , 24576, 32768, 0 } },  
    , 24576, 32768, 0 },  
    , 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 } }



```

Default_Coeff_Br_Cdf[ COEFF_CDF_Q_CTXS ][ TX_SIZES ][ PLANE_TYPES ][ LEVEL_CONTEXTS ][ BR_CDF_SIZE +
1 ] = {
  { { { { 19551, 26389, 29394, 32768, 0 },
        { 15086, 22440, 26364, 32768, 0 },
        { 9867, 16572, 21009, 32768, 0 },
        { 6574, 12039, 16670, 32768, 0 },
        { 4817, 9114, 12770, 32768, 0 },
        { 3666, 7177, 10609, 32768, 0 },
        { 1746, 3398, 5058, 32768, 0 },
        { 22294, 28462, 30592, 32768, 0 },
        { 18791, 26002, 29147, 32768, 0 },
        { 13982, 21661, 25851, 32768, 0 },
        { 10125, 17144, 21891, 32768, 0 },
        { 7478, 13554, 18127, 32768, 0 },
        { 5838, 10966, 15307, 32768, 0 },
        { 3097, 5933, 8647, 32768, 0 },
        { 17134, 23834, 26964, 32768, 0 },
        { 15992, 23132, 26695, 32768, 0 },
        { 12546, 19975, 24251, 32768, 0 },
        { 9750, 16337, 21035, 32768, 0 },
        { 7560, 13564, 18188, 32768, 0 },
        { 6154, 11431, 15727, 32768, 0 },
        { 3699, 7221, 10274, 32768, 0 } },
    { { { 18428, 24992, 28179, 32768, 0 },
        { 14152, 21273, 25185, 32768, 0 },
        { 9541, 16148, 20496, 32768, 0 },
        { 6782, 12221, 16530, 32768, 0 },
        { 5190, 9734, 13857, 32768, 0 },
        { 4244, 8233, 11808, 32768, 0 },
        { 2563, 5107, 7516, 32768, 0 },
        { 24616, 29743, 31576, 32768, 0 },
        { 20086, 27180, 30135, 32768, 0 },
        { 14955, 23030, 27367, 32768, 0 },
        { 11263, 18700, 23866, 32768, 0 },
        { 8653, 15468, 20410, 32768, 0 },
        { 7365, 13249, 18188, 32768, 0 },
        { 4647, 8900, 12706, 32768, 0 },
        { 23742, 29468, 31373, 32768, 0 },
        { 21826, 28503, 30879, 32768, 0 },
        { 16900, 25099, 28993, 32768, 0 },
        { 12951, 21209, 25853, 32768, 0 },
        { 10994, 18168, 23086, 32768, 0 },
        { 8611, 15481, 20320, 32768, 0 },
        { 6804, 11905, 16212, 32768, 0 } } },
    { { { 18147, 24912, 28104, 32768, 0 },
        { 13981, 21382, 25277, 32768, 0 },
        { 9783, 16554, 21286, 32768, 0 },
        { 6894, 12423, 16934, 32768, 0 },
        { 5308, 9739, 13703, 32768, 0 },
        { 3879, 7778, 11189, 32768, 0 },

```

```

{ 1754, 3458, 5126, 32768, 0 },
{ 18387, 24745, 27571, 32768, 0 },
{ 16925, 24036, 27536, 32768, 0 },
{ 13403, 20827, 24997, 32768, 0 },
{ 10016, 16766, 21323, 32768, 0 },
{ 7571, 13492, 17918, 32768, 0 },
{ 6033, 11142, 15323, 32768, 0 },
{ 2946, 5727, 8243, 32768, 0 },
{ 21853, 27668, 30027, 32768, 0 },
{ 19575, 26398, 29328, 32768, 0 },
{ 15913, 23532, 27355, 32768, 0 },
{ 12062, 19606, 24209, 32768, 0 },
{ 9277, 16022, 20814, 32768, 0 },
{ 7565, 13521, 18188, 32768, 0 },
{ 4145, 7899, 11241, 32768, 0 },
{ { 18987, 25320, 28207, 32768, 0 },
  { 14495, 21416, 24953, 32768, 0 },
  { 9644, 16089, 20432, 32768, 0 },
  { 6209, 11539, 15700, 32768, 0 },
  { 4614, 8897, 12620, 32768, 0 },
  { 3707, 7058, 10428, 32768, 0 },
  { 2133, 4370, 6280, 32768, 0 },
  { 20230, 25658, 28285, 32768, 0 },
  { 18809, 25538, 28589, 32768, 0 },
  { 15223, 22386, 26228, 32768, 0 },
  { 11028, 18262, 22688, 32768, 0 },
  { 8057, 14435, 19260, 32768, 0 },
  { 6957, 12514, 16745, 32768, 0 },
  { 4547, 8769, 12279, 32768, 0 },
  { 24638, 30008, 31594, 32768, 0 },
  { 22399, 28808, 31134, 32768, 0 },
  { 17539, 25444, 29031, 32768, 0 },
  { 12927, 20996, 25632, 32768, 0 },
  { 10192, 17374, 22231, 32768, 0 },
  { 8456, 14966, 20063, 32768, 0 },
  { 5218, 10245, 14469, 32768, 0 } } },
{ { { 16392, 22702, 25958, 32768, 0 },
    { 12223, 18336, 22043, 32768, 0 },
    { 7223, 12257, 16027, 32768, 0 },
    { 5147, 9350, 12639, 32768, 0 },
    { 4001, 7730, 10664, 32768, 0 },
    { 3231, 6521, 9334, 32768, 0 },
    { 1401, 2882, 4279, 32768, 0 },
    { 18215, 24183, 27317, 32768, 0 },
    { 17044, 24122, 27485, 32768, 0 },
    { 13672, 21231, 25203, 32768, 0 },
    { 10558, 17393, 21747, 32768, 0 },
    { 8040, 13822, 18467, 32768, 0 },
    { 6203, 11237, 15630, 32768, 0 },
    { 2705, 5161, 7397, 32768, 0 },

```



```

{ 23884, 28581, 30490, 32768, 0 },
{ 22647, 28396, 30485, 32768, 0 },
{ 18533, 25799, 28974, 32768, 0 },
{ 13947, 21648, 25883, 32768, 0 },
{ 10384, 17675, 22355, 32768, 0 },
{ 8315, 14706, 19350, 32768, 0 },
{ 4020, 7618, 10742, 32768, 0 },
{ { 18997, 24785, 27601, 32768, 0 },
  { 12268, 18352, 21872, 32768, 0 },
  { 7051, 12368, 15536, 32768, 0 },
  { 5731, 9979, 13350, 32768, 0 },
  { 4499, 7630, 10368, 32768, 0 },
  { 3875, 5757, 8524, 32768, 0 },
  { 1056, 2589, 3853, 32768, 0 },
  { 17554, 20285, 23016, 32768, 0 },
  { 15322, 22897, 26226, 32768, 0 },
  { 10940, 18004, 21987, 32768, 0 },
  { 9362, 15739, 19471, 32768, 0 },
  { 6741, 11947, 16555, 32768, 0 },
  { 5928, 10209, 15149, 32768, 0 },
  { 2893, 5825, 8187, 32768, 0 },
  { 25166, 30849, 31919, 32768, 0 },
  { 21563, 28560, 30954, 32768, 0 },
  { 18615, 26000, 29479, 32768, 0 },
  { 15017, 22815, 26675, 32768, 0 },
  { 11993, 19544, 23987, 32768, 0 },
  { 9525, 15728, 20908, 32768, 0 },
  { 6177, 11184, 15150, 32768, 0 } } },
{ { { 10837, 15701, 19712, 32768, 0 },
    { 9385, 14893, 18343, 32768, 0 },
    { 7579, 13650, 17912, 32768, 0 },
    { 5988, 11782, 16045, 32768, 0 },
    { 4751, 8271, 11819, 32768, 0 },
    { 2707, 6174, 9688, 32768, 0 },
    { 1305, 2512, 3937, 32768, 0 },
    { 10791, 22743, 28173, 32768, 0 },
    { 9839, 17031, 22109, 32768, 0 },
    { 8967, 15846, 19602, 32768, 0 },
    { 7062, 12429, 17231, 32768, 0 },
    { 5855, 10401, 14220, 32768, 0 },
    { 4695, 8820, 13562, 32768, 0 },
    { 2180, 4064, 5805, 32768, 0 },
    { 17254, 21692, 24122, 32768, 0 },
    { 20856, 26923, 29298, 32768, 0 },
    { 18254, 24995, 28332, 32768, 0 },
    { 13954, 21179, 25354, 32768, 0 },
    { 10291, 17387, 21898, 32768, 0 },
    { 8179, 14580, 19018, 32768, 0 },
    { 3337, 6302, 9006, 32768, 0 } } },
{ { 14532, 19661, 23935, 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 } } },
{ { { { 19774, 26358, 29082, 32768, 0 },
{ 14793, 22128, 26184, 32768, 0 },
{ 8992, 15426, 19832, 32768, 0 },
{ 5843, 10899, 15174, 32768, 0 },
{ 4555, 8453, 12187, 32768, 0 },
{ 3314, 6670, 10157, 32768, 0 },
{ 1952, 3884, 5849, 32768, 0 },
{ 22545, 28223, 30308, 32768, 0 },
{ 19259, 26622, 29670, 32768, 0 },
{ 13873, 21713, 25867, 32768, 0 },
{ 10038, 17342, 22311, 32768, 0 },
{ 7772, 13947, 18771, 32768, 0 },
{ 6253, 11629, 16157, 32768, 0 },
{ 3854, 7371, 10782, 32768, 0 },
{ 20735, 27052, 29706, 32768, 0 },
{ 18100, 25463, 28757, 32768, 0 },
{ 14127, 22191, 26638, 32768, 0 },
{ 10740, 18111, 23164, 32768, 0 },
{ 8260, 15001, 20096, 32768, 0 },
{ 7063, 13121, 17826, 32768, 0 },
{ 4926, 9401, 13266, 32768, 0 } },
{ { 18656, 25450, 28563, 32768, 0 },
{ 14172, 21601, 25681, 32768, 0 },
{ 9863, 16732, 21478, 32768, 0 },
{ 7011, 12934, 17759, 32768, 0 },
{ 5450, 10634, 14969, 32768, 0 },
{ 4454, 8494, 12264, 32768, 0 },
{ 2820, 5975, 8764, 32768, 0 },
{ 25785, 30620, 32085, 32768, 0 },
{ 21551, 28552, 31100, 32768, 0 },
{ 16467, 24836, 28985, 32768, 0 },
{ 12615, 20070, 25452, 32768, 0 },
{ 9175, 16469, 21887, 32768, 0 },
{ 8129, 14184, 19537, 32768, 0 },
{ 5565, 10857, 15333, 32768, 0 },
{ 26325, 30916, 31855, 32768, 0 },
{ 24536, 30416, 31975, 32768, 0 },
{ 19786, 27515, 30609, 32768, 0 },

```

```

    { 14246, 23341, 27922, 32768, 0 },
    { 11548, 19018, 24504, 32768, 0 },
    { 10118, 18051, 22765, 32768, 0 },
    { 6683, 13150, 19187, 32768, 0 } } },
{ { { 18898, 25448, 28561, 32768, 0 },
    { 14653, 22247, 26134, 32768, 0 },
    { 9800, 16705, 21415, 32768, 0 },
    { 6649, 12024, 16547, 32768, 0 },
    { 4799, 9158, 12993, 32768, 0 },
    { 3652, 7164, 10409, 32768, 0 },
    { 1788, 3504, 5273, 32768, 0 },
    { 19949, 26182, 28667, 32768, 0 },
    { 17762, 24858, 28208, 32768, 0 },
    { 13405, 20936, 25298, 32768, 0 },
    { 9720, 16629, 21324, 32768, 0 },
    { 7426, 13427, 17962, 32768, 0 },
    { 6115, 11311, 15573, 32768, 0 },
    { 3493, 6708, 9673, 32768, 0 },
    { 23508, 29198, 31135, 32768, 0 },
    { 21007, 27758, 30388, 32768, 0 },
    { 16514, 24271, 28148, 32768, 0 },
    { 12383, 20230, 24979, 32768, 0 },
    { 9679, 16637, 21660, 32768, 0 },
    { 7994, 14236, 19060, 32768, 0 },
    { 5020, 9445, 13404, 32768, 0 } } },
{ { { 18592, 24875, 28038, 32768, 0 },
    { 13893, 20779, 24708, 32768, 0 },
    { 9044, 15352, 19728, 32768, 0 },
    { 6146, 11585, 15632, 32768, 0 },
    { 4577, 8600, 12564, 32768, 0 },
    { 3854, 7249, 10820, 32768, 0 },
    { 2818, 5720, 8077, 32768, 0 },
    { 20716, 26285, 29153, 32768, 0 },
    { 18494, 25251, 28653, 32768, 0 },
    { 14384, 22310, 26560, 32768, 0 },
    { 11169, 18667, 23569, 32768, 0 },
    { 8587, 15606, 21134, 32768, 0 },
    { 7356, 13550, 18794, 32768, 0 },
    { 6277, 11555, 15702, 32768, 0 },
    { 25791, 30887, 32235, 32768, 0 },
    { 23193, 29469, 31681, 32768, 0 },
    { 18326, 26014, 29630, 32768, 0 },
    { 13377, 22009, 26765, 32768, 0 },
    { 11252, 19231, 24410, 32768, 0 },
    { 8855, 16098, 21505, 32768, 0 },
    { 6382, 12705, 17760, 32768, 0 } } } },
{ { { 13784, 20459, 23807, 32768, 0 },
    { 11188, 17339, 21036, 32768, 0 },
    { 7724, 13275, 17419, 32768, 0 },
    { 5328, 10198, 13773, 32768, 0 },

```

```

{ 4095, 7965, 11303, 32768, 0 },
{ 3392, 6600, 9642, 32768, 0 },
{ 1449, 2940, 4455, 32768, 0 },
{ 19327, 25497, 28310, 32768, 0 },
{ 17553, 24612, 28071, 32768, 0 },
{ 13365, 20981, 25167, 32768, 0 },
{ 10120, 16964, 21644, 32768, 0 },
{ 7567, 13643, 18303, 32768, 0 },
{ 5919, 11190, 15483, 32768, 0 },
{ 2696, 5219, 7571, 32768, 0 },
{ 24553, 29618, 31290, 32768, 0 },
{ 22244, 28450, 30726, 32768, 0 },
{ 17474, 25189, 28681, 32768, 0 },
{ 13371, 21111, 25560, 32768, 0 },
{ 10166, 17327, 22080, 32768, 0 },
{ 8273, 14790, 19458, 32768, 0 },
{ 4532, 8462, 11904, 32768, 0 },
{ { 14949, 20962, 24363, 32768, 0 },
  { 10935, 17021, 20951, 32768, 0 },
  { 5643, 11512, 15425, 32768, 0 },
  { 4777, 8599, 12056, 32768, 0 },
  { 3766, 6654, 9625, 32768, 0 },
  { 3491, 6250, 8839, 32768, 0 },
  { 1185, 2793, 4413, 32768, 0 },
  { 15352, 21415, 24641, 32768, 0 },
  { 15073, 22544, 25900, 32768, 0 },
  { 9968, 16632, 21230, 32768, 0 },
  { 8613, 14564, 18596, 32768, 0 },
  { 6692, 11538, 15969, 32768, 0 },
  { 5970, 10771, 15054, 32768, 0 },
  { 3974, 7567, 10825, 32768, 0 },
  { 25316, 30467, 32001, 32768, 0 },
  { 22258, 28968, 31209, 32768, 0 },
  { 17604, 25315, 28944, 32768, 0 },
  { 13228, 21372, 25545, 32768, 0 },
  { 10802, 18478, 23911, 32768, 0 },
  { 8865, 15418, 20468, 32768, 0 },
  { 6715, 12028, 16751, 32768, 0 } } },
{ { { 9468, 14755, 18209, 32768, 0 },
    { 7130, 11928, 15157, 32768, 0 },
    { 6383, 11474, 15200, 32768, 0 },
    { 4910, 9054, 12928, 32768, 0 },
    { 4024, 7364, 10586, 32768, 0 },
    { 2982, 5893, 8964, 32768, 0 },
    { 1403, 2798, 4342, 32768, 0 },
    { 12835, 21128, 25312, 32768, 0 },
    { 10046, 16548, 20629, 32768, 0 },
    { 8391, 15249, 19805, 32768, 0 },
    { 7357, 12706, 17214, 32768, 0 },
    { 6038, 10638, 14573, 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 } } } },
  { { { 18958, 25307, 28119, 32768, 0 },
    { 14936, 22106, 26063, 32768, 0 },
    { 8199, 15365, 20003, 32768, 0 },
    { 5888, 11268, 15877, 32768, 0 },
    { 4635, 9064, 13128, 32768, 0 },
    { 3410, 7535, 11086, 32768, 0 },
    { 2181, 4688, 6970, 32768, 0 },
    { 22519, 27770, 30318, 32768, 0 },
    { 20163, 27562, 30336, 32768, 0 },
    { 15113, 23148, 27188, 32768, 0 },
    { 10800, 18577, 23888, 32768, 0 },
    { 8230, 14991, 20344, 32768, 0 },
    { 6998, 13142, 18159, 32768, 0 },
    { 4899, 9673, 13712, 32768, 0 },
    { 24280, 29438, 31568, 32768, 0 },
    { 21377, 28137, 30752, 32768, 0 },
    { 16486, 24638, 28774, 32768, 0 },
    { 12566, 20449, 25387, 32768, 0 },
    { 9859, 18032, 23031, 32768, 0 },
    { 8678, 14689, 19545, 32768, 0 },
    { 5883, 12027, 16289, 32768, 0 } } },
  { { 20198, 27313, 30213, 32768, 0 },
    { 15491, 23301, 27462, 32768, 0 },
    { 10862, 18646, 23716, 32768, 0 },
    { 7564, 13393, 18579, 32768, 0 },
    { 5902, 11728, 15697, 32768, 0 },
    { 4223, 9034, 12669, 32768, 0 },
    { 3928, 6616, 9923, 32768, 0 },

```

```

{ 25378, 30467, 32141, 32768, 0 },
{ 23465, 29589, 31621, 32768, 0 },
{ 18438, 26951, 30412, 32768, 0 },
{ 13588, 21432, 26425, 32768, 0 },
{ 8826, 15492, 21313, 32768, 0 },
{ 9128, 14512, 18256, 32768, 0 },
{ 3542, 11956, 17270, 32768, 0 },
{ 26119, 31027, 31977, 32768, 0 },
{ 24753, 30531, 32246, 32768, 0 },
{ 22098, 28569, 30978, 32768, 0 },
{ 15099, 26022, 30840, 32768, 0 },
{ 9175, 15729, 20972, 32768, 0 },
{ 10923, 21845, 29127, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 19215, 25485, 28552, 32768, 0 },
{ 14627, 22164, 25968, 32768, 0 },
{ 8971, 15622, 20509, 32768, 0 },
{ 6047, 11224, 15688, 32768, 0 },
{ 4535, 8764, 12458, 32768, 0 },
{ 3554, 6907, 10398, 32768, 0 },
{ 1968, 4052, 6172, 32768, 0 },
{ 19942, 26272, 28703, 32768, 0 },
{ 18822, 26157, 29303, 32768, 0 },
{ 14461, 22168, 26504, 32768, 0 },
{ 10378, 17961, 22909, 32768, 0 },
{ 8353, 14777, 19740, 32768, 0 },
{ 6822, 12662, 17145, 32768, 0 },
{ 4468, 8494, 12287, 32768, 0 },
{ 24582, 29924, 31542, 32768, 0 },
{ 22095, 28661, 31008, 32768, 0 },
{ 17810, 25621, 29157, 32768, 0 },
{ 13664, 21851, 26466, 32768, 0 },
{ 10891, 18406, 23387, 32768, 0 },
{ 9093, 15914, 21002, 32768, 0 },
{ 6362, 11751, 16334, 32768, 0 } } },
{ { { 17933, 25150, 28317, 32768, 0 },
{ 14239, 21537, 25749, 32768, 0 },
{ 9140, 15870, 20832, 32768, 0 },
{ 6192, 12191, 16735, 32768, 0 },
{ 4743, 9681, 14326, 32768, 0 },
{ 4203, 7584, 11154, 32768, 0 },
{ 3263, 6690, 10415, 32768, 0 },
{ 22007, 27913, 30826, 32768, 0 },
{ 19865, 27291, 30016, 32768, 0 },
{ 16073, 24363, 28591, 32768, 0 },
{ 13377, 21796, 26375, 32768, 0 },
{ 10220, 17507, 23923, 32768, 0 },
{ 10246, 15969, 22522, 32768, 0 },
{ 6622, 14614, 17811, 32768, 0 },
{ 26801, 30910, 31985, 32768, 0 },

```



```

{ 24783, 30538, 32228, 32768, 0 },
{ 20026, 27705, 30698, 32768, 0 },
{ 15832, 23509, 28786, 32768, 0 },
{ 13184, 22144, 27136, 32768, 0 },
{ 8777, 20773, 25454, 32768, 0 },
{ 8048, 15522, 21845, 32768, 0 } } },
{ { { 14826, 21199, 24951, 32768, 0 },
{ 12131, 19147, 23377, 32768, 0 },
{ 8247, 14396, 18825, 32768, 0 },
{ 5803, 10858, 14609, 32768, 0 },
{ 4156, 8153, 11623, 32768, 0 },
{ 3150, 6565, 9494, 32768, 0 },
{ 1505, 3049, 4665, 32768, 0 },
{ 19971, 26601, 28818, 32768, 0 },
{ 17921, 25586, 28803, 32768, 0 },
{ 13764, 21432, 25786, 32768, 0 },
{ 10157, 16942, 21682, 32768, 0 },
{ 8025, 14231, 18547, 32768, 0 },
{ 6073, 11398, 15693, 32768, 0 },
{ 3087, 5938, 8576, 32768, 0 },
{ 25141, 30226, 31771, 32768, 0 },
{ 22599, 28867, 30997, 32768, 0 },
{ 17837, 25642, 28870, 32768, 0 },
{ 13541, 21609, 26045, 32768, 0 },
{ 10507, 17970, 22646, 32768, 0 },
{ 8440, 15059, 20001, 32768, 0 },
{ 5091, 9497, 13357, 32768, 0 } } },
{ { { 15295, 21345, 25217, 32768, 0 },
{ 11361, 18666, 23159, 32768, 0 },
{ 6123, 13057, 17587, 32768, 0 },
{ 4920, 8440, 12277, 32768, 0 },
{ 4855, 7754, 11125, 32768, 0 },
{ 3002, 6087, 8671, 32768, 0 },
{ 1853, 4156, 5560, 32768, 0 },
{ 15486, 22444, 26035, 32768, 0 },
{ 16111, 22883, 25659, 32768, 0 },
{ 10738, 18148, 22585, 32768, 0 },
{ 9762, 16521, 20070, 32768, 0 },
{ 8385, 13493, 17733, 32768, 0 },
{ 8561, 13432, 17565, 32768, 0 },
{ 5936, 10982, 15256, 32768, 0 },
{ 24333, 31146, 32444, 32768, 0 },
{ 23054, 29242, 31570, 32768, 0 },
{ 18369, 25940, 29714, 32768, 0 },
{ 16230, 23979, 27910, 32768, 0 },
{ 11685, 20293, 25366, 32768, 0 },
{ 10596, 18007, 22662, 32768, 0 },
{ 7742, 14286, 19731, 32768, 0 } } } },
{ { { 7761, 12357, 15729, 32768, 0 },
{ 6770, 11334, 14888, 32768, 0 },

```



[illegible]

```

{ 10426, 21349, 24990, 32768, 0 },
{ 13011, 18312, 24576, 32768, 0 },
{ 11796, 17039, 23593, 32768, 0 } },
{ { 22981, 29440, 31437, 32768, 0 },
{ 16902, 25206, 28312, 32768, 0 },
{ 11077, 18706, 23681, 32768, 0 },
{ 7022, 14746, 21299, 32768, 0 },
{ 5350, 11368, 16718, 32768, 0 },
{ 5699, 9973, 24220, 32768, 0 },
{ 8192, 10923, 19115, 32768, 0 },
{ 23177, 28772, 31969, 32768, 0 },
{ 24242, 28756, 31431, 32768, 0 },
{ 18404, 26933, 30524, 32768, 0 },
{ 10486, 15729, 26214, 32768, 0 },
{ 10923, 21845, 27307, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 20852, 23831, 29789, 32768, 0 },
{ 20025, 29127, 30948, 32768, 0 },
{ 16384, 20480, 24576, 32768, 0 },
{ 6554, 13107, 26214, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 19557, 25990, 28978, 32768, 0 },
{ 15096, 22887, 26498, 32768, 0 },
{ 9157, 16070, 21485, 32768, 0 },
{ 6681, 12336, 17066, 32768, 0 },
{ 4914, 9987, 14310, 32768, 0 },
{ 4175, 7840, 11639, 32768, 0 },
{ 2624, 5787, 8290, 32768, 0 },
{ 19680, 26176, 28870, 32768, 0 },
{ 20770, 28091, 30734, 32768, 0 },
{ 17304, 25338, 29230, 32768, 0 },
{ 13267, 21226, 25933, 32768, 0 },
{ 10659, 18828, 23071, 32768, 0 },
{ 8931, 15354, 20003, 32768, 0 },
{ 6811, 11962, 16892, 32768, 0 },
{ 26088, 30961, 32029, 32768, 0 },
{ 24413, 29987, 31845, 32768, 0 },
{ 20646, 28082, 30775, 32768, 0 },
{ 16772, 24641, 28871, 32768, 0 },
{ 13881, 21720, 26352, 32768, 0 },
{ 10238, 18590, 24113, 32768, 0 },
{ 9064, 15500, 20622, 32768, 0 } } },
{ { 20161, 26862, 30411, 32768, 0 },
{ 15698, 23673, 28129, 32768, 0 },
{ 10639, 18831, 23618, 32768, 0 },
{ 7366, 12955, 16003, 32768, 0 },
{ 5699, 14247, 19946, 32768, 0 },

```

```

{ 3855, 9638, 15420, 32768, 0 },
{ 2185, 8738, 15292, 32768, 0 },
{ 24030, 29127, 31312, 32768, 0 },
{ 20992, 28416, 30976, 32768, 0 },
{ 19895, 26917, 29842, 32768, 0 },
{ 11469, 18022, 24576, 32768, 0 },
{ 12288, 16384, 20480, 32768, 0 },
{ 10923, 21845, 27307, 32768, 0 },
{ 9362, 14043, 18725, 32768, 0 },
{ 24576, 27853, 31130, 32768, 0 },
{ 24966, 28867, 31988, 32768, 0 },
{ 20852, 25321, 28300, 32768, 0 },
{ 5461, 16384, 21845, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 10923, 16384, 21845, 32768, 0 } } },
{ { { 18258, 25057, 28291, 32768, 0 },
{ 13474, 20701, 24865, 32768, 0 },
{ 7940, 13964, 18778, 32768, 0 },
{ 5661, 10551, 14282, 32768, 0 },
{ 4274, 7901, 11333, 32768, 0 },
{ 2867, 6281, 9967, 32768, 0 },
{ 2102, 4307, 6422, 32768, 0 },
{ 18796, 25616, 27695, 32768, 0 },
{ 19022, 26638, 29265, 32768, 0 },
{ 14555, 22758, 27249, 32768, 0 },
{ 10836, 18891, 23387, 32768, 0 },
{ 8166, 14751, 19808, 32768, 0 },
{ 7233, 12491, 17550, 32768, 0 },
{ 5144, 9208, 13012, 32768, 0 },
{ 25863, 31435, 32344, 32768, 0 },
{ 23412, 29807, 31705, 32768, 0 },
{ 19540, 27010, 30118, 32768, 0 },
{ 15187, 23719, 27936, 32768, 0 },
{ 12291, 20372, 25089, 32768, 0 },
{ 10128, 17352, 22379, 32768, 0 },
{ 7104, 12777, 17403, 32768, 0 } },
{ { { 17820, 23712, 27315, 32768, 0 },
{ 13195, 20578, 25690, 32768, 0 },
{ 6746, 13600, 18526, 32768, 0 },
{ 6687, 11703, 14378, 32768, 0 },
{ 4161, 7802, 11963, 32768, 0 },
{ 5595, 9591, 11988, 32768, 0 },
{ 2016, 3529, 5545, 32768, 0 },
{ 20632, 24273, 30341, 32768, 0 },
{ 16846, 22384, 26537, 32768, 0 },
{ 11663, 19994, 23326, 32768, 0 },
{ 6827, 13653, 17749, 32768, 0 },
{ 6554, 13107, 19661, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },

```

```

{ 8192, 16384, 24576, 32768, 0 },
{ 18204, 25486, 29127, 32768, 0 },
{ 23406, 25746, 28087, 32768, 0 },
{ 25206, 27727, 30247, 32768, 0 },
{ 18725, 23406, 28087, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 },
{ 8192, 16384, 24576, 32768, 0 } } },
{ { { 9404, 15586, 19652, 32768, 0 },
{ 9286, 15268, 19629, 32768, 0 },
{ 6652, 12742, 17493, 32768, 0 },
{ 5620, 10675, 15165, 32768, 0 },
{ 4601, 8682, 12283, 32768, 0 },
{ 3679, 7220, 11247, 32768, 0 },
{ 1515, 3602, 5798, 32768, 0 },
{ 17538, 26768, 29076, 32768, 0 },
{ 16683, 24491, 27823, 32768, 0 },
{ 13944, 21392, 25937, 32768, 0 },
{ 10852, 17235, 22257, 32768, 0 },
{ 6780, 12562, 17747, 32768, 0 },
{ 6087, 10043, 14710, 32768, 0 },
{ 4058, 8290, 11432, 32768, 0 },
{ 23954, 29761, 31524, 32768, 0 },
{ 21992, 28343, 30572, 32768, 0 },
{ 18097, 25787, 29183, 32768, 0 },
{ 14507, 22607, 26994, 32768, 0 },
{ 10912, 19313, 23594, 32768, 0 },
{ 9279, 15164, 20096, 32768, 0 },
{ 6245, 10534, 13894, 32768, 0 } } },
{ { { 16326, 23042, 26979, 32768, 0 },
{ 16384, 23749, 27657, 32768, 0 },
{ 9947, 18139, 23991, 32768, 0 },
{ 6242, 15604, 21845, 32768, 0 },
{ 4096, 9557, 15019, 32768, 0 },
{ 5120, 7168, 13312, 32768, 0 },
{ 3449, 7761, 12072, 32768, 0 },
{ 21065, 25746, 30427, 32768, 0 },
{ 15381, 22737, 26081, 32768, 0 },
{ 12072, 22420, 28169, 32768, 0 },
{ 8495, 15777, 19418, 32768, 0 },
{ 7282, 14564, 21845, 32768, 0 },
{ 5174, 10348, 13797, 32768, 0 },
{ 4274, 7123, 11398, 32768, 0 },
{ 18725, 23406, 28087, 32768, 0 },
{ 18432, 21504, 25600, 32768, 0 },
{ 10426, 25321, 31279, 32768, 0 },
{ 3277, 16384, 19661, 32768, 0 },
{ 9362, 18725, 23406, 32768, 0 },
{ 10923, 16384, 21845, 32768, 0 },
{ 10923, 18204, 21845, 32768, 0 } } } },

```



## 9.4. Quantizer Matrix Tables

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 9.4.2](#). For informative purposes the subsampling process is defined in [section 9.4.1](#).

### 9.4.1. 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 9.4.2](#).

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  $j=0..Tx\_Width[ txSz ] - 1$ .



## 9.4.2. 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 }
```

Draft Document

```

Quantizer_Matrix[ 15 ][ 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 },
{ /* 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 },
},
{
  /* 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 },
{ /* 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 },
},
{
  /* 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 },
{ /* 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 },

```

```

},

```

```

{

```

```

{ /* 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, 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 },
{ /* 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, 64, 66, 67, 71,
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, 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 },

```

```

},
{

```

```

{ /* 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 },
{ /* 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 },
```

```
},
{
```

```
{ /* 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, 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, 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, 67, 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 },
{ /* 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, 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, 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, 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, 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, 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, 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 },

```

```

},
{

```

```

{ /* 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, 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, 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, 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, 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, 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 },
{ /* 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, 52, 52,
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,
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, 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, 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 },
},
{
  /* 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, 64, 67, 69, 72,
  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, 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, 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, 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, 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, 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 },
{ /* 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, 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,  
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, 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, 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, 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, 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, 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 },

```

```

},
{

```

```

{ /* 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, 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, 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, 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, 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, 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, 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 },
{ /* 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, 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, 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, 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, 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, 47, 48, 48, 49, 50,
50, 51, 53, 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, 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, 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, 46, 46, 47, 52, 53, 55,
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, 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 },

```

```

},
{

```

```

{ /* 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 35, 35, 38, 41, 41,
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, 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, 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, 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, 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, 33,
33, 33, 34, 34, 34, 35, 35, 35, 36, 37, 37, 38, 38, 40, 40, 40,
41, 43, 43, 43, 45, 46, 46, 48, 32, 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 },
{ /* 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, 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, 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, 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, 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, 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, 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, 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, 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, 58,
60, 60, 61, 61, 61, 63, 50, 49, 48, 48, 47, 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, 33, 35,  
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, 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, 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, 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 },
},
{
  /* 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, 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,
  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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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 },
{ /* 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, 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, 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,  
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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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,  
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, 48,
```

```

49, 49, 49, 51, 53, 53, 53, 53, 54, 54, 54, 55, 55, 55, 55, 48, 48, 47, 47,
47, 46, 45, 45, 45, 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, 46, 46, 46, 46, 46, 46, 46, 47, 49, 49, 49, 51, 53, 53, 53, 54,
55, 55, 55, 56, 57, 57, 49, 48, 47, 47, 47, 46, 45, 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, 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,
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, 53, 53, 53, 53, 53, 53, 49, 48, 46, 46, 46, 47, 48, 48, 48, 50,
53, 53, 53, 53, 53, 53, 49, 48, 46, 46, 46, 47, 48, 48, 48, 50, 53, 53,
53, 53, 53, 53, 48, 47, 46, 46, 46, 47, 47, 47, 47, 50, 53, 53, 53, 54,
54, 54, 48, 47, 46, 46, 46, 46, 47, 47, 47, 50, 53, 53, 53, 54, 56, 56,
48, 47, 46, 46, 46, 46, 47, 47, 47, 50, 53, 53, 53, 54, 56, 56, 48, 47,
46, 46, 46, 46, 47, 47, 47, 50, 53, 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,
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, 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, 49, 49, 49, 48, 47, 47, 47,
47, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 49, 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,
46, 46, 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, 48, 48, 47, 47, 47, 46, 45, 45, 45, 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 },

```

```

},

```

```

{

```

```

{ /* 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, 33, 33,  
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,  
42, 45, 48, 48,

/\* Size 32x32 \*/

32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32,  
32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 36, 36, 36, 37, 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, 35, 35, 35, 37, 31, 31, 31, 31, 31, 31, 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, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 34, 34, 34, 35, 35, 35,  
35, 36, 31, 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, 35, 35, 35, 35, 36, 31, 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, 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, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34,  
34, 34, 34, 35, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 35,  
31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
32, 32, 32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 31, 31, 32, 32,  
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,  
33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 31, 31, 32, 32, 32, 32, 32, 32,  
32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34,  
34, 35, 35, 35, 35, 36, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33,  
33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 36, 36, 36,  
36, 37, 31, 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, 36, 36, 36, 36, 36, 37, 31, 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, 36, 36, 36, 36, 36, 37, 31, 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, 36, 36, 36, 36, 36, 37, 32, 32, 32, 32, 32, 32, 32, 32, 32,  
32, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35, 36, 36, 36, 37,  
37, 37, 37, 38, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34,

```

34, 34, 34, 35, 35, 35, 35, 35, 36, 36, 36, 36, 37, 37, 38, 38, 38, 39,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 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, 32, 32, 32, 33, 33, 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,
32, 32, 32, 33, 33, 34, 34, 34, 34, 35, 35, 35, 35, 36, 36, 37, 37, 37,
37, 38, 38, 38, 38, 39, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
34, 34, 34, 34, 35, 35, 36, 36, 36, 36, 37, 38, 38, 38, 38, 39, 40, 40,
40, 41, 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, 34, 34,
34, 34, 34, 34, 34, 33, 33, 33, 33, 34, 35, 35, 35, 35, 36, 36, 37, 37,
37, 38, 39, 39, 39, 39, 41, 42, 42, 42, 42, 43, 34, 34, 34, 34, 34, 34,
34, 33, 33, 33, 33, 34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 38, 39, 39,
39, 39, 41, 42, 42, 42, 42, 43, 34, 34, 34, 34, 34, 34, 34, 33, 33, 33,
33, 34, 35, 35, 35, 35, 36, 36, 37, 37, 37, 38, 39, 39, 39, 39, 41, 42,
42, 42, 42, 43, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 36,
36, 36, 36, 37, 37, 37, 37, 38, 40, 41, 41, 41, 42, 44, 45, 45, 45, 45,
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, 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, 35, 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, 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, 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, 37,
37, 37, 32, 32, 32, 33, 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, 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, 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, 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, 33, 33, 33, 33, 34, 34,
34, 34, 34, 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, 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, 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, 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 },
{ /* 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, 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, 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, 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, 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, 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, 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, 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, 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, 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,  
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, 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, 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, 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, 47, 48, 48, 48, 48, 50, 52, 53, 53, 53, 49, 48, 47, 46, 46, 46,
47, 48, 48, 48, 48, 50, 52, 53, 53, 53, 49, 48, 47, 46, 46, 46, 47, 47,
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, 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, 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, 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, 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 },

```

```

},
{

```

```

{ /* 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, 32, 33, 31, 32,
  32, 32, 32, 32, 32, 33, 31, 32, 32, 32, 32, 32, 32, 33, 32, 32, 32, 32,
  33, 33, 34, 35, 32, 32, 32, 32, 33, 34, 34, 35, 32, 32, 32, 32, 34, 34,
  35, 36, 33, 33, 33, 33, 35, 35, 36, 38,
  /* Size 16x16 */
  32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 34, 31, 31,
  31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 33, 33, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
  32, 32, 33, 33, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
  33, 34, 31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 34, 35,
  31, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 31, 32,
  32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 34, 35, 32, 32, 32, 32,
  32, 32, 32, 33, 33, 33, 33, 34, 35, 35, 35, 36, 32, 32, 32, 32, 32, 32,
  32, 33, 33, 34, 34, 35, 35, 35, 36, 37, 32, 32, 32, 32, 32, 32, 32, 33,
  33, 34, 34, 35, 35, 35, 36, 37, 32, 33, 33, 33, 33, 33, 33, 33, 34, 34,

```

```
/* Size 32x32 */
```

```

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, 33, 33, 33, 33, 33, 34, 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, 34, 35,
35, 35, 36, 36, 36, 36, 36, 37, 38, 38, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35, 35, 36, 36, 36,
36, 36, 37, 38, 38, 38, 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, 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, 33, 33, 33, 34, 34,
34, 35, 35, 35, 36, 37, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 35,
35, 35, 36, 37, 32, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 35, 36, 36,
36, 38,
/* Size 16x32 */
32, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 31, 31,
31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 33, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
33, 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, 32, 32, 33, 33, 31, 32,
32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 31, 32, 32, 32,
32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 31, 32, 32, 32, 32, 32,
32, 32, 33, 33, 33, 33, 33, 33, 33, 31, 32, 32, 32, 32, 32, 32, 32, 33,
33, 33, 33, 33, 33, 34, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33,
33, 33, 34, 34, 31, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 34, 34, 34,

```

```

34, 35, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35,
32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35, 32, 32,
32, 32, 32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35, 32, 32, 32, 32,
32, 32, 32, 33, 33, 33, 34, 34, 34, 34, 34, 35, 32, 32, 32, 32, 32, 32,
32, 33, 33, 34, 34, 34, 34, 34, 35, 35, 32, 32, 32, 32, 32, 32, 32, 33,
33, 34, 35, 35, 35, 35, 35, 36, 32, 32, 32, 32, 33, 33, 33, 33, 33, 34,
35, 35, 35, 35, 36, 36, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 35, 35,
35, 35, 36, 37, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35,
36, 37, 32, 32, 32, 32, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 36, 37,
32, 32, 32, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 36, 37, 32, 33,
33, 33, 33, 33, 33, 33, 34, 35, 36, 36, 36, 36, 36, 38, 33, 33, 33, 33,
33, 33, 33, 34, 34, 35, 36, 36, 36, 36, 37, 38, 34, 34, 34, 34, 34, 34,
34, 34, 35, 36, 37, 37, 37, 37, 38, 39, 34, 34, 34, 34, 34, 34, 34, 34,
35, 36, 37, 37, 37, 37, 38, 39,
/* Size 32x16 */
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, 31,
31, 31, 31, 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, 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, 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, 32, 32, 32, 32, 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, 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, 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, 33, 34, 34, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
33, 34, 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, 32, 32, 32, 32, 32, 32, 33, 33,
33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 36, 36, 32, 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, 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, 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, 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, 36, 36,
37, 37, 37, 37, 38, 38, 39, 39 },
{ /* 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,  
35, 38, 40, 40, 41, 42, 30, 31, 31, 31, 31, 32, 32, 32, 32, 33, 35, 36, 36, 38, 40, 40,  
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, 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, 31, 32, 32, 34, 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, 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, 32, 33, 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, 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, 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, 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, 31, 31,  
31, 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, 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, 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, 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, 43, 43,
44, 45, 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, 33, 33, 34, 34, 34, 35, 35,
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, 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, 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,
```

$$\left. \begin{array}{l} \{ \\ \} \end{array} \right\}$$

```
/* Size 8x4 */
```

32, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,

Page 579 of 607

```
/* Size 32x32 */
```

```

35, 35, 35, 35, 35, 35, 35, 36, 37, 37, 38, 39, 39, 39, 39, 39, 39, 40,
33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35,
35, 35, 35, 36, 37, 37, 38, 39, 39, 39, 39, 39, 39, 40, 33, 33, 34, 34,
34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36,
37, 37, 38, 39, 39, 39, 39, 39, 39, 40, 33, 33, 34, 34, 34, 34, 34, 34,
34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 37, 37, 38, 39,
39, 39, 39, 39, 39, 40, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 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, 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, 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, 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, 31, 32, 34, 35, 36, 38, 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, 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, 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, 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, 33, 34, 36, 38, 39, 30, 31, 31, 31, 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, 30, 31, 31, 31, 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, 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, 35, 36,
37, 39, 40, 42, 33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43,
41, 43, 33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 33, 34,
33, 34, 34, 35, 35, 36, 36, 36, 36, 36, 36, 37, 38, 40, 41, 43, 33, 34, 34, 35,
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, 33, 34, 31, 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, 32, 33, 33, 34, 35, 35, 35, 35,
35, 35, 31, 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, 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, 31, 32, 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, 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, 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, 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, 36, 32, 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, 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, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 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 },

```

```

},

```

```

{

```

```

{ /* Luma */

```

```

/* Size 4x4 */

```

```

31, 31, 31, 31, 31, 32, 32, 32, 31, 32, 32, 32, 31, 32, 32, 32,

```

```

/* Size 8x8 */

```



```
/* Size 32x32 */
```

Page 584 of 607

[illegible]

[illegible]

```
/* Size 4x8 */
```

```
/* Size 8x4 */
```

[illegible]

$$\left. \begin{array}{l} \} \\ \} \end{array} \right\},$$

# Annex A: Profiles and Levels

## A.1. Overview

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.1.1. 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:** Main profile consists of YUV 4:2:0 or monochrome sequences with bit depth equal to 8 or 10. High profile consists of all main profile sequences plus YUV 4:4:4 sequences with bit depth equal to 8 or 10 (monochrome is not allowed when `seq_profile=1`). Professional profile consists of all high and main profile sequences plus YUV 4:2:2 sequences with bit depth equal to 8 or 10 or 12, 4:2:0 sequences with bit depth equal to 12, 4:4:4 sequences with bit depth equal to 12, and monochrome sequences with bit depth equal to 12.

### A.1.2. Levels

TODO: Still under discussion.



# Annex B: Bitstream Format

## B.1. Overview

An AV1 bitstream consists of a number of OBUs that are normally held within a container format alongside audio and timing information.

This annex defines a simple method of packing OBUs into a bitstream format.

## B.2. Bitstream Syntax

```
bitstream( sz ) {  
    while ( sz > 0 ) {  
        obu_length  
        sz -= Leb128Bytes  
        open_bitstream_unit( obu_length )  
        sz -= obu_length  
    }  
}
```

## B.3. Bitstream semantics

**sz** specifies the number of bytes in the entire bitstream and is provided by external means.

**obu\_length** specifies the length in bytes of the next OBU.

**Note:** It is legal 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 file is deemed invalid.

# Annex C: Included Experiments

## C.1. Overview

This is a temporary section that describes how the specification relates to the current code base.

## C.2. Included Experiments

The specification has been updated to include the following list of adopted experiments:

Experiment	Notes
alt_intra	
altref2	
amvr	
aom_qm	
aom_qm_ext	
cb4x4	
cdef	
cdef_singlepass	
cdf_update_mode	
cfl	
chroma_sub8x8	
CICP	
colorspace_headers	
compound_round	
compound_segment	
convolve_round	
daala_ec	
deblock_13tap	
delta_q	
dual_filter	
ec_adapt	
ec_multisymbol	

Experiment	Notes
ec_smallmul	
explicit_order_hint	
ext_comp_refs	
ext_delta_q	
ext_intra	
ext_intra_mod	
ext_intra_mod2	
ext_inter	
ext_partition	
ext_partition_types	
ext_qm	
ext_refs	
ext_skip	
ext_tile	
ext_tx	
ext_warped_motion	
fast_sgr	
film_grain_noise	
film_grain_noise_showex	
filter_7bit	
filter_intra	
frame_refs_signaling	
frame_marker	
frame_sign_bias	
frame_size	
fwd_kf	
global_motion	
gm_ref_mv	

Experiment	Notes
HLS R18	
HLS R19	
horzonly_frame_superres	
independent_row_tile	
interintra	
intrabc	
intra_edge	
intra_edge2	
jnt_comp	
kf_ctx	
loop_restoration	
loopfilter_level	
lowprecision_blend	
lv_map	
lv_map_multi	
max_tile	
mfmv	
misc_fixes	
mono_video	
motion_var	
mv_compress	
new_multisymbol	
new_tokenset	
no_frame_context_signalling	
obu_frame	
obu_redundant_frame_header	
obu_size_after_header	
obu_sizing	

Experiment	Notes
one_sided_compound	
operating_points	
opt_ref_mv	
palette	
palette_delta_encoding	
palette_throughput	
parallel_deblocking	
q_adapt_probs	
rect_intra_pred	
rect_tx	
rect_tx_ext	
rect_tx_ext_intra	
ref_adapt	
ref_mv	
reference_buffer	
restricted segmentation map update	
scalability	
segment_pred_last	
segment_zeromv	
short_filter	
simp_mv_pred	
simple_bwd_adapt	
simplify_tx_mode	
skip_sgr	
smooth_hv	
spatial_segmentation	
striped_loop_restoration	
tempmv_signaling	

Experiment	Notes
tile_info_first	
tile_groups	
timecodes	
trailing_bits	
tx64x64	
txk_sel	
txmg	
var_tx	
warped_motion	
wedge	

### C.3. Excluded Experiments

The specification has not yet been updated with the following experiments:

Experiment	Notes

# 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.