International Standard

ISO/IEC 21794-5

First edition
2024-11

Information technology — Plenoptic image coding system (JPEG Pleno) —

Part 5: Holography

Technologies de l'information — Système de codage d'images plénoptiques (JPEG Pleno) —

Partie 5: Holographie

Reference number
ISO/IEC 21794-5:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 21794 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document is part of a series of standards for a system known as JPEG Pleno and defines JPEG Pleno Holography. It specifies a codec mechanism for holographic modalities and associated codestream syntax and file format elements. JPEG Pleno Holography allows for efficient compression of holograms for a wide range of applications such as holographic microscopy, tomography, interferometry, printing and display and their associated hologram types. Key functionalities include support for both lossy and lossless coding, scalability, random access, and integration within the system architecture of the JPEG Pleno framework.

Information technology — Plenoptic image coding system (JPEG Pleno) —

Part 5: Holography

1 Scope

This document defines the syntax and an accompanying decompression process that is capable of representing binary and continuous-tone holograms while supporting one or multiple color/spectral components. The supported compression mechanisms are lossless for binary holograms and lossy for continuous-tone holograms. Additional information on the encoding tools is provided as well. The document also defines extensions to the JPEG Pleno File Format and associated metadata descriptors specific to holographic modalities.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 21794-1:2020, *Information technology — Plenoptic image coding system (JPEG Pleno) — Part 1: Framework*

ISO/IEC 21794-2:2021, *Information technology — Plenoptic image coding system (JPEG Pleno) — Part 2: Light field coding*

ISO/IEC 21794-3, *Information technology — Plenoptic image coding system (JPEG Pleno) — Part 3: Conformance testing*

ISO/IEC 21794-4, *Information technology — Plenoptic image coding system (JPEG Pleno) — Part 4: Reference software*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 21794-1, ISO/IEC 21794-2, ISO/IEC 21794-3, ISO/IEC 21794-4 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp/ui

— IEC Electropedia: available at https://www.electropedia.org/

3.1
holography
technique based on coherent light allowing for the recording and reconstruction of a wavefront, thereby encoding three-dimensional information about objects

Note 1 to entry: This has many applications, including 3D display technology, microscopy, tomography, interferometry, telecommunications, data storage, and non-destructive testing.

3.2
hologram
two-dimensional representation of the complex-valued coherent wavefield of light encoding an interference pattern describing the amplitude and phase of the scalar wavefield

Note 1 to entry: This pattern may be encoded directly in the spatial domain or indirectly using optical transformations such as Fourier-transforming lens systems, magnification, or modulation.

3.3
digital hologram
one or more two-dimensional arrays of coefficients representing the sampled coherent wavefield of light

3.4
tile
a spatial segment of a digital hologram, each coded independently

3.5
window block
unit of a series of 2D windows over the propagated hologram's input coefficients, corresponding to 2D contiguous subsets of input coefficients and serving as input for the STFT

3.6
transform block
coefficient set resulting from applying a transform to a tile

3.7
code block
input coefficient set of the arithmetic coding within a transform block, where each code block is independently arithmetically encoded

3.8
quantization block
unit of quantization within a code block

3.9
transform size
4-tuple of positive integers, describing the number of elements along each dimension of the entire set of coefficients after applying the transform step, which is represented by a 4D array

3.10
source hologram
uncompressed input hologram

3.11
decoded hologram
output decompressed hologram decoded back into raw hologram samples

4 Symbols and abbreviated terms

4.1 Symbols

H	Digital hologram
λ_i	The wavelength of the i^{th} colour channel at which the hologram was recorded or generated
$\odot$	Hadamard product
$\mathcal{F}$, $\mathcal{F}^{-1}$	Forward and reverse two-dimensional Fourier transform

$\boldsymbol{S}$, $\boldsymbol{S}^{-1}$	Forward and reverse two-dimensional FFTSHIFT operator. This operation rearranges Fourier transform coefficients by shifting the zero-frequency component to the centre of the array, matching its conventional mathematical representation.
H_i	I[th] monochrome component of the digital hologram *H;* Corresponds to λ_I.
p	Hologram pixel pitch sampled on a square lattice
d	Distance parameter for propagation operator
x, y	Symbols for the first and second spatial dimensions
f_x , f_y	Symbols for the first and second frequency dimensions
NT	Number of tiles
NCB	Number of code blocks per tile
NQB	Number of quantization blocks per tile
$XThoc$, $YThoc$	Tile x and y dimensions
$XTcod, YTcod,$ $UTcod, VTcod$	Power of 2 exponents of the four-dimensional transform block dimensions
$XCBcod, YCBcod,$ $UCBcod, VCBcod$	Power of 2 exponents of the four-dimensional code block dimensions
$XQBcod, YQBcod,$ $UQBcod, VQBcod$	Power of 2 exponents of the four-dimensional quantization block dimensions
$\mathcal{Q}$, $\mathcal{Q}^{-1}$	Mid-rise quantizer and dequantizer
Λ_1 , Λ_2	Lagrangian multipliers for systems 1 and 2, respectively

4.2 Abbreviated terms

2D	Two-dimensional
4D	Four-dimensional
CB	Code block
CBP	Code block payload
DFT	Discrete Fourier transform
FFTSHIFT	Fourier transform coefficient shifting operator
L2	Euclidean norm
MRQ	Mid-rise quantizer
MRDQ	Mid-rise dequantizer
QB	Quantization block
QBP	Quantization block payload
RDO	Rate-distortion optimization

SNR	Signal-to-noise ratio
STFT	Short-time Fourier transform
TB	Transform block

5 Conventions

5.1 Naming conventions for numerical values

Integer numbers are expressed as bit patterns, hexadecimal values, or decimal numbers. Bit patterns and hexadecimal values have both a numerical value and an associated length in bits.

Hexadecimal notation, indicated by prefixing the hexadecimal number with "0x", may be used instead of binary notation to denote a bit pattern having a length that is an integer multiple of 4. For example, 0x41 represents an eight-bit pattern with only its second most significant bit and its least significant bit equal to 1. Numerical values that are specified under a "Code" heading in tables that are referred to as "code tables" are bit pattern values (specified as a string of digits equal to 0 or 1 in which the left-most bit is considered the most significant bit). Other numerical values not prefixed by "0x" are decimal values. When used in expressions, a hexadecimal value is interpreted as having a value equal to the value of the corresponding bit pattern evaluated as a binary representation of an unsigned integer (i.e. as the value of the number formed by prefixing the bit pattern with a sign bit equal to 0 and interpreting the result as a two's complement representation of an integer value). For example, the hexadecimal value 0xF is equivalent to the 4-bit pattern '1111' and is interpreted in expressions as being equal to the decimal number 15.

5.2 Operators

NOTE Many of the operators in this document are similar to those in the C programming language.

5.2.1 Arithmetic operators

+	Addition
−	subtraction (as a binary operator) or negation (as a unary prefix operator)
×	Multiplication
/	division without truncation or rounding
<<	left shift; `x<<s` is defined as $x \times 2^s$
>>	right shift; `x>>s` is defined as $\lfloor x/2^s \rfloor$
++	increment with 1
--	decrement with 1
umod	Unsigned modulo operator; x umod a is the unique value y between 0 and a−1 for which y+Na = x with a suitable integer N
&	bitwise AND operator; compares each bit of the first operand to the corresponding bit of the second operand If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

^ bitwise XOR operator; compares each bit of the first operand to the corresponding bit of the second operand
If both bits are equal, the corresponding result bit is set to 0. Otherwise, the corresponding result bit is set to 1.

5.2.2 Logical operators

|| logical OR

&& logical AND

! logical NOT

5.2.3 Relational operators

> greater than

>= greater than or equal to

< less than

<= less than or equal to

== equal to

!= not equal to

5.2.4 Precedence order of operators

Operators are listed in descending order of precedence. If several operators appear in the same line, they have equal precedence. When several operators of equal precedence appear at the same level in an expression, evaluation proceeds according to the associativity of the operator, either from right to left or from left to right.

Operators	Type of operation	Associativity
()	Expression	left to right
[]	indexing of arrays	left to right
++, --	increment, decrement	left to right
!, –	logical not, unary negation	
×, /	multiplication, division	left to right
umod	unsigned modulo (remainder)	left to right
+, –	addition and subtraction	left to right
&	bitwise AND	left to right
^	bitwise XOR	left to right
&&	logical AND	left to right
||	logical OR	left to right
<<, >>	left shift and right shift	left to right

< , >, <=, >=	Relational	left to right

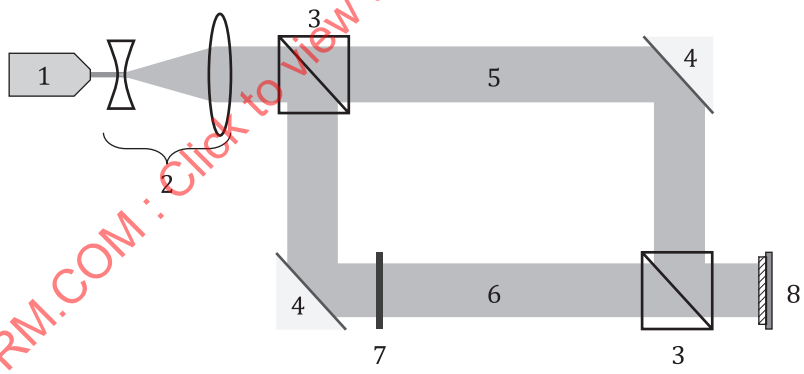
5.2.5 Mathematical functions

|x|	absolute value, is –x for x < 0, otherwise x
sign(x)	sign of x, zero if x is zero, +1 if x is positive, -1 if x is negative
clamp(x,min,max)	clamps x to the range [min,max]: returns min if x < min, max if x > max or otherwise x
⌈x⌉	ceiling of x; returns the smallest integer that is greater than or equal to x
⌊x⌋	floor of x; returns the largest integer that is less than or equal to x
⌊x⌉	rounding of x to the nearest integer, equivalent to

6 Representation of digital holograms

6.1 Digital holograms and their signal properties

Like conventional images, holograms can be optically recorded, computer-generated, or some hybrid thereof. A classic holographic recording setup utilizes interferometry, where a coherent, monochromatic light beam is split into a reference beam and an object beam. Only the object beam will illuminate the sample to be imaged, after which both beams are recombined, forming an interference pattern (cf. Figure 1). This pattern encodes the complex-valued amplitudes of the object wavefront, thus describing both the wavefront amplitude and phase. Alternatively, one can utilize numerical diffraction simulations in virtual objects and scenes to compute the hologram pattern, called computer-generated holography. Although this approach is not limited by the physics of an optical setup, such as geometrical constraints or optical component nonidealities, it tends to require significant computational resources to calculate.



Key

1 laser
2 beam expander
3 beam splitter
4 mirror
5 reference beam
6 object beam
7 sample
8 image sensor

Figure 1 — Simplified diagram of a holographic recording setup, in case of a transparent sample, with corresponding legend.

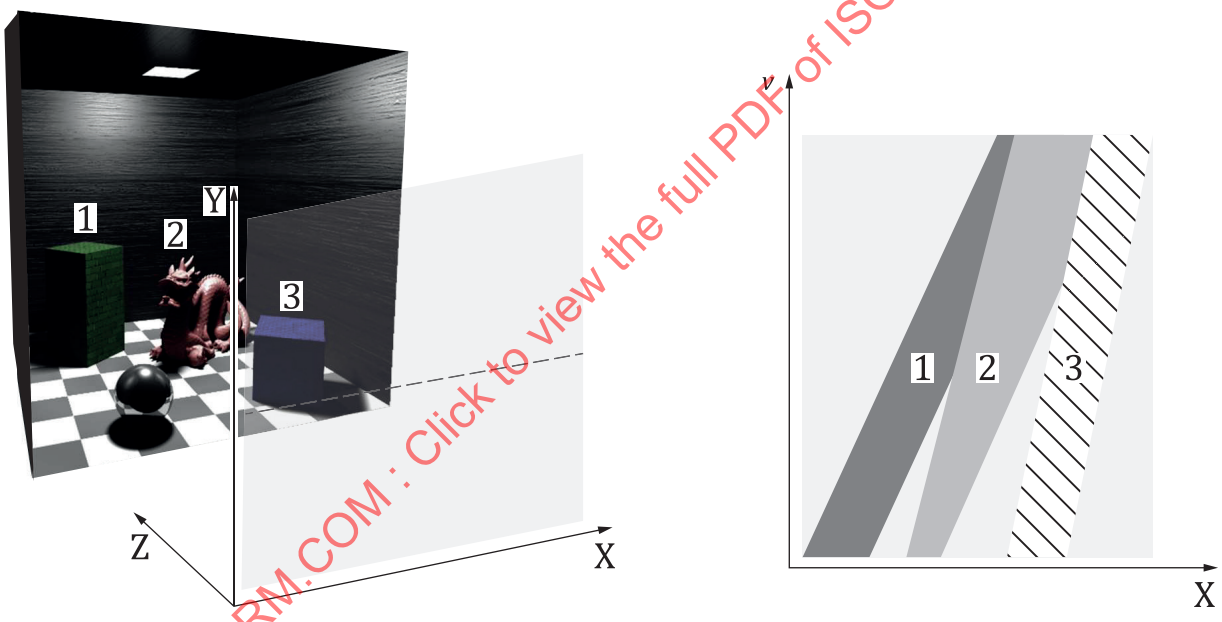
However, holography is often confused with other display techniques that do not involve interference. Many systems that claim to be "holographic" consist of various projection systems that display flat images in mid-air, using, for example, semi-transparent screens. These "false holograms" are produced by technologies such as Pepper's ghost illusion (or its modern variants), lenticular printing, and volumetric displays. These technologies are not addressed in this standard.

Holograms and discrete light fields (hereupon just called "light fields") are closely related, as both representations encode spatial and angular information. A full-parallax light field can be represented by a 4D array of rectilinear samples with two spatial and two angular dimensions, as specified in ISO/IEC 21794-2.

Because of the discrete sampling, light fields will have a discretized motion parallax and limited depth cues, but they will have fewer samples than a hologram to cover the same spatio-angular range and work with incoherent light. In contrast, holograms can account for all human visual cues, including continuous motion parallax and exact eye-focusing cues.

In holography, frequency components correspond to plane waves whose propagation angle is proportional to the frequency magnitude. Thus, one can characterize the relationship to light fields by looking at phase space, jointly representing space and frequency data using, e.g., a spectrogram. (cf. Figure 2). This matches closely with the spatio-angular phase space of 4D light fields.

This relationship was the primary rationale for utilizing the short-time Fourier transform for hologram compression in this specification. The size of the transform window will dictate the trade-off between spatial and angular resolution.



a) 3D scene with numbered (coloured) objects, creating a hologram in the x-y plane, analyzing the signal along the dashed horizontal line

b) corresponding spatio-frequency phase diagram, with matching numbered (coloured) footprints in phase space

Key

X space

v frequency

Figure 2 — Relationship between objects in 3D space and the phase space representation.

The hologram sample values are represented by discretized two-dimensional arrays of coefficients. These coefficients may encode the complex-valued amplitudes in whole or in part, for example, by only encoding the phase, the real part, or the amplitude of the complex coefficients. Moreover, the hologram coefficients may have different levels of precision, ranging from floating-point numbers down to 1 bit per pixel, denoted as "binary holograms."

A hologram consists of one or more arrays with the same dimensions, denoted "channels," representing different colour components. Every channel may have different wavelengths and pixel pitches.

6.2 Functional overview of the decoding process

This document specifies the JPEG Pleno Holography superbox and the JPEG Pleno Holography decoding algorithm. These are detailed in the annexes; the status of Annexes A to I is normative, while the status of Annex J is informative. The generic JPEG Pleno Holography superbox syntax is specified in Annex A.

The overall architecture of JPEG Pleno Holography (Figure 3) provides the flexibility to configure the encoding and decoding system depending on the requirements of the addressed use case.

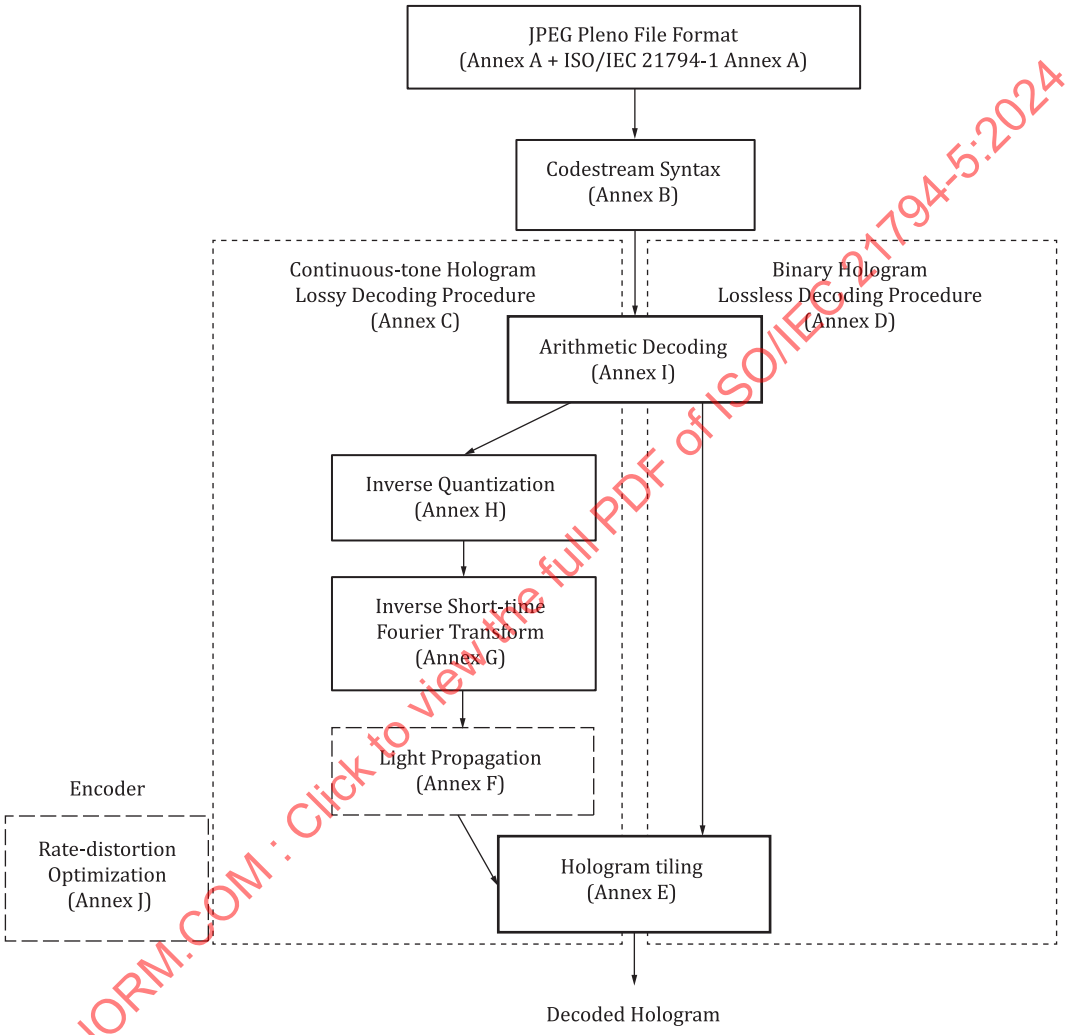


Figure 3 — General JPEG Pleno Holography decoder architecture

As can be seen in Figure 3, each annex provides information about the individual components of the codec architecture. Annex A provides a description of the JPEG Pleno Holography superbox and its composing boxes. The codestream syntax issued in this specification is specified in Annex B. Subsequently, Annex E describes the tiling mechanism issued to handle potentially large binary and non-binary holograms.

The annexes specific to the encoding/decoding of non-binary holograms are:

— Annex C gives a general overview of the non-binary lossy coding pipeline

— To enable the propagation of the hologram to a more suitable plane for compression Annex F specifies several supported propagation models.

— Annex G specifies the short-time Fourier transform (STFT).

— Quantization mechanisms are specified in Annex H.

— Annex I describes the context-based arithmetic codec.

— Finally, Annex J illustrates how rate-distortion optimization can be implemented.

The module specific to the encoding and decoding of binary holograms is described in Annex D, particularly focused on the Bayesian context-based arithmetic (de)coding mechanism.

6.3 Encoder requirements

An encoding process converts source hologram data to coded hologram data.

To conform with this document, an encoder shall conform with the file format syntax and codestream format syntax specified, respectively, in Annex A and Annex B for the encoding process(es) embodied by the encoder.

6.4 Decoder requirements

A decoding process converts coded hologram data to decoded hologram data. The resulting data may or may not be perfectly reconstructed depending on whether the lossless coding mode was used. Annex E through Annex I describe and specify the decoding process.

A decoder is an embodiment of the decoding process. To conform to this document, a decoder shall convert all, or specific parts of, any coded hologram data that conform to the file format syntax and codestream syntax specified, respectively, in Annex A and Annex B to a decoded hologram.

Annex A
(normative)

JPEG Pleno Holography superbox

A.1 General

This annex specifies the use of the JPEG Pleno Holography superbox, which is designed to contain compressed holographic data and associated metadata. The listed boxes shall comply with their definitions as specified in ISO/IEC 21794-1 and ISO/IEC 21794-2.

This document may redefine the binary structure of some boxes defined as part of the ISO/IEC 15444-1 or ISO/IEC 15444-2 file formats. For those boxes, the definition found in this document shall be used for all JPL files (cf. ISO/IEC 21794-1:2020, Annex A).

A.2 Organization of the JPEG Pleno Holography superbox

Figure A.1 shows the hierarchical organization of the JPEG Pleno Holography superbox contained by a JPL file. This illustration does not specify nor imply a specific order to these boxes. The file will often contain several boxes of a particular box type. Each box's meaning depends on the placement and order of that particular box within the file. This superbox comprises the JPEG Pleno Holography Header box containing parameterization information about the hologram, such as size and wavelength parameters.

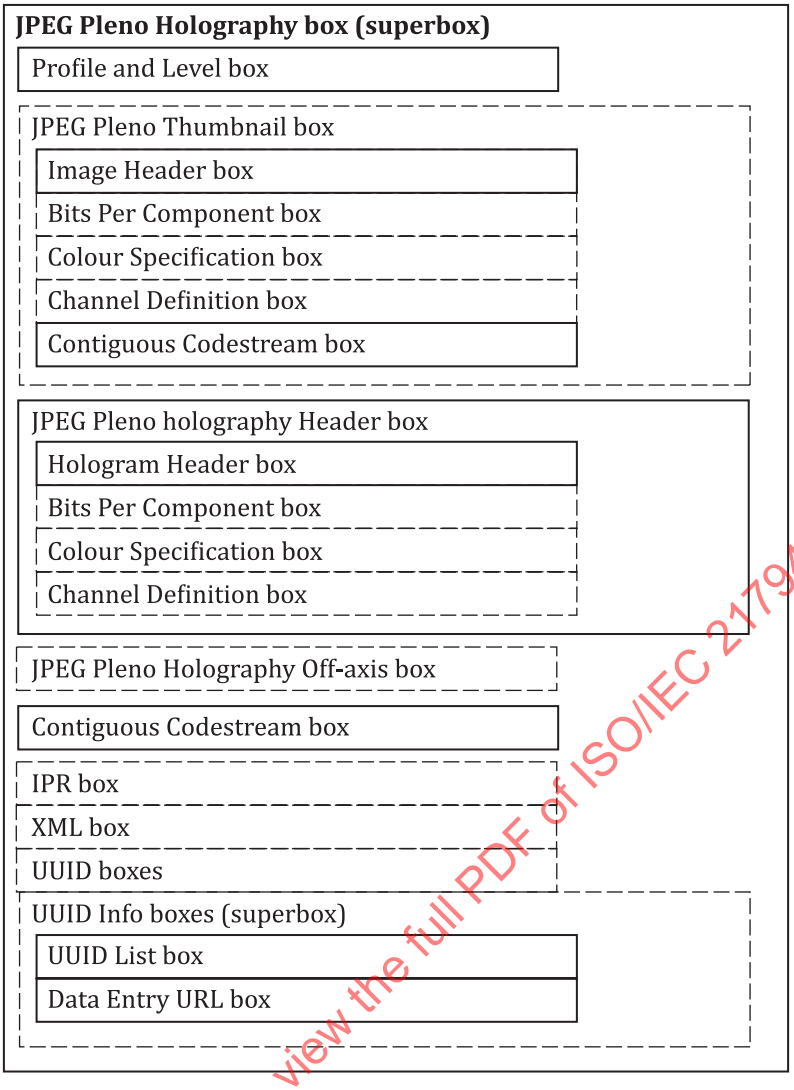


Figure A.1 — Hierarchical organization of a JPEG Pleno Holography superbox

Table A.1 lists all boxes defined as part of this document. Boxes defined as part of the ISO/IEC 15444-1 or ISO/IEC 15444-2 file formats are not listed. A box that is listed in Table A.1 as "Required" shall exist within all conforming JPL files. For the placement and restrictions on each box, see the relevant section defining that box.

The IPR, XML, UUID and UUID boxes introduced in ISO/IEC 15444-1:2024, Annex A can be signalled, as well at the level of the JPEG Pleno Holography box, to carry hologram-specific metadata.

A.3 Defined boxes

A.3.1 Overview

The boxes in Table A.1 shall properly be interpreted by all conforming readers. Each box conforms to the standard structure defined in ISO/IEC 21794-1:2020, Annex A. The following clauses define the value of the DBox field. It is assumed that the LBox, TBox, and XLBox fields exist for each box in the file as defined in ISO/IEC 21794-1:2020, Annex A.

Table A.1 — Defined boxes

Box name	Type	Superbox	Required?	Comments
JPEG Pleno Holography box	'jpho' (0x6A70 6C66)	Yes	Yes	This box contains a series of boxes that contain the encoded hologram, its parameterization and associated metadata. (Defined in ISO/IEC 21794-1:2020, Annex A)
JPEG Pleno Holography Header box	'jphh' (0x6A70 6C6C)	Yes	Yes	This box contains generic information about the file, such as the number of components, bits per component and colour space.
Hologram Header box	'hhdr' (0x6C6C 6472)	No	Yes	This box contains fixed length generic information about the hologram, such as hologram dimensions, number of components, codec and bits per component.

A.3.2 JPEG Pleno Holography Header box

A.3.2.1 General

Within a JPEG Pleno Holography superbox, there shall be one and only one JPEG Pleno Hologram Header box. The JPEG Pleno Holography Header box contains generic information about the hologram, such as the hologram size, number of components, and bits per component. This box is a superbox.

The JPEG Pleno Holography Header box may be located anywhere within the JPEG Pleno Holography superbox but should be before the enclosed Contiguous Codestream box containing the hologram codestream.

The JPEG Pleno Holography Header box type shall be 'jphh' (0x6A70 6C6C).

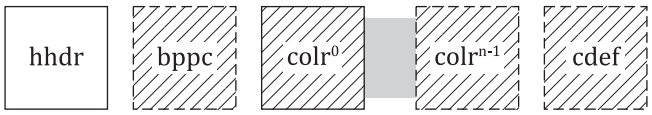
This box contains several boxes. Other boxes may be defined in other documents and may be ignored by conforming readers. Those boxes contained within the JPEG Pleno Holography Header box that are defined within this document are shown in Figure A.2:

— The Hologram Header box specifies information about the hologram size, bit depth, and the number of components. This box shall be the first box in the JPEG Pleno Holography Header box and is specified in subclause A.3.2.2.

— The Bits Per Component box specifies the bit depth of the components in the file in cases where the bit depth is not constant across all components. Its structure shall be as specified in ISO/IEC 15444-1.

— The Colour Specification boxes specify the colour space of the decompressed image. Their structures shall be as specified in ISO/IEC 15444-2. There shall be at least one Colour Specification box within the JPEG Pleno Header box. Using multiple Colour Specification boxes gives a decoder multiple optimization or compatibility options for colour processing. These boxes may be found anywhere in the JPEG Pleno Holography Header box, provided they come after the Holography Header box. All Colour Specification boxes shall be contiguous within the JPEG Pleno Holography Header box.

— The Channel Definition box defines the channels in the image. Its structure shall be as specified in ISO/IEC 15444-1. This box may be found anywhere in the JPEG Pleno Holography Header box, provided that it comes after the Hologram Header box.



Key

hhdr	Hologram Header box
bppc	Bits Per Component box
colr^{i}	Colour Specification boxes
cdef	Channel Definition box

Figure A.2 — Organization of the contents of a JPEG Pleno Holography Header box

A.3.2.2 Hologram Header box

A.3.2.2.1 General

This box contains fixed-length generic information about the hologram field, such as hologram dimensions, number of components, codec, and bits per component. The contents of the JPEG Pleno Holography Header box shall start with a Hologram Header box. Instances of this box in other places in the file shall be ignored. The length of the Hologram Header box shall be 24 bytes, including the box length and type fields. Much of the information within the Hologram Header box is redundant, with information stored in the codestream itself.

All references to "the codestream" in the descriptions of fields in this Hologram Header box apply to the codestream found in the first Contiguous Codestream box embedded in the current JPEG Pleno Holography superbox. Files containing contradictory information between the Hologram Header box and the first codestream are not conforming. However, readers may attempt to read these files by using the values within the codestream.

The type of the Hologram Header box shall be 'hhdr' (0x6C6C 6472) and the contents of the box shall have the format shown in Table A.3:

— **Width (Xhoc):** The value of this parameter indicates the width of the sample grid. This field is stored as a 4-byte big-endian unsigned integer.

— **Height (Yhoc):** The value of this parameter indicates the height of the sample grid. This field is stored as a 4-byte big-endian unsigned integer.

— **Number of components (NC):** This parameter specifies the number of components in the codestream and is stored as a 2-byte big-endian unsigned integer. The value of this field shall be equal to the value of the NC field in the HOC marker in the codestream (as defined in subclause B.5.3). If no Channel Definition Box is available, the order of the components for colour images is R-G-B-Aux or Y-U-V-Aux.

— **Type (Thoc):** The value of this parameter indicates the hologram type. This field is stored as a 1-byte big-endian unsigned integer. Legal values for this field are 0 if the hologram is real-valued (includes amplitude only), 1 if the hologram is complex-valued in Cartesian coordinates (real-imaginary), 2 if the hologram is phase-only and 3 if the hologram is complex-valued in polar coordinates (amplitude-phase). All other values are reserved for ISO/IEC use.

— **Data type (DThoc):** The value of this parameter indicates the hologram data type. This 8-bit field is configured as '00TT00GG', where every symbol denotes a different bit. TT=00 for signed integer values, TT=01 for unsigned integer values, TT=10 for floating-point, and TT=11 for packed binary. In the case of floating-point values, the maximum exponent value equals 2^{3+GG}, where "GG" represents a 2-bit unsigned integer. All other values are reserved for ISO/IEC use (Table A.4).

— **BPC:** This parameter specifies the bit depth of the components in the codestream, minus 1, and is stored as a 1-byte field (Table A.5). The low 7 bits of the value indicate the bit depth of the components. The high-bit indicates whether the components are signed or unsigned. If the high-bit is 1, then the components contain signed values. If the high-bit is 0, then the components contain unsigned values. If the components

vary in bit depth or sign, or both, then the value of this field shall be 255, and the Hologram Header box shall also contain a Bits Per Component box defining the bit depth of each component (as defined in subclause A.3.2.2.1).

— **C:** This parameter specifies the coding algorithm used to compress the hologram data. It is encoded as a 1-byte unsigned integer. It the value is 0, the ISO/IEC 21794-5 codec is used. All other values are reserved for ISO/IEC use.

— **UnkC:** This field specifies if the actual colour space of the hologram data in the codestream is known. This field is encoded as a 1-byte unsigned integer. Legal values for this field are 0 if the colour space of the hologram is known and correctly specified in the Colour Space Specification boxes within the file or 1 if the colour space of the hologram is not known. A value of 1 will be used in cases such as the transcoding of legacy holograms where the actual colour space of the image data is not known. In these cases, while the colour space interpretation methods specified in the file may not accurately reproduce the hologram with respect to an original, the hologram should be treated as if the methods do accurately reproduce the hologram. Values other than 0 and 1 are reserved for ISO/IEC use.

— **IPR:** This parameter indicates whether this JPL file contains intellectual property rights information related to the holographic content. If the value of this field is 0, this file does not contain rights information, and thus the file does not contain an IPR box. If the value is 1, then the file does contain rights information and thus does contain an IPR box as defined in ISO/IEC 15444-1. Other values are reserved for ISO/IEC use.

Table A.3 — Format of the contents of the Hologram Header box

Field name	Size (bits)	Value
WIDTH	32	1 to (2^{32}– 1)
HEIGHT	32	1 to (2^{32}– 1)
Type	32	0
Data Type	32	See Table A.4
NC	16	1 to 16 384
BPC	8	See Table A.5
UnkC	8	0 to 1
IPR	8	0 to 1

Table A.4 — Data Type values

Values (bits) MSB LSB	Component sample precision
x000 0000 to x010 0101	Component bit depth = value + 1. From 1 bit deep to 38 bits deep respectively (counting the sign bit, if appropriate).
0xxx xxxx	Components are unsigned values.
1xxx xxxx	Components are signed values.
1111 1111	Components vary in bit depth.
	All other values reserved for ISO/IEC use.

Table A.5 — BPC values

Values (bits) MSB LSB	Component sample precision
00xx00**00**	8-bit datatype values
00xx00**01**	16-bit datatype values
00xx00**10**	32-bit datatype values
00xx00**11**	64-bit datatype values
00**00**00xx	Signed integer datatype values
00**01**00xx	Unsigned integer datatype values
00**10**00xx	Floating-point datatype values
00**11**00xx	Packed binary datatype values
	All other values reserved for ISO/IEC use.

Annex B
(normative)

JPEG Pleno Holography codestream syntax

B.1 General

This section specifies the marker and marker segment syntax and semantics defined by this document. These markers and marker segments provide codestream information for this document. Further, this subclause provides a marker and marker segment syntax that is designed to be used in future specifications that include this document as a normative reference.

This document does not include a definition of conformance. The parameter values of the syntax described in this annex are not intended to portray the capabilities required to be compliant.

B.2 Markers, marker segments, and headers

This document uses markers and marker segments to delimit and signal the characteristics of the source image and codestream. This set of markers and marker segments is the minimal information needed to achieve the features of this document and is not a file format.

The main header is a collection of markers and marker segments. The main header is found at the beginning of the codestream.

Every marker is 4 bytes long, using the "0xFFFFFFXX" pattern, where the last marker byte, "XX," defines the type of marker segment and can have any value in the range 0x01 to 0xFE.

A marker segment includes a marker and associated parameters, called marker segment parameters. When a marker segment that is not specified in the document appears in a codestream, the decoder shall discard the marker segment.

The proposed codec does not put any direct restrictions on the payload byte structure of the arithmetic encoder. This maximizes flexibility and reduces average file sizes compared to, for example, ISO/IEC 15444-1 (JPEG 2000), which forbids the signalling of 0xFF bytes, thereby reducing average compression efficiency.

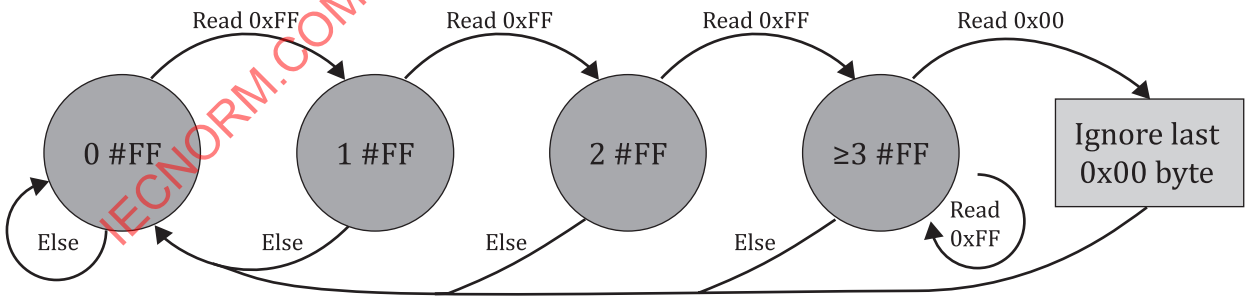


Figure B.1 — Marker segment manipulation.

Rarely, sequences of 3 or more subsequent 0xFF bytes may be signalled in the payload. To avoid the signalling of spurious marker segments, an "escape sequence system" is introduced, as shown in the state machine diagram illustrated in Figure B.1. To achieve this, the marker bytes 0x00 and 0xFF are reserved, and may not be used as part of the 4-byte marker segment code.

Whenever 3 consecutive 0xFF bytes are encountered in the payload, the state machine will check for escape bytes (state "≥3 #FF" in the diagram above). Any extra 0xFF bytes will be decoded as payload while

remaining in that same state. If the escape byte 0x00 is encountered, that 0x00 byte is skipped and will not be considered part of the payload; the state machine then resets to its initial state “0 #FF”. If, however, any other byte in the range 0x01-0xFE is encountered after at least 3 subsequent 0xFF bytes, the pattern will be recognized as a 4-byte marker segment code.

Aside from avoiding spurious marker code signalling, it also allows for the efficient signalling of long 0xFF sequences without additional coding cost, not requiring an extra 0x00 byte after every three 0xFF bytes.

Each marker segment is described in terms of its function, usage, and length. The function describes the information contained in the marker segment. The usage describes this marker segment's logical location and frequency in the codestream. The length describes which parameters determine the length of the marker segment.

The marker segments are designated by the three-letter code of the marker associated with the marker segment. The parameter symbols have capital letter designations followed by the marker's symbol in lower-case letters. The remaining clauses in this annex describe the structure and meaning of each parameter in every marker segment.

Every marker segment begins with a 32-bit marker code represented with the symbol MAR. Some marker segments have an adaptive size, given by a size of the length parameter SLmar (Table B.1), followed by a “length of marker segment” parameter. These are detailed in Table B.2. This SLmar marker is not mandatory for every marker segment; only HOC, TPM, and CPM require SLmar.

After the list is a table that either describes the allowed parameter values or references other tables that describe these values. Tables for individual parameters are provided to describe any parameter without a simple numerical value. In some cases, these parameters are described by a bit value in a bit field. In this case, an "x" is used to denote bits that are not included in the specification of the parameter or sub-parameter in the corresponding row of the table.

Table B.1 — Size parameters for the SLmar

Value (bits)		Parameter size
MSB	LSB	
xxxx xx00		Length parameter is 16 bits.
xxxx xx01		Length parameter is 32 bits.
xxxx xx10		Length parameter is 64 bits.
		All other values are reserved

Table B.2 — Adaptive size parameters (SLL); “xxx” placeholder is marker segment dependent

Field name	Size (bit)	Value	Details
SLxxx	8	0, 1, 2	size of Lxxx parameter
Lxxx	16/32/64	0 - 2^(2^(4+SLxxx))	length of marker segment in bytes (not including the marker)

B.3 Defined marker segments

Table B.3 lists the markers specified in this document.

Table B.3 — List of defined marker segments

	Symbol	Code	Main Header	Tile Header	Code-block Header
Start of codestream	SOC	0xFFFFFFB0	Required	Not allowed	
Hologram Configuration	HOC	0xFFFFFFB1	Required	Not allowed	
Coding style default	COD	0xFFFFFFB2	Required	Not allowed	
Coding style component	COC	0xFFFFFFB3	Optional	Optional	
Quantization style default	QCD	0xFFFFFFB4	Required	Not allowed	
Quantization style component	QCC	0xFFFFFFB5	Optional	Optional	
Tile pointer	TPM	0xFFFFFFB6	Optional		
Code-block pointer	CPM	0xFFFFFFB7		Optional	
Start of tile	SOT	0xFFFFFFB8		Required	
Start of tile channel	STC	0xFFFFFFB9		Required	
Start of code block	SOB	0xFFFFFFBA			Required
End of codestream	EOC	0xFFFFFFBB	Required		

B.4 Construction of codestream

In Figure B.2, the order of the different marker segments in the codestream is shown. Blocks with a solid line are required, while dashed-line segments are optional. The signalling logic is as follows: COD and QCD shall be signalled in the global header. The values will be used for all tiles and all channels unless overwritten. COC and QCC may be used to update specific channels. The updates will only apply within the tile and channels where the COC and QCC are signalled. The COD, COC, QCD, and QCC settings of the global header always apply, at least to the first tile. That is, for the first tile, no QCCs may be signalled.

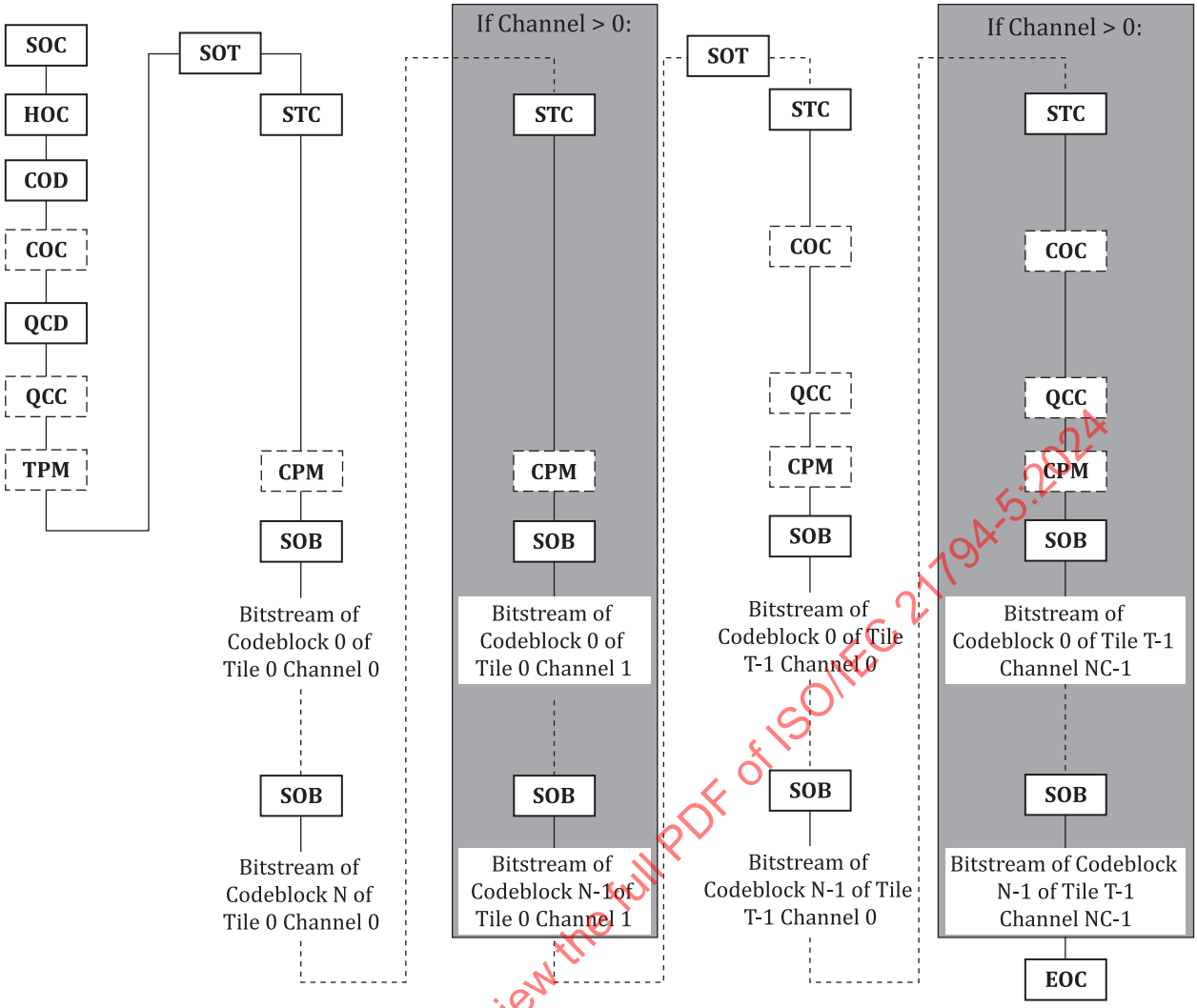


Figure B.2 — Codestream structure

B.5 Delimiting markers and marker segments

B.5.1 General

The delimiting markers and marker segments shall be present in all codestreams conforming to this document. Each codestream has only one SOC marker and one EOC marker and contains at least one 4D block. Each 4D block has a SOB marker. The SOC, SOB, and EOC are delimiting markers, not marker segments, with no explicit length information or other parameters.

B.5.2 Start of codestream (SOC)

This marker indicates the start of the codestream. Table B.4 provides the syntax for the Start of Codestream (SOC) marker.

Table B.4 — Start of codestream (SOC) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB0	marker code

B.5.3 Hologram configuration

This marker signals the hologram's parameters and properties. Table B.5 provides an overview of the fields in the Hologram configuration (HOC) marker syntax.

Table B.5 — Hologram configuration (HOC) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB1	marker code
Adaptive size parameters (SLL) — Table B.2 — SLhoc, Lhoc			
Xhoc	32	1 to 2^{32}-1	hologram width
Yhoc	32	1 to 2^{32}-1	hologram height
NC	16	1 to 16384	number of (colour components)
SMhoc	8	0-2	signalling mode: (0 = default) nothing, (1) non-square pixel pitch, (2) pixel pitch per channel
Thoc	8	0-3	hologram type: real-valued (0), complex-valued in Cartesian coordinates (i.e. real-imaginary) (1), phase-only (2), complex-valued in polar coordinates (i.e. amplitude-phase) (3)
DThoc	8	8-bit bitmask pattern, defined as 00TT00GG (See Table A.4 for more details)	Datatype: TT = 00: signed integer / 01: unsigned integer / 10: floating-point / 11: packed binary, GG = size exponent = 2^{3+GG}
(...			
$Shoc^i$	8	See Table B.19 of ISO/IEC 21794-2:2021	precision (depth) in bits and sign of the i^{th} component samples
$Whoc^i$	32	float	wavelength (in m)
$XPhoc^i$	32	float [if i = 0 || bit #3 of SMhoc]	pixel pitch x-dimension (in m)
$YPhoc^i$	32	float [optional: bit #2 of SMhoc]	pixel pitch y-dimension (in m)
...)			

B.5.4 Coding style default

This marker signals the generic configuration of the coding modes for all components. Table B.7 provides an overview of the fields in the syntax for the Coding style default (COD) marker.

Table B.6 — Coding style parameter (CO) sub-table

Field name	Size (bit)	Value	Details
XTcod	8	0-15 (uint) [optional]	transform block width exponent value
YTcod	8	0-15 (uint) [optional]	transform block height exponent value
XCBcod	8	0-15 (uint)	codeblock dimension X exponent value
YCBcod	8	0-15 (uint)	codeblock dimension Y exponent value
UCBcod	8	0-15 (uint) [optional: if 4D transform output]	codeblock dimension U exponent value
VCBod	8	0-15 (uint) [optional: if 4D transform output]	codeblock dimension V exponent value

Table B.7 — Coding style default (COD) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFFB2	marker code
Lcod	16	Max. 65535	length of marker segment in bytes (not including the marker)
Ccod	8		coding mode: (0=default) lossless binary, (1) lossy floating-point. The lossy/lossless attribute only refers to the transform, independently of the chosen quantizer.
Pcod	8	0-5	propagation mode: (0=default) no transform, (1) Angular spectrum method, (2) Convolutional Fresnel transform, (3) Fourier-type Fresnel transform, (4) Fourier-domain Fresnel transform, (5) Fraunhofer transform.
PDcod	32	float [optional: Pcod > 0]	propagation distance (in m)
Tcod	8		transform mode (0=default) no transform, (1) Short-time Fourier transform (STFT)
Coding style parameters (CO) - Table B.6			

B.5.5 Coding style component

This marker allows to signal for each component separately a different coding configuration than the one defined in the COD marker segment. Table B.8 provides an overview of the fields in the syntax for the Coding style component (COC) marker.

Table B.8 — Coding style component (COC) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB3	marker code
Lcod	16	Max. 65535	length of marker segment in bytes (not including the marker)
Ccoc	16	0 to 16383	The index of the component to which this marker segment relates. The components are indexed 0, 1, 2, etc.
Coding style parameters (CO) - Table B.6			

B.5.6 Quantization style default

This marker signals the generic configuration of the quantization modes for all components. Table B.10 provides an overview of the fields in the syntax for the Quantization style default (QCD) marker. Details on the quantization block (QB) organization and processing can be found in Annex E.2.4.

Table B.9 — Quantization style parameter (QC) sub-table

Field name	Size (bit)	Value	Details
MSVqcd	32	float [if mode = 1]	Maximum saturation value for quantizer
BDqcd	8	uint [if mode = 1]	Maximum bit depth of quantizer
XQBqcd ... VQBqcd	4x 8	4x 0-15 uint for x,y,u,v [if mode = 2]	4D QB dimension exponent values
QBbdmax	8	uint [if mode = 2]	Maximal bit depth of coefficients (global)
QBTqcd	16	16-bit binary mask [if mode = 2]	Used QB table entries for bit depths 1 to 16, popcount #elements
(...			

Table B.9 *(continued)*

Field name	Size (bit)	Value	Details
QBDqcd^{i}	8	uint [if mode = 1 & bit #i of QBTqcd]	Bit depth of Xmax
QBMIDqcd^{i}	32		Xmid value
QBMAXqcd^{i}	32		Xmax value
...)			
CTXTRqcd	8	uint [if mode = 3]	Context row count
(...			
CTXTDqcd^{i}	16	Uint [if mode = 3 & i < CTXTRqcd]	Context shape coordinates (uint8 x, uint8 y)
...)			

Table B.10 — Quantization style default (QCD) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB4	marker code
Lqcd	16	Max. 65535	length of marker segment in bytes (not including the marker)
ECqcd	8	0	Entropy Coding Mode (0 = Fixed-point Arithmetic Coder)
Mqcd	8	0-3	General Quantization Mode (0 = None, 1 = Saturated Uniform Quantizer, 2 = Double-Adaptive quantizer, 3 = Binary Quantizer)
Quantization style parameters (QC) - Table B.9			

B.5.7 Quantization style component

This marker signals the configuration of the quantization mode for a specific component i. Table B.11 provides an overview of the fields in the syntax for the Quantization style component (QCC) marker.

Table B.11 — Quantization style component (QCC) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB5	marker code
Lqcd	16	Max. 65535	length of marker segment in bytes (not including the marker)
Cqcc	16	0 to 16383	The index of the component to which this marker segment relates. The components are indexed 0, 1, 2, etc.
Quantization style parameters (QC) - Table B.9			

B.5.8 Tile pointer

This marker signals the tile pointer table, allowing for efficient codestream parsing and, hence, random access. Table B.12 provides an overview of the fields in the syntax for the Tile pointer (TPM) marker.

Table B.12 — Tile pointer (TPM) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB6	marker code
Adaptive size parameters (SLL) - Table B.2 - SLtpm, Ltpm			
SPtpm	8	0, 1	size of Ptpm parameter
(...			
$Ptpm^i$	32, 64	1 to 2^{32}-1, 1 to 2^{64}-1	pointer to codestream of Tile i, relative offset in bytes from the start of the codestream.
...)			

B.5.9 Code-block pointer

This marker signals the code block pointer table, allowing for efficient codestream parsing and, hence, random access. Table B.13 provides an overview of the fields in the syntax for the Code-block pointer (CPM) marker.

Table B.13 — Code-block pointer (CPM) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB7	marker code
Adaptive size parameters (SLL) - Table B.2 - SLtpm, Ltpm			
SPcpm	8	0, 1	size of Pcpm parameter
(...			
$Pcpm^i$	32, 64	1 to 2^{32}-1, 1 to 2^{64}-1	pointer to codestream of codeblock i, relative offset in bytes from the start of the tile.
...)			

B.5.10 Start of tile

This marker signals the start of a new tile. Table B.14 provides an overview of the fields in the syntax for the Start of tile (SOT) marker.

Table B.14 — Start of tile (SOT) parameter values

Field name	Size (bit)	Value	Details
MAR	16	0xFFFFFFB8	marker code
Lsot	16	Max. 65535	length of marker segment in bytes (not including the marker)
Nsot	16	0-65535	Tile index. This number refers to the tiles in raster scan order starting at the number 0.
TLsot	32		Length, in bytes, from the beginning of the first byte of this SOT marker segment to the end of the data of this tile Nsot.

B.5.11 Start of tile channel

This marker signals the start of a new tile channel. Table B.15 provides an overview of the fields in the syntax for the Start of tile channel (STC) marker.

Table B.15 — Start of tile channel (STC) parameter values

Field name	Size (bit)	Value	Details
MAR	16	0xFFFFFFB9	marker code
Lstc	16	Max. 65535	length of marker segment in bytes (not including the marker)
Nstc	16	0-65535	Tile channel index. This number refers to the tile channel number corresponding to the wavelength vector starting at the number 0.

B.5.12 Start of code block

This marker signals the start of a new code block. Table B.16 provides an overview of the fields in the syntax for the Start of code block (SOB) marker.

Table B.16 — Start of code block (SOB) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFB9	marker code
Lsob	16	Max. 65535	length of marker segment in bytes (not including the marker)
Nsob	16	0-65535	Codeblock index. This number refers to the code-blocks in raster order starting at the number 0.
TLsob	32		Length, in bytes, from the beginning of the first byte of this SOB marker segment to the end of the data of this code block.

B.5.13 End of codestream

This marker signals the end of the codestream. Table B.17 provides an overview of the fields in the syntax for the End of codestream (EOC) marker.

Table B.17 — End of codestream (EOC) parameter values

Field name	Size (bit)	Value	Details
MAR	32	0xFFFFFFBA	marker code

Annex C
(normative)

Non-binary lossy coding

C.1 Encoding

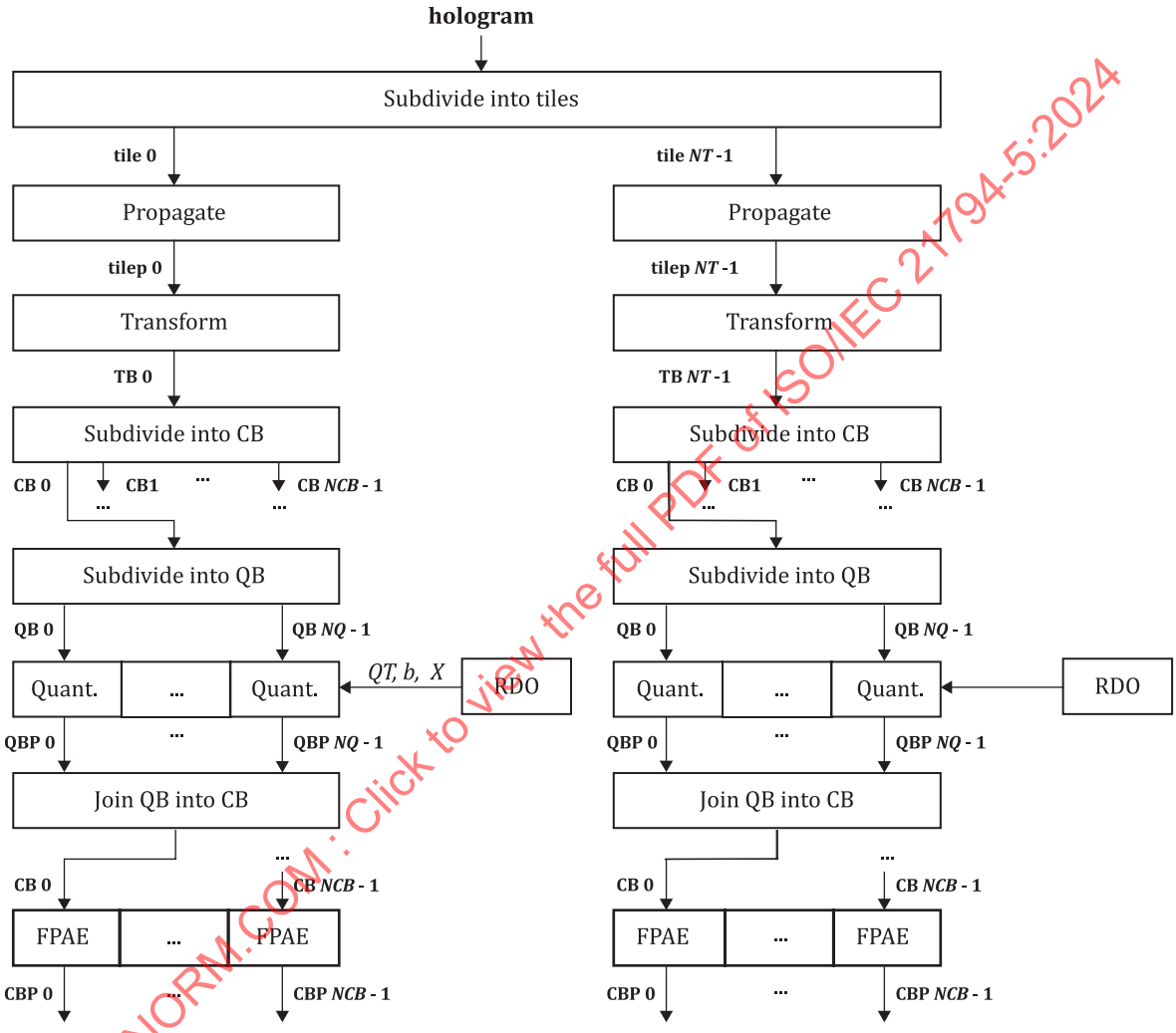


Figure C.1 — verview of the encoder data flow

This section summarizes the different encoding steps and the corresponding data flow, whose architecture is illustrated in Figure C.1. The components will be detailed further in subsequent annexes.

First, the hologram is divided into NT tiles, where $NT = \left\lceil \frac{\text{Xhoc}}{\text{XThoc}} \right\rceil \times \left\lceil \frac{\text{Yhoc}}{\text{YThoc}} \right\rceil$ and where each tile is encoded independently. More details can be found in Annex E. The encoding of a tile comprises the following steps – propagation, transformation, quantization, and entropy coding, which shall be performed in the mentioned order and will be detailed in the subsections below. At the end of the encoding process, NCB encoded code blocks are obtained per tile. The final codestream representing the hologram's encoded output shall comprise one or more of these code blocks. The main encoding procedure is given in Table C.1.

Table C.1 — Main encoding procedure for lossy non-binary holograms

Procedure Encoding (hologram){	See also Tables C.2, C.3 and J.1.
tiles = subdivide_hologram(hologram);	Subdivide hologram into tile
for (tile : tiles){	Loop over tiles present in hologram
tilep = propagate(tile);	Propagate tile
for (wb : tilep){	Loop over propagated tile with a series of windows, returning WB
tb_slice = transform(wb);	Do transform per WB (e.g. 2D-DFT), returning a spatial slice of the 4D TB
tb.append(tb_slice);	... and append to the 4D TB
}	
rdo_data = RDO(tb);	Perform RDO for each tile
for (cb : tb){	Loop over all CB in the TB
for (qb : cb){	Loop over all QB in CB
{qt, b, X} = rdo_data.next();	Get corresponding RDO QB data
qbp = Quantization(qb, qt, b, X);	Quantize QB into QBP
}	
cbp = FPAE(cb);	Entropy coding on CB
codestream.append(cbp);	Append CBP to codestream
}	
}	

C.1.1 Propagation

Every tile can be propagated over space with numerical diffraction operators for potentially improved compressibility and to convert the data to a common complex-valued wavefield representation. This functionality is detailed in Annex F on light propagation models.

C.1.2 Short-Time Fourier transform

Every propagated tile is subsequently processed with the Short-time Fourier transform to obtain a 4D array of coefficients called a "transform block" (TB). This is achieved by iterating over the propagated tile with a series of 2D windows returning contiguous subsets of its coefficients, called “window block” (WB). For the rectangular window STFT, the propagated tile is partitioned into non-overlapping WB, each transformed with the 2D-DFT, which are then all combined together into a single TB. Details and definitions are given in Annex G on the Short-time Fourier transform.

C.1.3 Rate-Distortion Optimization

After organizing the transform data from the tile into NQB QBs, the RDO module shall determine the following for each QB:

— The bit depth b to be used by the MRQ for the QB coefficients

— The quantization range X and then used by the MRQ for the QB coefficients

Additionally, it shall determine QT containing the auxiliary parameters $Q\text{off}[b], q[b]$ and $Q[b]$ to be used by the MRQ for quantizing the range for all bit depths $b \in \{0, \ldots, BDqcd\}$. An informative discussion on how the RDO may be done is given in Annex J, where the L2 distortion is used for optimization.

C.1.4 Quantization

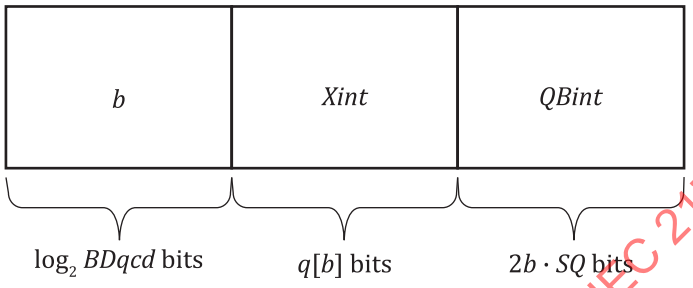
The quantization procedure discussed in Annex H utilizes the information provided by the RDO module to quantize the information in some QB to a quantization block payload (QBP). The QBP consists of the following information

— Bit depth b used by the MRQ for the QB coefficients

— Quantized range $Xint$ used by the MRQ for QB coefficients

— Quantized QB coefficients QBint obtained from the MRQ

The QBP corresponding to an exemplary QB is visualized in Figure C.2.



The bits expressed at the bottom are the contribution of each component to the payload prior to the application of entropy coding

Figure C.2 — Visualization of the QBP payload

The QBPs will then be entropy coded, as discussed in the next section. The subroutine used for quantization at the encoder is given in Table C.2.

Table C.2 — Quantization procedure for encoding the QBPs belonging to a tile

Procedure Quantization (qb, qt, b ,X){	See Tables H.1 and H.2
qbp.b = b;	Add bit depth to QB payload
qbp.Xint = MRQ(X-qt.Qoff[b],qt.q[b],qt.Q[b]);	Add quantized QB range to QB payload
Xhat = MRDQ(qbp.Xint, qt.q[b], qt.Q[b]) + qt.Qoff[b];	
qbp.qbint = MRQ(qb,b,Xhat);	Add quantized QB STFT coefficients to QB payload
return qbp;	
}	

C.1.5 Entropy encoding

The data from the *NQB* QBPs belonging to the tile is organized into *NCB* CBPs. Each CBP will be independently encoded by fixed point arithmetic encoding (FPAE). Entropy coding is achieved using the adaptive fixed-point arithmetic coding module detailed in Annex I. For the purpose of entropy coding, as discussed in Annex C, the *NQB* QBPs belonging to the tile are organized into *NCB* code blocks (CB). where *NCB* = $\mathrm{XThoc}\times\mathrm{YThoc}\times 2^{-\mathrm{XCBcod}-\mathrm{YCBcod}-\mathrm{UCBcod}-\mathrm{VCBcod}}$. Each CB is an organization of $2^{(\mathrm{XCBcod}-\mathrm{XQBqcd})+(\mathrm{YCBcod}-\mathrm{YQBqcd})+(\mathrm{UCBcod}-\mathrm{UQBqcd})+(\mathrm{VCBcod}-\mathrm{VQBqcd})}$ neighbouring QBPs and can be entropy encoded and decoded independently.

Adaptive fixed point arithmetic coding is applied on all three QB components (bit depth, range and coefficients) belonging to the QBP in the code block. Inputs to the arithmetic encoder and decoder are

expressed in terms of a cumulative count vector (see Annex J) corresponding to the probability distribution of all possible alphabets of the currently encoded/decoded symbol.

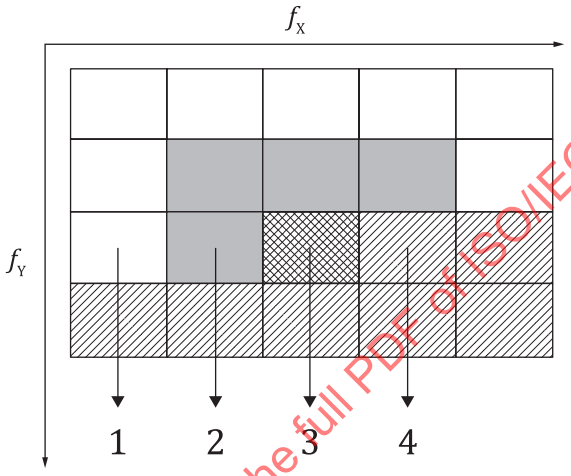
Each component in the QBP shall be assigned a context structure as given below, where the occurrence of some symbol belonging to the component is assigned to some context in the structure, with each context having its own independent cumulative count vector. After encoding/decoding the symbol, the cumulative counts of the encountered context is updated with the alphabet of the symbol. In particular

— $CCbd[kbd]$,where $kbd \in \{0,1,\ldots(BDqcd+1)^4-1\}$, stores the context information of QBP bit depth

— $CCX[kX]$, where $kX \in \{1,\ldots BDqcd\}$, stores the context information of QBP range

— $CCcf[kcf]$, where $kcf \in \{1,\ldots BDqcd\}$, stores the context information of QBP coefficients

kX and kcf are obtained simply as the current QBP bit depth. kbd is the 4-tuple representing the 4 QBP bit depth neighbours belonging to the same TB shown in Figure C.3



Key

1 previously decoded QB bit depth not used for context
2 previously decoded QB bit depth used for context
3 current QB bit depth being encoded
4 QB bit depths not yet encoded

Figure C.3 — Determination of context *kbd* (indicated by the 4-tuple in grey region) from previously encoded/decoded QB bit depths belonging to the same TB

The main routine used for entropy coding at the encoder is given in Table C.3. The cumulative counts used for coding the QB component symbols are all initialized, as shown in Table C.4. The symbols are encoded using Table C.5.

Table C.3 — FPAE Procedure used for entropy encoding a code block

`Procedure FPAE (cb){`	See also Tables C.4, C.5, I.1 and I.4.
`AE = InitializeArithmeticEncoder();`	Initialize arithmetic encoder for the code block.
`CCCB = InitializeCountsCB();`	
`for (qbp : cbp) {`	Loop over QBs belonging to code block
`  k = cbp.FetchBitdepthContext(qbp);`	Fetch bit depth pattern of current QB from neighbouring QBs as shown in Fig. C.3
`  EncodeSymbolNB(qbp.b, CCCB.CCbd[k], AE);`	Encode current QB bit depth

Table C.3 *(continued)*

EncodeSymbolNB(qbp.Xint, CCB.CCX[qbp.b], AE);	Encode current QB range
for (s : qbp.qbint){	Loop over STFT coefficients belonging to QBP
EncodeSymbolNB(s, CCCB.CCcf[qbp.b], AE);	Encode STFT coefficient
}	
}	
AE.DoneArithmeticEncoding();	Terminate arithmetic encoding for code block.
return AE.bitstream;	
}	

Table C.4 — Procedure used for initializing histogram counts for a code block

Procedure InitializeCountsCB (){	See also Table I.8.
for (k = 0; k < pow(BDqcd+1,4); k++){	Loop over all possible neighbour bit depth patterns for QBP bit depth
CCCB.CCbd[k] = InitializeCounts(BDqcd+1);	Initialize cumulative counts for encoding QBP bit depth.
}	
for (k = 1; k <= BDqcd; k++){	Loop over all non-zero bit depths
CCCB.CCX[k] = InitializeCounts(1 << QT.q[k]);	Initialize cumulative counts for encoding QB range.
CCCB.CCcf[k] = InitializeCounts(1 << k);	Initialize cumulative counts for encoding QB STFT coefficients.
}	
return CCCB;	
}	

Table C.5 — Procedure used for encoding non-binary symbol

Procedure EncodeSymbolNB (s,CC,AE){	See also Tables I.3 and I.9
AE.EncodeSymbol(s,CC);	Encodes symbol with probabilities for its alphabet dictated by cumulative count
CC.UpdateSymbol(s);	Updates cumulative count with symbol occurrence
}	

C.2 Decoding

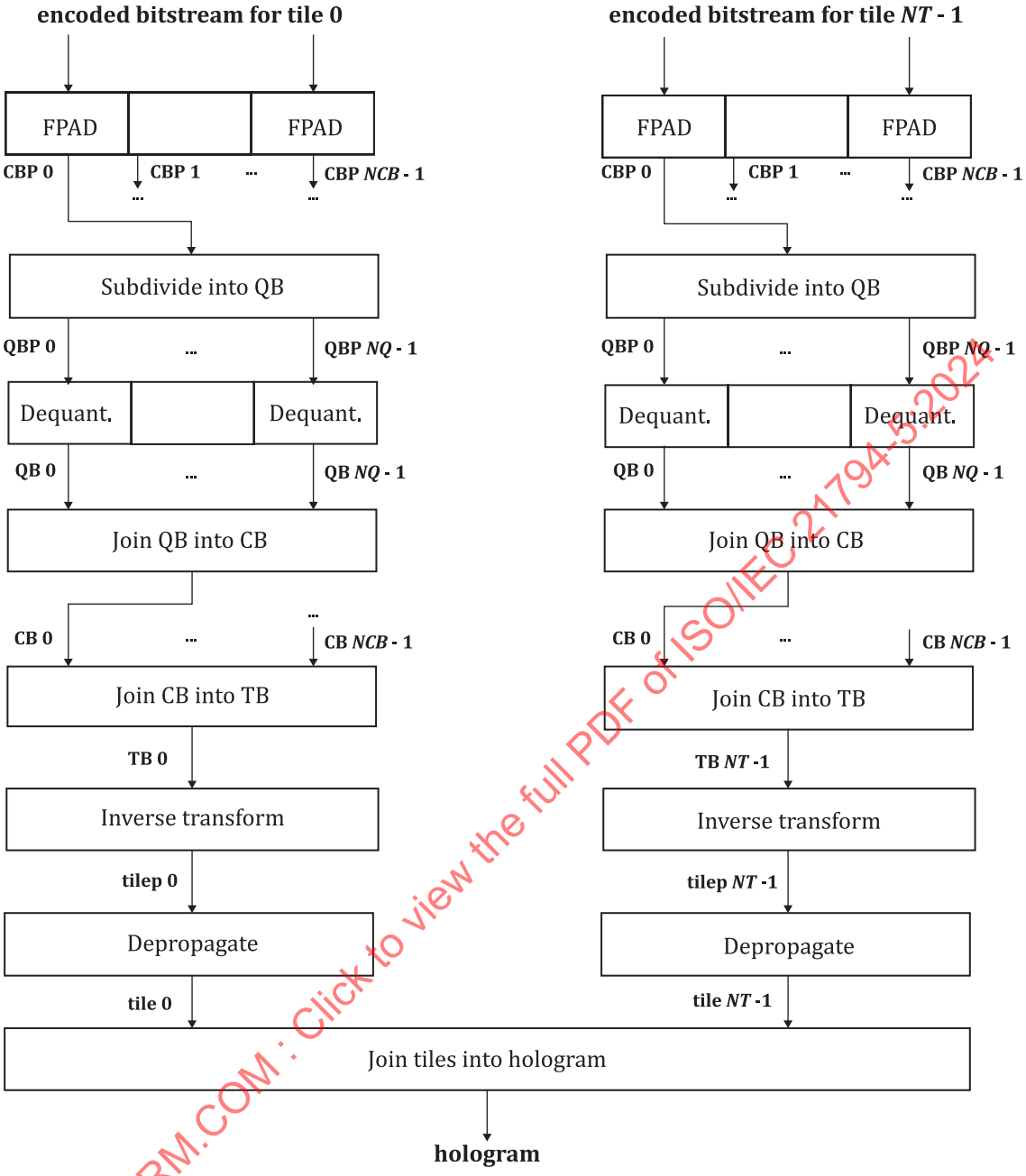


Figure C.4 — Overview of decoding for non-binary holograms when all code blocks are available

The decoder operates on one or more code blocks independently, where each code block represents a contiguous 4D hyperrectangle of the transform coefficients belonging to a code block, as discussed in Annex E. An overview of the decoding procedure is given in Figure C.4 illustrates the decoding procedure. Codeblocks are independently decodable, enabling random access.

During decoding, the steps to be performed are essentially the inverses of the encoding steps described in Annex section C.1 performed in reverse order. Table C.6 describes the main decoding procedure for non-binary holograms.

Table C.6 — Main decoding procedure for lossy non-binary holograms

Procedure Decoding (Codestream){	See also Tables C.7 and C.9.
for (tilestream : Codestream){	For every tile codestream segment
tb = {}; tilep = {};	Empty initializations
for (cbstream : tilestream){	Loop over every code-block stream data
cbp = FPAD(cbstream);	Do entropy decoding, ...
cbcoeffs = Dequantization(cbp);	... dequantize to transform coefficients ....
tb.append(cbcoeffs);	... and append to the transformed tile.
}	
for (tb_slice : tb) {	Loop over every TB slice
wb = inverse_transform(tbp);	Inverse transform (e.g. 2D-IDFT) ...
tilep.append(wb);	... and append to the propagated tile.
}	
tile = inverse_propagate(tilep);	Inverse tile propagation
decoded_hologram.append(tile);	Append tile to the decoded hologram

C.2.1 Entropy decoding

Entropy decoding is to be applied on the arithmetic encoded bitstream belonging to the CB. This procedure will return the QBPs (quantization-block payload) corresponding to the QBs contained in the code block, cf. Table C.7. As discussed in Annex section C.1.3, the QBP consists of the following integer-valued components – QB bit depth, quantized QB range, and the quantized QB coefficients. The QBP components are assigned cumulative count vectors $CCbd$, CCX and $CCcf$ respectively, which are initialized and updated in exactly the same way as the encoder discussed in Annex C.1.4, so that the same cumulative count that was used by the fixed-point arithmetic encoder for any given symbol will be used by the fixed-point arithmetic decoder. Thus, the order of processing of the QBPs within the CB and the transform coefficients within the QBP shall be the same for the entropy encoder and entropy decoder.

The main routine used for entropy decoding is given in Table C.7, while Table C.8 contains the subroutine for decoding a QB component symbol.

Table C.7 — Procedure used for entropy decoding a code block

Procedure FPAD (cb){	See also Tables C.4, C.8 and I.5.
AD = InitializeArithmeticDecoder(cb.bitstream);	Initialize arithmetic decoder for the code block.
CCCB = InitializeCountsCB();	
for (qbp : cbp) {	Loop over QBPs belonging to code block
k = cbp.FetchBitdepthContext(qbp);	Fetch bit depth context of current QB from previously decoded QBs
qbp.b = DecodeSymbolNB(CCB.CCbd[k],AD);	Decode bit depth of current QB
if(cb.qt.q[qbp.b] == 0){	If QB range bit depth is zero
qbp.Xint = NULL;	
}	
else{	Decode QB range for non-zero bit depths
qbp.Xint = DecodeSymbolNB(CCX[qbp.b],AD);	
}	
for (s : qbp.qbint){	Loop over STFT coefficients belonging to QB
if (qbp.b == 0){	
qbp.qbint.append(NULL);	

Table C.7 *(continued)*

}	
else{	
qbp.qbint.append(DecodeSymbolNB(CCcf[qbp.b],AD));	
}	
}	
}	
return CBP;	
}	

Table C.8 — Procedure used for decoding non-binary symbol

Procedure DecodeSymbolNB (CC,AD){	See also Tables I.6 and I.9
s = AD.DecodeSymbol(CC);	Decodes symbol
CC.UpdateSymbol(s);	Updates cumulative count with symbol occurrence
return s;	
}	

C.2.2 Dequantization

After obtaining the QBPs associated with the code-block dequantization are applied as shown by Table C.9 for a singular codebook.

Table C.9 — Dequantization procedure

Procedure Dequantization (CBP){	See also Table H.2.
cbcoeffs = {};	
for (qbp : cbp){	Loop over all QBPs in CBP
b = qbp.b;	Obtain bit depth of QBP
Xhat = MRDQ(qbp.Xint,cbp.qt.q[b], cbp.qt.Q[b]) + cbp.qt.Qoff[b];	Obtain quantization range of QBP
qbdot = MRDQ(qbp.qbint, b, Xhat);	Obtained dequantized QB STFT coefficients
cbcoeffs.append(qbdot);	Append QB coefficients to decoded CB
}	
return cbcoeffs;	
}	

C.2.3 Inverse short-time Fourier transform

The TB can completely be reverted to the original tile coefficients using inverse DFTs to compute the inverse Short-time Fourier transform. Definitions are also given in Annex G.

C.2.4 Depropagation

Every tile can be propagated back with numerical diffraction operators, undoing the propagation operator. Propagation is a highly symmetrical operator, as detailed further in Annex F on light propagation models.

Annex D
(normative)

Binary lossless coding

D.1 General

Binary holograms exhibit significant local spatial correlations that can be leveraged for compression. The underlying coding mechanism assumes a quasi-stationary process for describing the correlation between neighbouring pixels. It relies on the principle that when the hologram is viewed at an appropriately chosen spatial scale, the hologram behaves approximately in a stationary manner.

For capturing the spatial correlation, the coding mechanism utilizes a Markov model, similar to the one used in ISO/IEC 11544 [1] and 14492 [2] for bilevel image coding, where the probability state of the pixel being coded is determined from the value and relative positions of neighbouring pixels that belong to a causal template. Although plain Markov models permit describing all classes of finite state models for stationary signals, they also require a large number of prior observations to obtain reliable probability estimates, especially when the model size gets larger.

The coding mechanism utilizes a tree-based model comprising several Markov models of different sizes, starting from a depth 1 up to a depth $DepBC$. Out of all available models, a single model will be used to code the given current pixel. In the beginning, smaller models are preferred, and larger models are selected as more of the hologram gets processed. The construction of these models is uniquely described by an ordered list $Order = \{p_1, .., p_{DepBC}\}$ of size $DepBC$, containing the relative positions of the most correlated neighbouring pixels, given in decreasing order of their predictive power. The Markov model of depth d is then constructed from the first d neighbouring pixels in $Order$, and represented using a cumulative count structure $CCD[d].CC[kd]$, where $d \in \{1, .., DepBC\}$ and $kd \in \{0, .., 2^d - 1\}$. Here, the context kd represents any of the possible unique d-pixel neighbourhoods, consisting of a binary pattern. The cumulative counts $CC[kd]$ keep track of the number of 0 and 1 occurrences of a pixel for a given context kd encountered so far in the encoding/decoding process for the current tile.

The cumulative counts are initialized with an occurrence of 1 for both alphabets, for all contexts and model depths. After determining the best model depth dop to be used, the current pixel is encoded/decoded using binary arithmetic coding with the cumulative counts corresponding to the encountered context in $CCD[dop]$.

For any given pixel, the determination of the model depth with the best coding efficiency dop is done based on expected rate comparisons. The expected rate is calculated using the available observations in the cumulative counts, assuming no a priori information about the hologram. The search procedure starts by comparing the coding performance of the context model at the largest depth of the tree with that of the depth level just above (its parent). If the coding performance of that parent is better, it will now recursively be compared in turn with its parent; this search procedure is terminated once a node with a better coding performance than its parent is encountered or when it reaches the top of the tree, that is, the smallest possible depth of 1. When comparing and evaluating the coding efficiency of a parent node, both of its children are utilized, irrespective of which child context has been currently encountered.

D.2 Encoding

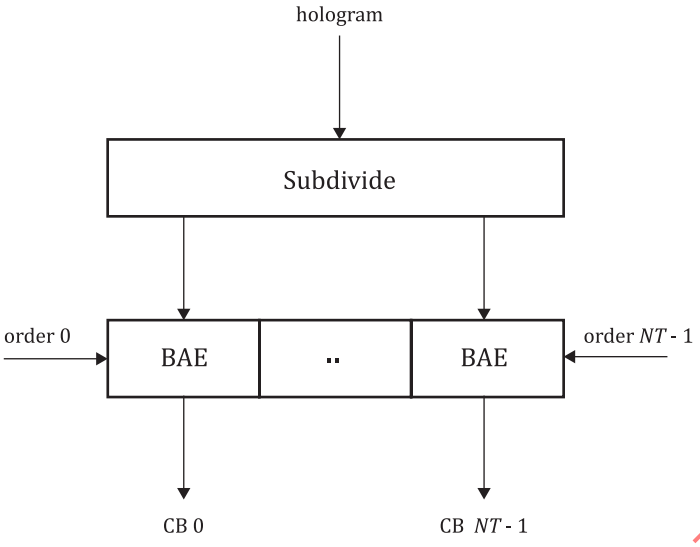


Figure D.1 — Overview of encoding for binary holograms

In binary lossless encoding, the binary hologram is subdivided into NT tiles, where $NT = \left\lceil \frac{\text{Xhoc}}{\text{XThoc}} \right\rceil \times \left\lceil \frac{\text{Yhoc}}{\text{YThoc}} \right\rceil$. Each tile is encoded independently utilizing binary arithmetic coding as shown in Figure D.1. Each tile requires as additional input the order vector that describes the construction of the model tree discussed previously for the tile. Each tile corresponds exactly to one codeblock in this case.

The main encoding procedure is detailed in Table D.1. The tree-based Markov model is initialized as shown in Table D.3, and updated using Table D.4. The coding efficiency comparisons between the different depths of the tree is to be conducted as given in Table D.2, where `HB` is the binary entropy function given by

$$H_B(x) = -x\log_2 x - (1-x)\log_2(1-x) \text{ for } 0 \leq x \leq 1$$

Table D.1 — Entropy encoding of a binary tile

Procedure BAE (tile){	See also Tables D.2, D.3, D.4, I.1, I.3 and I.4
`AE = InitializeArithmeticEncoder();`	Initialize arithmetic encoder for the tile and code blocks
`CCD = InitializeContextsBinary();`	Initialize the context count tree (see Table D.3)
`for (context : tile){`	Fetch contexts of each tile pixel in raster-scan order
`  for (d=DepBC; d>=0; d--){`	Determine best depth
`    subcontext_ = truncate(context, d);`	Get d most significant positions of binary pixel values from context
`    lc = CCD[d].CC[append_bit(subcontext, 0)];`	Get left child context count for the subcontext, with 0 added for the next most significant pixel (d+1) …
`    rc = CCD[d].CC[append_bit(subcontext, 1)];`	… and analogously, right child context count with 1 appended
`    if (check_delta_sign(lc, rc)) break;`	Check whether best depth is found
`  }`	
`  s =  get_current_pixel();`	

Table D.1 *(continued)*

AE.encode_symbol(s, CCD[d+1].CC[get(truncate(context, d+1))];	Encode symbol (current pixel) taking into account the relevant context counts.
CCD.update_counts_binary(context,s);	Update occurrence for all depths
}	
AE.DoneEncoding();	
codestream.append(AE.bitstream);	
}	

Table D.2 — Comparing the coding efficiency of a parent node with its child nodes in the tree-based adaptive context model

Procedure check_delta_sign(lc, rc){	
denom = lc[0] - rc[0] - 2;	Intermediary denominator variable
delta = HB((lc[1] - rc[1] - 1)/denom) - HB(lc[1]/lc[0]) * (rc[1] - 1)/denom - HB(rc[1]/rc[0]) * (rc[0] - 1)/denom;	Compare coding efficiency of parent node with respect to children nodes
return (delta > 0);	Return whether children or parent are better
}	

Table D.3 — Initialize cumulative counts for all depths in tree

Procedure InitializeContextsBinary (){	See also Table I.8.
for (d=1; d <= kDepBC; d++){	Loop over all depths
for (c=0; c < (1<<d); c++){	Loop over all possible contexts in depth
CCD[d].CC[c]= InitializeCounts(2);	Initialize counts for binary alphabet
}	
}	
return CCD	
}	

Table D.4 — Update cumulative counts for all depths in tree

Procedure CCD.**update_counts_binary**(context,s){	See also Table I.9.
for (d=1; d <= kDepBC; d++){	Loop over all depths
subcontext = truncate(context, d);	Fetch occurred context at depth d
CCD[d].CC[subcontext]. UpdateCounts(s);	Update cumulative counts of context
}	
}	

D.3 Decoding

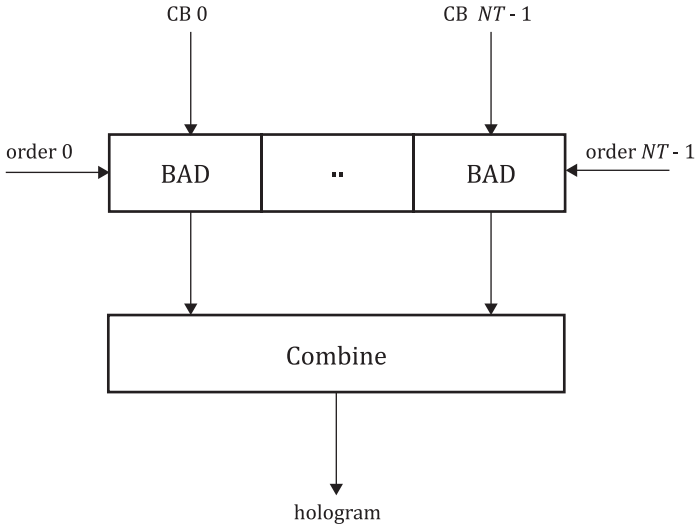


Figure D.2 — Overview of decoding for binary holograms

The decoding starts from one or more codeblocks, which is then subject to binary arithmetic decoding, as shown in Figure D.2 to recover the information in the corresponding tiles. The same model that was used in encoding is to be used in an identical manner, and the main procedure is given in Table D.5.

Table D.5 — Entropy decoding of a binary tile

Procedure BAD (cb){	See also Table D.2, D.3, D.4, I.5 and I.6.
AD = InitializeArithmeticDecoder(cb.bitstream);	Initialize arithmetic encoder for the tile and code blocks
CCD = InitializeContextsBinary();	Initialize the context count tree
for (context : tile){	Fetch contexts of each tile pixel in raster-scan order
for (d=DepBC; d>=0; d--){	Determine best depth
subcontext = truncate(context, d);	Get d most significant positions of binary pixel values from context
lc = CCD[d].CC[append_bit(subcontext, 0)];	Get left child context count for the subcontext, with 0 added for the next most significant pixel (d+1) ...
rc = CCD[d].CC[append_bit(subcontext, 1)];	... and analogously, right child context count with 1 appended
if (check_delta_sign(lc, rc)) break;	Check whether best depth is found
}	
s = AD.decode_symbol (CCD[d+1].CC[truncate(context, d+1)]	Decode symbol (current pixel) taking into account the relevant context count.
CCD.update_counts_binary(context,s);	Update context counts with occurrence
tile.append(s);	Update tile with pixel
}	
}	

Annex E
(normative)

Hologram tiling

E.1 Hologram parametrization

A digital hologram H is represented as one or more two-dimensional array(s) H_0, H_1, … . In this document, x is the symbol of the first hologram dimension. It enumerates the columns. y is the symbol of the second hologram dimension and enumerates the rows. One two-dimensional array H_i is defined per wavelength λ_i. The number of wavelengths is signalled in the codestream as NC. The size of each array H_i is signalled as $[X_{hoc}, Y_{hoc}]$. Without loss of generality, a digital hologram may, therefore, be considered as a non-empty set of two-dimensional arrays, all with the same dimensions. This is illustrated in Figure E.1.

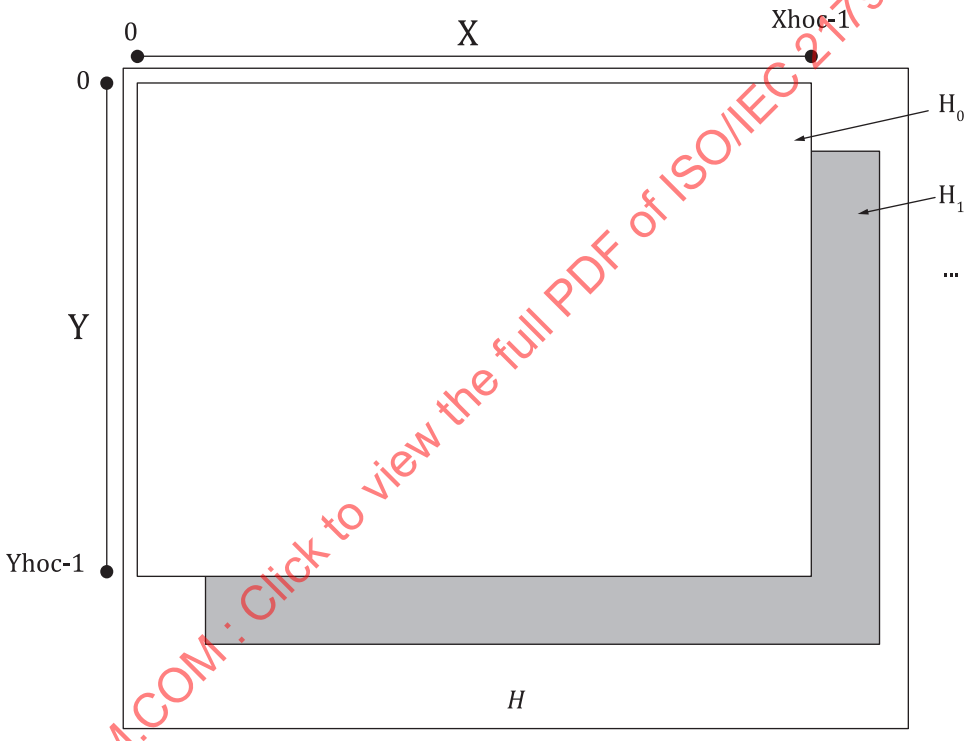


Figure E.1 — Coordinate system and composition of a digital hologram *H*

Note For the hologram and the transform blocks, code blocks, and quantization blocks, x and y are used for, respectively, the horizontal and vertical spatial coordinates. Please observe that the spatial coordinate systems differ for the hologram and the transform, code, and quantization blocks; hence, their x and y coordinate values are unrelated and should be considered independent values. However, for reasons of readability, this simple notation was preferred.

E.2 Hologram segmentation

Each digital hologram may be segmented in multiple steps. Each step with identical segmentation parameters for all wavelengths. We may, therefore, speak of a singular two-dimensional array H_i in this clause. The subsequent subclauses describe the segmentations listed in Table E.1. As a rule of thumb, serialization will always occur along the dimensions 1, 2, 3, and 4 in that order. Note, in C/C++, this corresponds to a "row-major" order.

Table E.1 — Hologram segmentation stages overview.

Segmentation	Purpose	Subclause
holograms → tiles	Control memory requirements	E.2.1
TB → code blocks (CB)	Control of minimal unit of random access in the codestream. Also: control maximal supported parallelization and multi-threading of codec.	E.2.2
For lossy, non-binary only: CB → quantization blocks (QB)	Control trade-off between systems 1 and 2 (see Annex H) of the adaptive quantizer.	E.2.3

E.2.1 Segmentation into tiles

First, a set of two-dimensional arrays of a colour digital hologram can be partitioned in the native hologram domain using a two-dimensional block size. One segment, which is identical across all colour channels, is called (colour) tile. Each colour channel of a tile is called a "tile channel" and may be partitioned further independently of other tile channels into Transform-/Code-/Quantization blocks.

The number of colour components per hologram/tile is encoded as NC in the HOC, see Table B.5. The (spatial) block size of a tile is signalled as [*XThoc* , *YThoc*] in the codestream, see Figure E.2.

The tile size controls the maximal amount of hologram coefficients processed at once. The tile size allows to trade off requirements of random access with rate-distortion performance. For example, the tile size is relevant during the global rate distortion upon encoding or during the STFT transform.

One tile channel (see B.5.11) per wavelength is required, and all colour tiles can be processed simultaneously and independently. All tile channels within one tile may be processed independently or jointly. The following description of the partitioning will thus only consider a single tile channel.

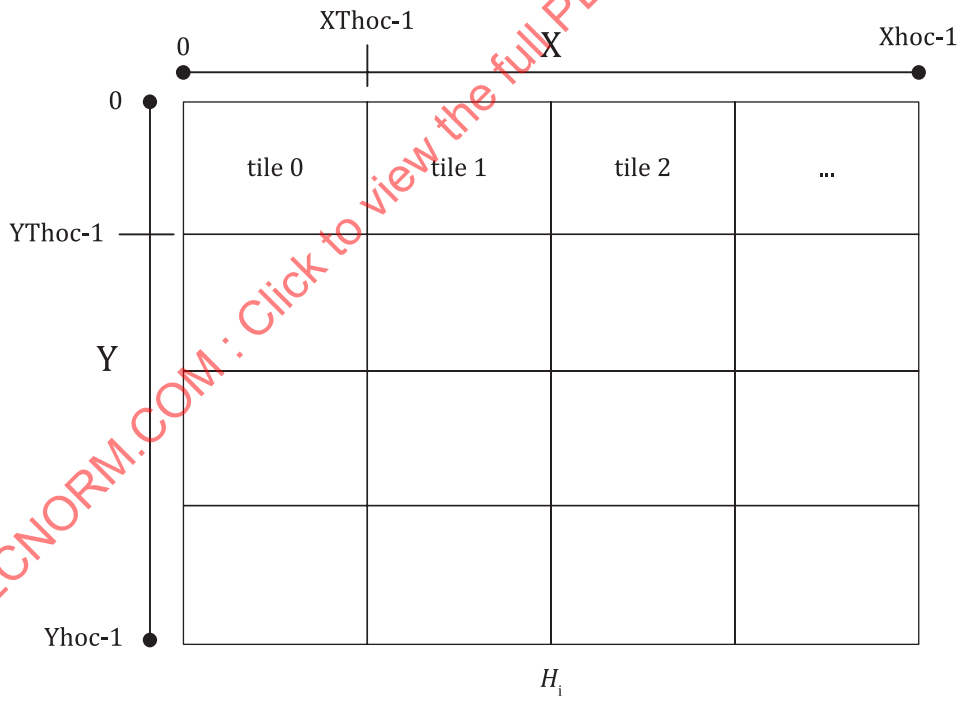


Figure E.2 — Coordinate system and partition of a monochrome digital hologram H_i into (monochrome) tiles

E.2.2 Partitioning tiles into window blocks (WB) and transformation into transform blocks (TB)

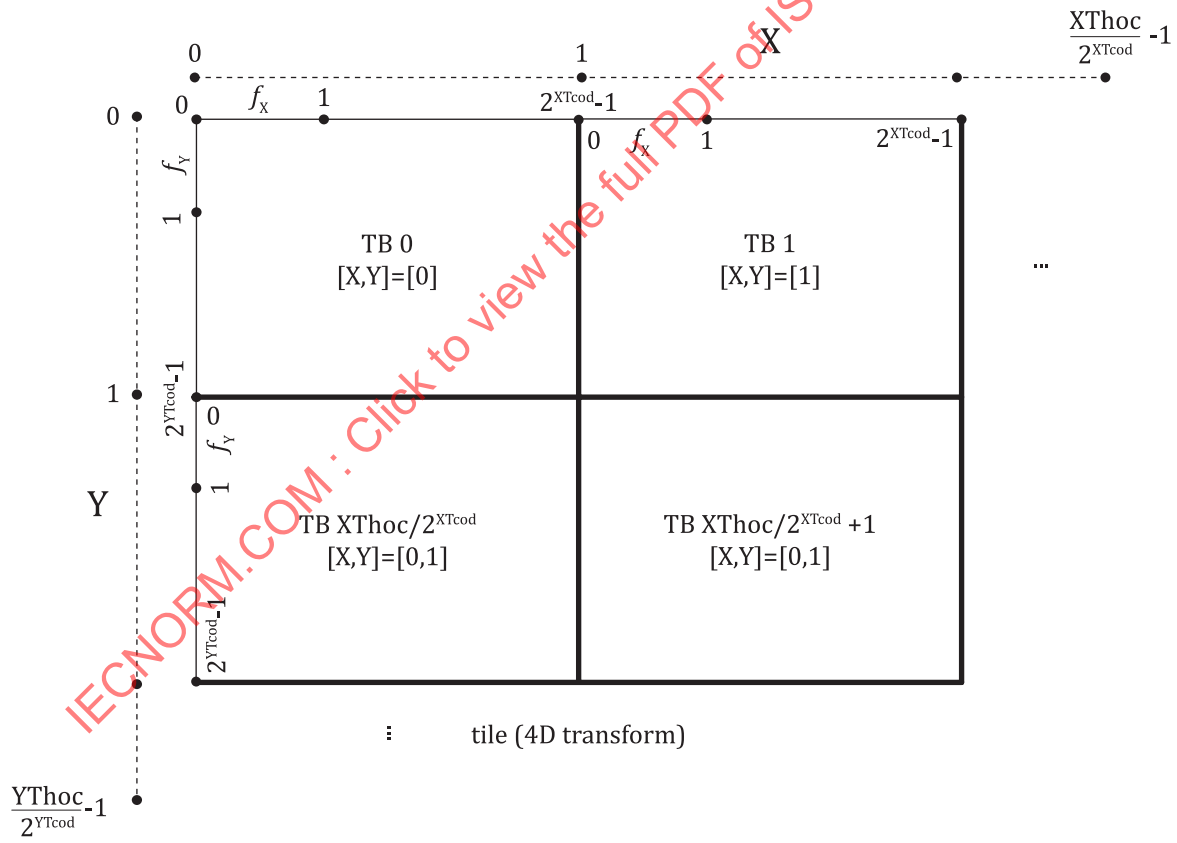
The two-dimensional array of a tile channel can be subdivided into different rectangular views, called window blocks, in preparation for the subsequent transform step. For the rectangular window STFT, this

amounts to a partition of the native hologram domain into non-overlapping two-dimensional WB. The transform size is specified in the codestream as $\left[2^{XTcod},2^{YTcod}\right]$. The transform size trades off resolution in the native hologram domain with its Fourier domain. Larger transforms result in more frequency bands but also in a lower spatial resolution. The WBs may be transformed independently of each other but may possess a joint interpretation. These are recombined into a 4D TB with two spatial and two frequency dimensions.

A tile of size $[XThoc,YThoc]$ can be split into $\left[\frac{XThoc}{2^{XTcod}},\frac{YThoc}{2^{YTcod}}\right]$ disjoint WBs using a transform of size $\left[2^{XTcod},2^{YTcod}\right]$. Each transformed tile may therefore be interpreted as a four-dimensional TB of size $\left[2^{XTcod},2^{YTcod},\frac{XThoc}{2^{XTcod}},\frac{YThoc}{2^{YTcod}}\right]$. A TB, resulting from the factorization of a tile using two-dimensional transforms, can thus always be reinterpreted as a four-dimensional arrangement of coefficients.

If the STFT transform is used, the dimensions correspond to (TB) positions *x*, *y* (in the native hologram domain) and spatial frequencies f_x, f_y as follows $[f_x, f_y, x, y]$, see Figure E.3. This joint four-dimensional interpretation will be used in the following in the lossy, non-binary coding pipeline.

This notation is used irrespective of the hologram type. For example, for a Fourier hologram, the native hologram domain corresponds to the Fourier domain. Thus, the actual interpretation of spatial frequency and "spatial" dimensions would have to be switched. This specification considers a digital hologram to be a general signal, and hence, spatial frequencies and positions refer to its characteristics.

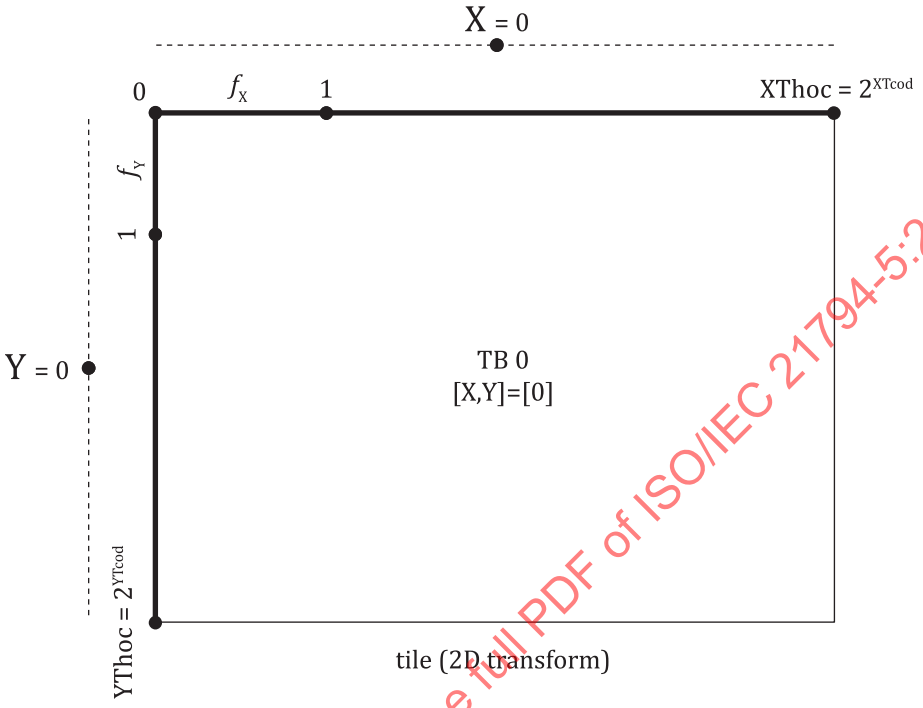


The first (top-left) tile is shown in its 4D interpretation induced by a non-degenerate 2D transform. The location in the native tile domain is indexed by [x,y]. For a specific location [x,y], the transform coefficients are indexed by $[f_x, f_y]$.

Figure E.3 — Exemplary arrangement STFT transform

If no transform is used, such as in the lossless, binary pipeline, the transform size is the same as the tile size and the degenerate four-dimensional TB size is $\left[2^{XTcod},2^{YTcod},1\equiv 2^{UTcod},1\equiv 2^{VTcod}\right]=[XThoc,YThoc,1,1]$. It is designated as "degenerate" since two of the TB dimensions have size 1.

The binary processing pipeline works with such a degenerate four-dimensional transform size. For this sub-section consider a "no transform" to be modelled as the identity transform. The arrangement of one "TB" per tile is shown in Figure E.4.



The first (top-left) tile is shown after transform with a degenerate 2D transform. The location of the transform block is indexed by [x,y]. The coefficients are indexed by [f_x, f_y].

Figure E.4 — Exemplary arrangement of one "TB" per tile

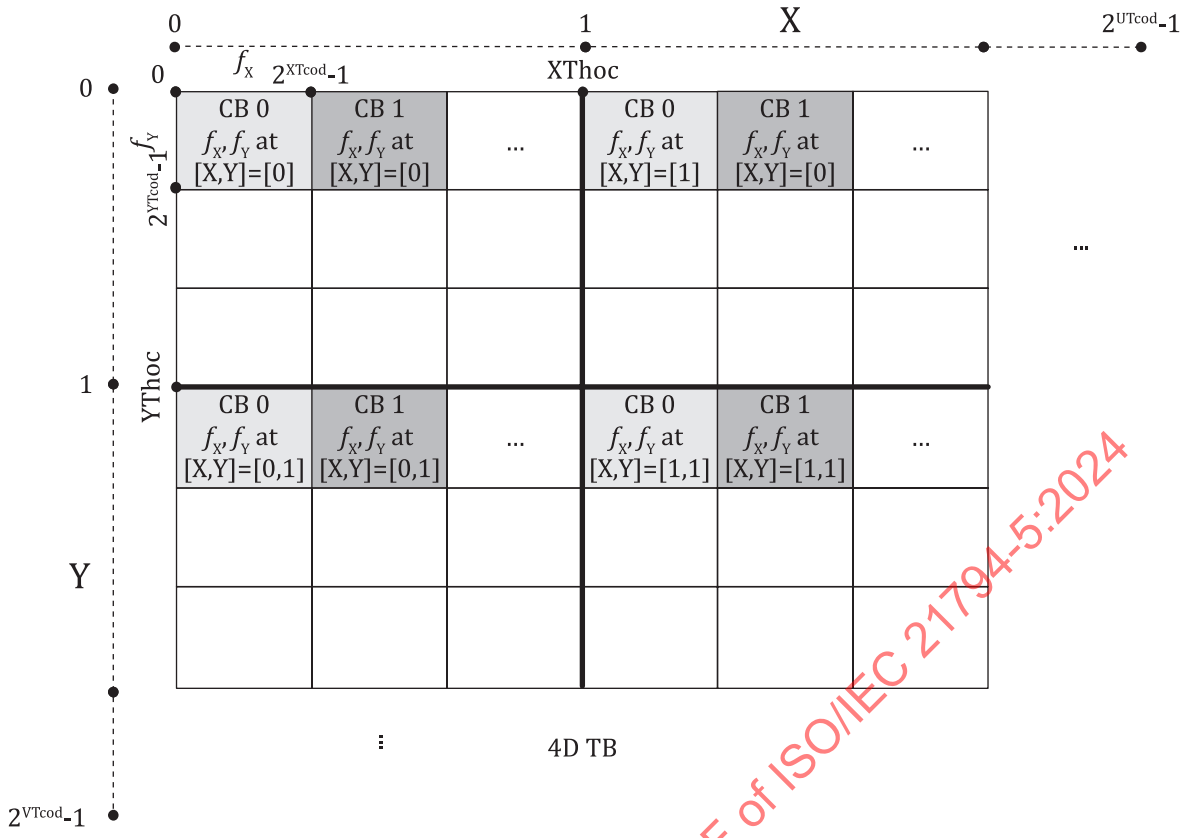
For an arbitrary two-dimensional transform, a four-dimensional TB structure as described in E.2.2 arises, but each TB may be processed/interpreted independently. The arrangement of TB per tile is identical with E.2.2 shown in Figure E.5.

E.2.3 Segmentation into code blocks (CB)

A four-dimensional TB of size $\left[2^{XTcod},2^{YTcod},2^{UTcod},2^{VTcod}\right]=\left[2^{XTcod},2^{YTcod},\frac{XThoc}{2^{XTcod}},\frac{YThoc}{2^{YTcod}}\right]$ can be disjointly split over multiple four-dimensional CBs related to a particular spatial frequency band and x, y spatial coordinate range. The CB size is specified in the codestream as $\left[2^{XCBcod},2^{YCBcod},2^{UCBcod},2^{VCBcod}\right]$. The interpretation of the four dimensions is as with a TB: $[f_x, f_y, x, y]$.

A single CB is guaranteed to be coded independently and, therefore, the minimal random access unit in the compressed bitstream. The CB size trades off the size of the minimal random-access unit with compression performance. It also controls the number of maximal possible work threads for en-/decoding as each CB can be en-/decoded independently.

The CB size should factor the TB size. A degenerate TB size, e.g., $\left[2^{XTcod},2^{YTcod},1\equiv 2^{UTcod},1\equiv 2^{VTcod}\right]$, will require the CB size to be degenerate in the last two dimensions, too. This is the case for the lossless binary coding. An exemplary arrangement for a non-degenerate TB partition is shown in Figure E.5.



The 4D CB partition of the first (top-left tile equivalent) 4D TB is shown. Note that each 4D CB consists of multiple 2D CB slices. In the arrangement shown, the 2D slices are grouped in spatial frequency and indexed by their spatial position. The [x,y] coordinates have a different offset for other tiles.

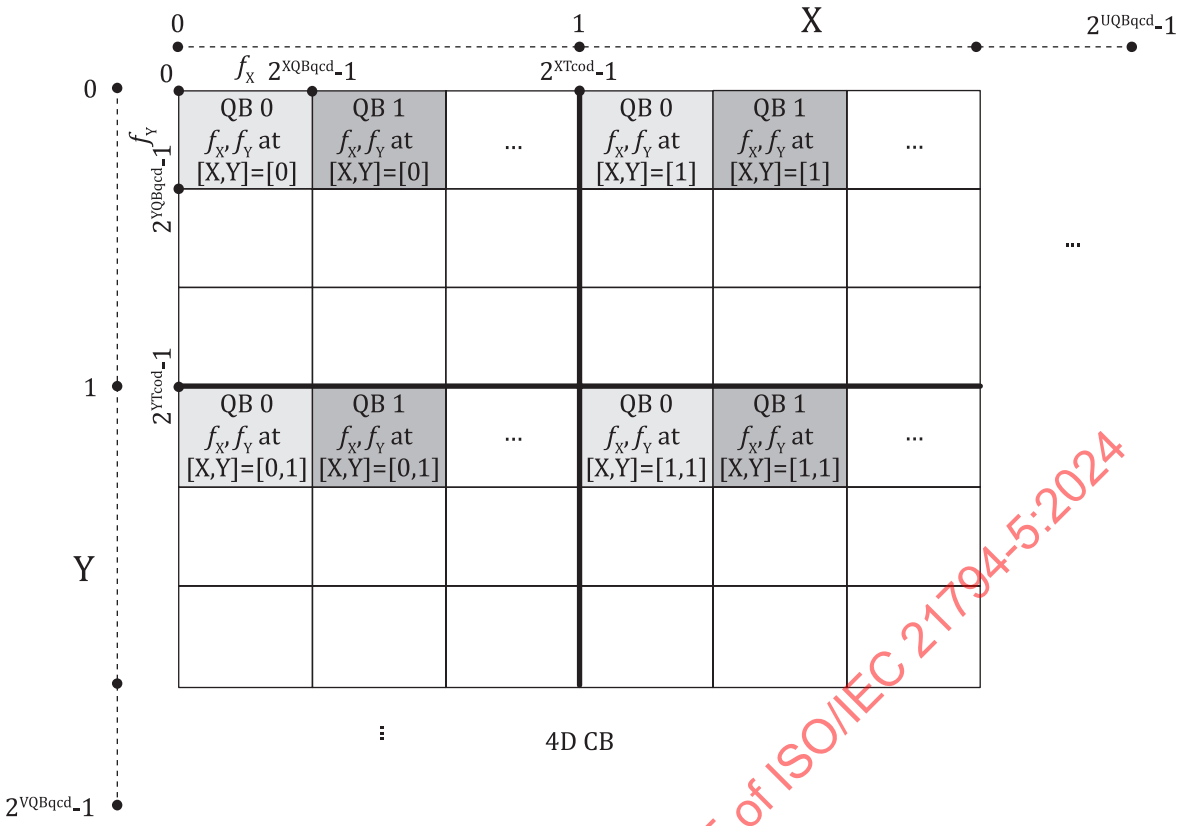
Figure E.5 — Exemplary arrangement for a non-degenerate TB partition

E.2.4 Segmentation into quantization blocks (QB) – non-binary, lossy coding only

In the case of non-binary, lossy coding, a four-dimensional CB can be disjointly split up into QBs. In the codestream the QB size is specified as $\left[2^{XQBqcd}, 2^{YQBqcd}, 2^{UQBqcd}, 2^{VQBqcd}\right]$. The interpretation of the four dimensions is as with TB and CB: $[f_x, f_y, x, y]$.

All coefficients within one QB are quantized to the same bit depth using the double adaptive quantizer. In the encoder, rate-distortion optimization may be performed on the level of per QB rate-distortion performance as a function of its bit depth. The bit depth in a QB may be 0 or any integer value smaller than the maximal supported bit depth in the implementation. For more details, see Annex G - Quantization.

The QB size should factor the CB size. An exemplary arrangement is shown in Figure E.6.



The 4D QB partition of the first (CB in the first, top-left tile) 4D CB is shown. Note, as with the CB each 4D QB consists of multiple 2D QB slices. In the arrangement shown the 2D slices are grouped in spatial frequency and indexed by their spatial position. For other CBs, tiles, the *[x,y]* coordinates have a different offset.

Figure E.6 — Exemplary arrangement into quantization blocks – non-binary, lossy coding only

E.3 Implicit padding

By convention, the four-dimensional TB size, CB size, and QB size are required to be integer multiples of each other (in each dimension) in the specified order, that is, a TB is completely partitioned into CB without overlapping nor missing coefficients. The same requirement holds for the two-dimensional tile and transform sizes. Note that due to the codestream syntax of signalling QB and CB sizes, either one shall be a positive integer power of 2 or 0.

If the two-dimensional size of any monochrome hologram is not an integer multiple of the tile size, zero padding is applied to the monochrome hologram at the end of the row/column ranges to the next full tile size. That is, a [1920,1080] size monochrome hologram to be encoded with a [512,512] tile size will be padded on the right and bottom such that it has size [2048,1536]. An example is shown in Figure E.7.

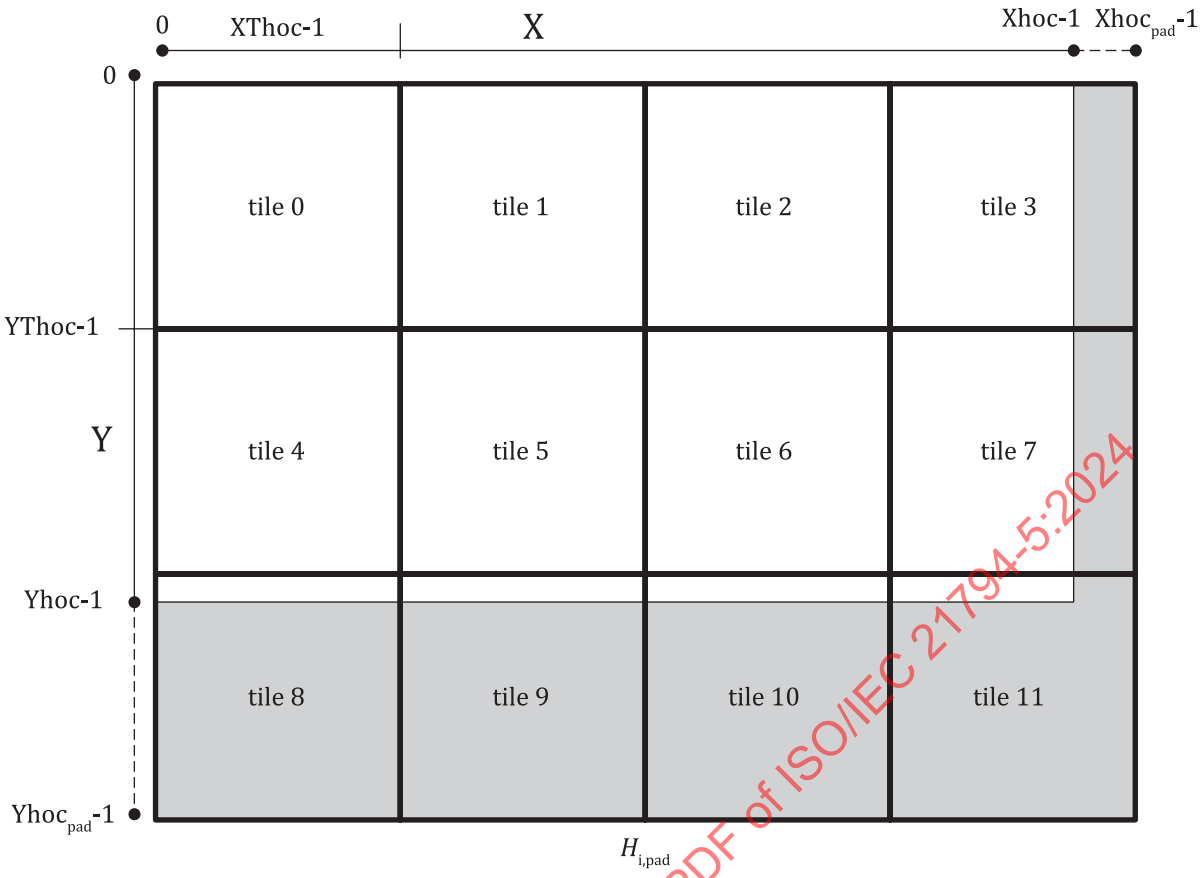


Figure E.7 — Monochrome hologram zero padding (grey area) to next integer multiple of tile size per dimension

For the binary processing pipeline(s) an extra padding rule applies. Because access to individual data bits is inefficient (due to processor words being larger), the data is implicitly assumed to be packed as 64 bits per first dimension (i.e. number of columns). In other words, the first dimension (x) of a CB in the binary processing pipeline shall be a multiple of 64.

Therefore, all sizes, including the two-dimensional tile size, are multiples of 64 in the column dimension *x*. The padding to multiples of 64 bits per row is therefore implicit with the rules outlined above.

Annex F
(normative)

Light propagation models

F.1 Rationale

Light propagation models mathematically describe how the wavefield of light evolves as it propagates over space. These consist of numerical diffraction operators, which can calculate the value of the wavefield on some target surface(s) given source surface values as input. These are often used for computing the propagation between parallel source and target planes, where the operators are numerically reversible.

Light propagation is a core element in computer-generated holography, enabling the computation of digital holograms of virtual scenes by modelling how light interacts with virtual object surfaces and propagates over space. These models have been shown to be useful for hologram compression as well for two main reasons: (1) sparsification and (2) conversion to a universal complex-valued wavefield representation.

Starting from the recorded or computed hologram, one can bring an object into focus by backpropagating the wavefield to the distance of the object with respect to the hologram plane. This will concentrate the light into a smaller region and make the propagated wavefield appear more image-like, effectively promoting sparsity. Therefore, propagation models such as Fresnel diffraction have been utilized jointly with other transforms, such as cascaded with wavelets or the DCT, to significantly improve compression performance over standard unmodified image or video codecs.

Moreover, hologram recording (either optical or virtual) can happen under vastly different conditions. Depending on the optics and camera parameters, holograms can be recorded both in the spatial or the Fourier domain, with different effective pixel pitches depending on e.g., magnification, with objects at many distances from the hologram plane. By pre-processing the holograms with an appropriate light propagation operator, one can reversibly convert the hologram into a wavefield expressed in the spatial domain in a constrained pixel pitch range with similar object distance ranges, resulting in a more uniform intermediate representation format better conducive to efficient compression.

F.2 Mathematical definitions

Light propagation models are orthogonal operators on complex-valued signals. Starting from a two-dimensional array of complex-valued entries representing the (sampled) hologram, they transform this input signal mainly by a cascade of Discrete Fourier Transforms (DFT) and point-wise multiplication with phase-only signals, that is, entries whose complex magnitude is equal to 1. The Fourier transforms are defined as follows:

$\mathcal{F}$ Forward two-dimensional DFT.

$\mathcal{F}^{-1}$ Inverse two-dimensional DFT.

The 2D DFT transforms are separable into 1D DFTs, first applied along the rows and then along the columns. The 1D DFTs are defined for a n-sample 1D complex-valued input array a and same-size output array b. The forward 1D DFT transform is defined to be

$$b[k]=\sum_{j=0}^{n-1} a[j]\cdot w^{jk}$$

and the inverse 1D DFT transform is defined as

$$b[k]=\frac{1}{n}\sum_{j=0}^{n-1} a[j]\cdot w^{-jk}$$

where $w=\exp\left(-\frac{2\pi i}{n}\right)$ and i is the imaginary unit. These transforms can be computed efficiently using the Fast Fourier Transform (FFT), optionally in-place.

Because the signals are complex-valued, their DFTs are generally not conjugate symmetric. The frequency components have physical significance, making the diffraction equations more parsimonious when centering the zero-frequency component. For this purpose, we define the (I)FFTSHIFT operators

$\boldsymbol{S}$ Forward 2D FFTSHIFT operator

$\boldsymbol{S}^{-1}$ Reverse 2D FFTSHIFT operator.

separable into 1D FFTSHIFT operators along all rows and columns. The forward 1D FFTSHIFT is given as

$$b[j]=a\left[\left(j+\left\lfloor\frac{n}{2}\right\rfloor\right)\operatorname{umod} n\right]$$

and the reverse 1D FFTSHIFT is defined as

$$b[j]=a\left[\left(j+\left\lceil\frac{n}{2}\right\rceil\right)\operatorname{umod} n\right]$$

Both operators are identical whenever n is even. For some light propagation operators, the FFTSHIFT need not be explicitly computed, and can be absorbed into the point-wise multiplication operator.

Point-wise multiplication are defined using the Hadamard product operator $\odot$. The 2D array index coordinates x and y are zero-centered, so for a $m \times n$ sized hologram, the sample ranges are given by $x\in\left\{-\left\lceil\frac{m}{2}\right\rceil,-\left\lceil\frac{m}{2}\right\rceil+1,\ldots,+\left\lfloor\frac{m}{2}\right\rfloor\right\}$ and $y\in\left\{-\left\lceil\frac{n}{2}\right\rceil,-\left\lceil\frac{n}{2}\right\rceil+1,\ldots,+\left\lfloor\frac{n}{2}\right\rfloor\right\}$. So, for example, applying the operator $b=\left(x^2+y^3\right)\odot a$

will multiply every element of a by the sum of the square of its x index and the cube of its y index. Concretely, for an input 2D zero-indexed array a, we would get the output b as follows

$$b[j,k]=a[j,k]\cdot\left(\left(j-\left\lceil\frac{m}{2}\right\rceil\right)^2+\left(k-\left\lceil\frac{n}{2}\right\rceil\right)^3\right)$$

F.3 List of light propagation operators

Using the previous definitions, we can define multiple light propagation operators. Using canonical mathematical notation, operators are cascaded, starting with the input A and returning the output B, as illustrated in Figure F.1.

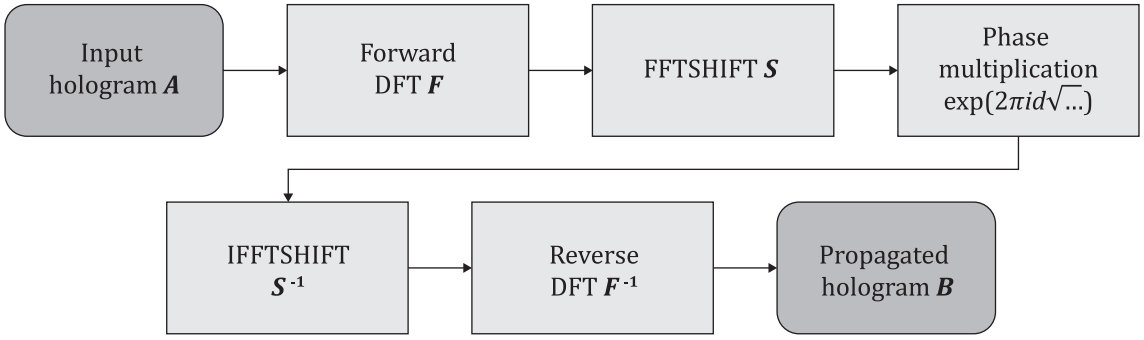


Figure F.1 — Example flowchart for the "Angular spectrum method" implementation.

In summary, these propagation operators are parameterized by the wavelength λ, the pixel pitch p, propagation distance d, and dimensions $m \times n$. They utilize zero-centred indexing coordinates (x, y); the Fourier operator $\mathcal{F}$, the FFTSHIFT operator $\boldsymbol{S}$, and their inverses.

Angular spectrum method

$$B = \mathcal{F}^{-1}\,\boldsymbol{S}^{-1}\left[\exp\left(2\pi i d\sqrt{\lambda^{-2} - \left(\frac{px}{m}\right)^2 - \left(\frac{py}{n}\right)^2}\right) \odot \boldsymbol{S}\mathcal{F}A\right]$$

Convolutional Fresnel transform

$$B = \frac{\exp\left(\frac{2\pi i d}{\lambda}\right)}{i\lambda d} \cdot \mathcal{F}^{-1}\left[\mathcal{F}\left(\exp\left(\frac{i\pi p^2}{\lambda d}\left(x^2 + y^2\right)\right)\right) \odot \mathcal{F}A\right]$$

Fourier-type Fresnel transform

$$B = \frac{\exp\left(\frac{2\pi i d}{\lambda}\right)}{i\lambda d} \cdot \exp\left(\frac{i\pi q^2}{\lambda d}\left(x^2 + y^2\right)\right) \odot \mathcal{F}\left[\exp\left(\frac{i\pi p^2}{\lambda d}\left(x^2 + y^2\right)\right) \odot A\right]$$

Where the output pixel pitch $q = \frac{p}{\lambda d}$ changes as a function of the distance d and wavelength λ.

Fourier-domain Fresnel transform

$$B = \frac{\exp\left(\frac{2\pi i d}{\lambda}\right)}{i\lambda d}\,\boldsymbol{S}\left[\mathcal{F}\left(\exp\left(\frac{i\pi p^2}{\lambda d}\left(x^2 + y^2\right)\right)\right) \odot \mathcal{F}A\right]$$

Fraunhofer transform

$$B = \mathcal{F}A$$

All listed operators are mathematically reversible and can be applied in the reverse order for decoding. The pairs $(\mathcal{F}, \mathcal{F}^{-1})$ and $(\boldsymbol{S}, \boldsymbol{S}^{-1})$ are each other inverses, and the pointwise multiplications with complex exponentials of the form $\exp(iz), z \in \mathbb{R}$ can be inverted by multiplying with their complex conjugate $\exp(-iz)$.

Annex G
(normative)

Short-time Fourier transform

G.1 Forward Discrete Short-time Fourier transform

The discrete STFT of a 2D input signal h results in a 4D output representation H with spatial dimensions (x,y) and frequency dimensions (u,v). For every frequency component, the signal is convolved with a window function g of dimensions $m \times n$, which can be, e.g., Gaussian, Hamming, Hann, or rectangular. This can be expressed as

$$H[x,y,u,v] = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} h\left[\tau_x x + j, \tau_y y + k\right] \cdot g[j,k] \cdot \exp\left(-i\left(\omega_u u j + \omega_v v k\right)\right)$$

where the carrier frequencies are rectilinearly sampled in Fourier space with periods ω_u and ω_v, and the convolution is spatially sampled rectilinearly with integer periods τ_x and τ_y. Depending on the desired signal properties of H, the total number of samples in H should be larger than h to ensure the reversibility of the STFT with compact windows. The main version of the STFT applied for this coding standard uses a rectangular window g, where the sample count remains constant. We then have

$$g[j,k] = \begin{cases} 1 \text{ if } 0 \le j < \tau_x \ \wedge \ 0 \le k < \tau_y \\ 0 \quad \text{otherwise} \end{cases}$$

which reduces to applying a block-based FFT to the input signal. These blocks correspond to the "window blocks" defined in section 3.1.5. For square blocks $\tau = \tau_x = \tau_y$, we get

$$H[x,y,u,v] = \sum_{j=0}^{\tau-1} \sum_{k=0}^{\tau-1} h[\tau x + j, \tau y + k] \cdot w^{ju+kv}$$

where $w = \exp\left(-\frac{2\pi i}{\tau}\right)$, equivalent to applying $\mathcal{F}$ (from annex D) on every $\tau \times \tau$ block.

The resulting 4D array H can then subsequently be subdivided into 4D code blocks.

G.2 Inverse Discrete Short-time Fourier transform

The general discrete STFT can be inverted provided it satisfies the "perfect reconstruction condition," depending on the properties of the window function g, and the sampling periods τ_x and τ_y. This can be efficiently computed using e.g., the "overlap-add method". When using a rectangular window g, this can be simplified to

$$h[x,y] = \sum_{u=0}^{\tau-1} \sum_{v=0}^{\tau-1} H\left[\left\lfloor \frac{x}{\tau} \right\rfloor, \left\lfloor \frac{y}{\tau} \right\rfloor, u, v\right] \cdot w^{-xu-yv}$$

equivalent to applying $\mathcal{F}^{-1}$ (from annex D) on every 2D section of H along dimensions (u,v), reorganizing them in $\tau \times \tau$ blocks along dimensions (x,y), and concatenating them in 2D preserving spatial ordering.

This can be written in pseudocode, as shown in Table G.1:

Table G.1 — Inverse short time Fourier transform.

Procedure ISTFT(H){	
`for (int x = 0; x < xdim; ++x) {`	Loop over the TB x-coordinate
`  for (int y = 0 ; y < ydim ; ++y) {`	Loop over the TB y-coordinate
`  B = idft2_slice(H, x, y);`	Compute the inverse 2D DFT of the slice (x,y) along the first two dimensions of H, return block in B
`  for (int u = 0; u < udim; ++u) {`	For every frequency (u-coordinate)
`  for (int v = 0; v < vdim; ++v) {`	For every frequency (v-coordinate)
`  h[x*udim + u, y*vdim + v] = B[u,v];`	
`  }`	
`  }`	
`  }`	
`}`	

Where `H` is the input 4D coefficient array of dimensions `(xdim, ydim, udim, vdim)`, `idft2_slice` is the inverse discrete 2D Fourier transform sliced along spatial dimensions `(x,y)`, and `h` is the output 2D hologram.

Annex H
(normative)

Quantization

H.1 Uniform scalar mid-rise quantizer

This Annex describes the quantization procedures to be followed for lossy non-binary hologram coding. As illustrated in Annex E.2.4, the transform coefficients belonging to the tile are organized into quantization blocks (QB) for quantization. Each QB contains the transform coefficients corresponding to a 4D hyperrectangle of dimensions $2^{\mathrm{XQBqcd}} \times 2^{\mathrm{YQBqcd}} \times 2^{\mathrm{UQBqcd}} \times 2^{\mathrm{VQBqcd}}$. A unique quantizer may be described for each QB, whose description is to be included as side information. Compression techniques are also applied to the quantizer description, wherein the description itself is to be quantized further and subject to entropy coding. In short, two types of quantization operations are performed

— Quantization of the QB STFT coefficients

— Meta quantization performed on the coefficient quantizer description

For both purposes, a uniform scalar mid-rise quantizer (MRQ) shall be used. For input x , the output of an MRQ centered at 0 with an integer bit depth $b>0$ and positive valued range X is

$$\mathcal{Q}(x,b,X)=\begin{cases} -2^{b-1}, & x\le -X \\ \left\lfloor \dfrac{2^{b-1}x}{X} \right\rfloor, & -X<x<X \\ 2^{b-1}-1, & x\ge X \end{cases}$$

The MRQ output is dequantized with an MRDQ as

$$\mathcal{Q}^{-1}(x,b,X)=\left(x+\frac{1}{2}\right)\frac{X}{2^{b-1}}$$

The transformation after a quantize-dequantize cycle is denoted as

$$\mathcal{Q}^{-1}\mathcal{Q}(x,b,X)=\mathcal{Q}^{-1}(\mathcal{Q}(x,b,X),b,X)$$

The MRQ/MRDQ can be uniquely defined by its quantization bit depth b and quantization range X . For typical coefficient distributions, the influence of the quantization bit depth b on the L2 distortion of the quantizer is convex (proportional to 2^{-2R} where R is the rate in bits). The reduction in global L2 distortion is most effective for the initial bit depths and exponentially decreases for each additional bit.

The influence of the quantization range X can be categorized into two for the value being quantized – either it is outside the quantization range and experiences clipping error, or it is inside the range and experiences a granular error. For the same bit depth, increasing the quantization range may decrease the clipping error but will increase the step size, which then potentially increases the granular error. At any bit depth, the optimal quantization range depends on the tradeoff between these two errors, and the distortion function tends to be unimodal in nature. Typically, as the MRQ bit depth increases, the influence of the granular error reduces, and as a result, the optimal quantization range increases as well.

The (e.g., real and imaginary) coefficients belonging to some QB can be quantized using an MRQ with a bespoke bit depth $b\in\{0,1,..,BDqcd\}$ and quantization range X . The motivation for permitting such adaptivity in quantization over relatively small 4D hypercuboids is due to the high degree of dynamic range

in the STFT domain for holographic content, where correlation in dynamic range decreases as QB size increases. In particular,

— Adaptive bit depth permits allocating more bits to regions with high energy and vice versa for the regions with low energy, thereby enabling an overall optimal allocation of bits to minimize the L2 distortion.

— Adaptive range permits tuning the MRQ to the coefficients residing in the QB and the bit depth its being quantized at, such that the L2 distortion can be minimized.

Since the STFT discussed in Annex G is orthonormal, the global L2 distortion as a result of the quantization is equal to the sum of the individual L2 distortions of the QB. This can be utilized by the encoder for optimizing the MRQ quantization parameters to be chosen for each QB. For example, it may utilize the unimodal dependence of the MRQ error on the quantization range to devise numerical optimization strategies to obtain the L2 distortion minimizing range for each QB at each bit depth. Afterward, it can utilize the Lagrangian multiplier method to determine the optimal bit depth allocation for each QB, either with or without convexity assumptions on quantizer rate-distortion behavior. As the optimization procedures are executed at a QB level, the search space is much more limited when compared to a global search. Thus, near-optimal parameters for the QB coefficient quantizer can be found for each of the QBs while still maintaining acceptable computational complexity.

H.2 Meta quantizer for quantization ranges

The drawback for permitting such adaptability is that the bit depths and quantization ranges used for the QBs with non-zero bit depths need to be transmitted as side information to the decoder. To mitigate this, the quantizer description is also subject to compression techniques. Since the chosen QB bit depth depends roughly on the energy content of the QB, and the energy content in neighbouring QBs are highly correlated, the QB bit depths are expected to be smoothly varying across phase-space and amenable to entropy coding. For efficiently representing the quantization range, it is quantized as well using a meta quantizer.

Here, the quantization ranges belonging to the QBs with the same QB bit depth are to be quantized with the same meta quantizer, as they are observed to be correlated with the QB bit depth. The meta quantizer is also an MRQ (not centered around zero as QB ranges are not zero-mean), and its parameters are obtained as a one-to-one mapping with respect to the QB bit depth. For each table entry index $b \in \{1,\ldots,BDqcd\}$, the following quantization parameters are to be obtained as a one-to-one mapping and shall be referred to as the Quantization Table. Each entry contains the following data fields:

— Offset $Qoff[b]$, which shall be subtracted from the QB range X before quantization and to be added back after dequantization to better address the dynamic range encountered by the meta quantizer.

— Bit depth $q[b] \in \{0,1,..,QBDqcdi\}$, used by the meta quantizer

— Range $Q[b]$, used by the meta quantizer

For each entry with an index b, three possible cases for the data layout are possible, elaborated on in Table H.1. Empty data fields (i.e. containing no data, without allocated bits) are denoted “NULL.”

Annex J describes a non-normative strategy not requiring convexity assumptions on quantizer rate-distortion behavior, for determining these quantization parameters.

Figure H.1 illustrates the quantization procedure at the encoder to obtain the payload corresponding to a QB, while Figure H.2 illustrates the dequantization procedure taking place at the decoder to obtain the decoded QB from the QB payload. Table H.1 lists the possible quantization entry types, and Tables H.2, H.3, and H.4, respectively, provide the pseudocode description of the mid-rise quantizer, the inverse mid-rise quantizer, and the quantization/inverse quantization process.

The QB bit depth b, the quantized QB range $Xint$, and $QBint$ quantized QB coefficients shown in Figure H.1 are stored at the same level and are subject to entropy coding as described in Annexes C.1.4 and C.1.5. The quantization table utilized by the meta quantizer can be signalled at both the tile level, as well as the code-block level. See Annexes B.5.6 and B.5.7. for the signalling syntax.

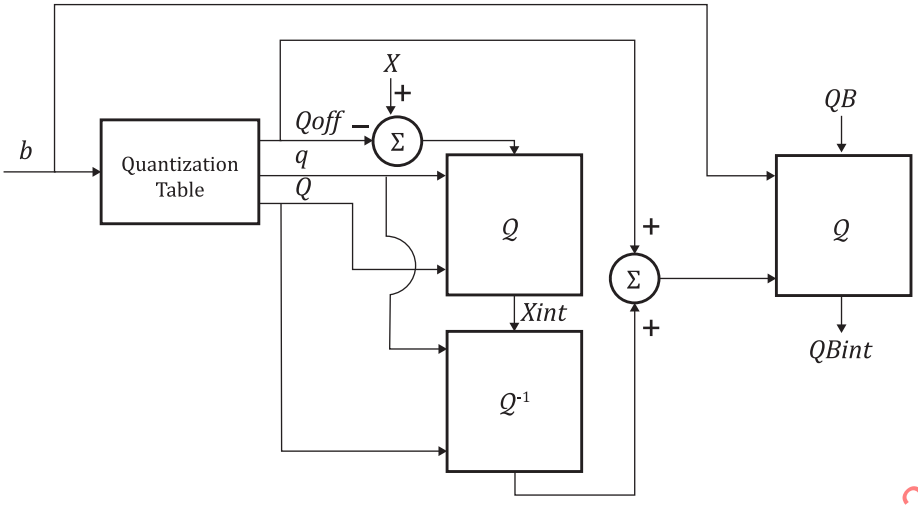


Figure H.1 — Application of the quantization procedure at the encoder on the QB to obtain the QB payload.

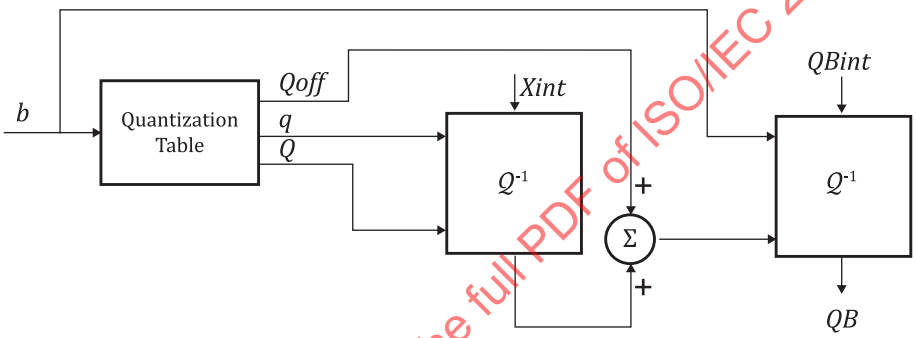


Figure H.2 — Application of the dequantization procedure at the decoder on the QB payload to obtain the decoded QB.

Table H.1 — Possible quantization table entries containing mapping between QB coefficient bit depth j and parameters used by the meta quantizer for quantizing QB range.

Case	Bit depth	Offset	Range	Comments
1	NULL	NULL	NULL	No coefficients are quantized with this bit depth and hence no parameters are signalled
2	$q[j]=0$	$Qoff[j]$	NULL	Coefficients corresponding to some QBs are quantized with this bit depth j. The quantization range is not quantized and stored for this bit depth and is assigned a constant value equal to the offset $Qoff[j]$.
3	$q[j]\neq 0$	$Qoff[j]$	$Q[j]$	Coefficients corresponding to some QBs are quantized with this bit depth j which has a non-zero mapping $q[j]$ for the meta quantizer. The other parameters (offset and range) also need to be specified.