INTERNATIONAL STANDARD

ISO/IEC 14495-2

Second edition
2003-04-01

Information technology — Lossless and near-lossless compression of continuous-tone still images: Extensions

Technologies de l'information — Compression sans perte et quasi sans perte d'images fixes à modelé continu: Extensions

Reference number
ISO/IEC 14495-2:2003(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

CONTENTS

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 14495-2 was prepared jointly by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*, in collaboration with ITU-T. The identical text is published as ITU-R Recommendation T.870.

This second edition cancels and replaces the first edition (ISO/IEC 14495-2:2002), which has been technically revised.

ISO/IEC 14495 consists of the following parts, under the general title *Information technology — Lossless and near-lossless compression of continuous-tone still images*:

— *Part 1: Baseline*

— *Part 2: Extensions*

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Lossless and near-lossless compression of continuous-tone still images: Extensions

1 Scope

This Recommendation | International Standard defines a set of lossless (bit-preserving) and nearly lossless (where the error for each reconstructed sample is bounded by a predefined value) compression methods for coding continuous-tone (including bi-level), gray-scale, or colour digital still images.

This Recommendation | International Standard:

- specifies extensions (including arithmetic coding, extension of near lossless coding, extension of prediction and extension of Golomb coding) to processes for converting source image data to compressed image data;
- specifies extensions to processes for converting compressed image data to reconstructed image data including an extension for sample tranformation for inverse colour transforms;
- specifies coded representations for compressed image data;
- provides guidance on how to implement these processes in practice.

2 Normative references

The following Recommendations and International Standards contain provisions which, through references in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- CCITT Recommendation T.81 (1992) | ISO/IEC 10918-1:1994, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines.*
- ITU-T Recommendation T.83 (1994) | ISO/IEC 10918-2:1995, *Information technology – Digital compression and coding of continuous-tone still images: Compliance testing.*
- ITU-T Recommendation T.84 (1996) | ISO/IEC 10918-3:1997, *Information technology – Digital compression and coding of continuous-tone still images: Extensions.*
- ITU-T Recommendation T.87 (1998) | ISO/IEC 14495-1:2000, *Information technology – Lossless and near-lossless compression of continuous-tone still images: Baseline.*

2.2 Additional references

- ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*
- ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.*
- ISO/IEC 9899:1999, *Programming languages – C.*

3 Definitions, abbreviations, symbols and conventions

3.1 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply in addition to the definitions used in ITU-T Rec. T.87 | ISO/IEC 14495-1.

3.1.1 **arithmetic encoder**: An embodiment of an arithmetic encoding procedure.

3.1.2 **arithmetic encoding**: A procedure which encodes a sample as a binary representation of the sequence of previously encoded samples by means of a recursive subdivision of a unit interval.

3.1.3 **arithmetic decoder**: An embodiment of an arithmetic decoding procedure.

3.1.4 **arithmetic decoding**: A procedure which recovers source data from an encoded bit stream produced by an arithmetic encoder.

3.1.5 **binary context**: Context used to determine the binary arithmetic coding of the present binary decision.

3.1.6 **binary decision**: Choice between two alternatives.

3.1.7 **colour transform**: A procedure for sample transformation for inverse colour transform.

3.1.8 **sign flipping**: The procedure which reverses the sign of a prediction error according to accumulated prediction errors.

3.1.9 **symbol packing**: A procedure which may be applied to source images in which sample values are sparsely distributed.

3.1.10 **visual quantization**: An extended function of near-lossless coding which enables to change the difference bound according to the context.

3.2 Abbreviations

In additions to the abbreviations used in ITU-T Rec. T.87 | ISO/IEC 14495-1, the abbreviations used in this Recommendation | International Standard are listed below.

FLC	Fixed length code
LPS	Less probable symbol
MPS	More probable symbol

3.3 Symbols

In addition to the symbols used in ITU-T Rec. T.87 | ISO/IEC 14495-1, the symbols used in this Recommendation | International Standard are listed below. A convention is used that parameters which are fixed in value during the encoding of a scan are indicated in **boldface** capital letters, and variables which change in value during the encoding of a scan are indicated in *italicised* letters.

Areg	current numerical-line interval being renormalized
ArithmeticEncode()	a function in the C programming language
Av[0..30]	31 constants corresponding to LPS probability estimate
Avd	auxiliary variable storing modified **Av**
BASIC_T1, BASIC_T2, BASIC_T3, BASIC_T4	basic default threshold values
Bin	binary decision
Buf[0..1]	bytes stored to avoid carry-over propagation to the encoded bit stream
Creg	value of code register storing the trailing bits of the encoded bit stream
ENT	indication of the coding process used for the scan
Flag[0..**MAXVAL**]	**MAXVAL**+1 flags which indicate if corresponding sample values already occurred

GetBinaryContext()	a function in the C programming language
GetByte()	a function in the C programming language
GetGolombk()	a function in the C programming language
Hd	auxiliary variable storing an integer value corresponding to a half of the full range but shifted according to the size of the current interval
LPScnt[0..**MAXS**]	accumulated occurrence count of the LPS (less probable symbol) at each binary context
MAXcnt	threshold value at which *MLcnt* and *LPScnt* are halved
MAXS	maximum index of binary contexts
MLcnt[0..**MAXS**]	accumulated occurrence count of each binary context
MPSvalue[0..**MAXS**]	sense of the MPS (more probable symbol) at each binary context
nearq	context-dependent difference bound for near-lossless coding using visual quantization
NEARRUN	difference bound for near-lossless coding in run mode
NMCU	number of MCUs
Prob	LPS probability estimated from *MLcnt* and *LPScnt*
Qx	the (quantized) value of a sample to be encoded with fixed length code
S	*index for binary contexts*
SOF$_{57}$	JPEG-LS frame marker for this extension
SPf[0..**RANGE**]	**RANGE**+1 flags indicating if corresponding mapped error values already occurred
SPm[0..**RANGE**]	mapping table of *MErrval* or *EMErrval* for symbol packing
SPt	the smallest positive integer greater than all mapped error values that occurred in the scan up to this point
SPx	number of the different mapped error values that already occurred
T1, T2, T3	thresholds for local gradients
T4	threshold for an additional local gradient
TEMErrval	auxiliary variable storing *EMErrval*
Th[0..29]	threshold to determine suitable value of **Av**
TMErrval	auxiliary variable storing *MErrval*
TQ	visual quantization threshold
wct	number of bits by which *Areg* is shifted
Zerograd	flag indicating local gradients are all zero

4 General

The purpose of this clause is to give an overview of this Recommendation | International Standard.

This Recommendation | International Standard defines extensions to the elements specified in ITU-T Rec. T.87 | ISO/IEC 14495-1. Extensions which pertain to encoding or decoding are defined as procedures which may be used in combination with the encoding and decoding processes of ITU-T Rec. T.87 | ISO/IEC 14495-1. This Recommendation | International Standard also defines extensions to the compressed data formats, i.e., interchange. Each encoding or decoding extension shall only be used in combination with particular coding processes and only in accordance with the requirements set forth herein. These extensions are backward compatible in the sense that decoders which implement these extensions will also support configuration subsets that are currently defined by ITU-T Rec. T.87 | ISO/IEC 14495-1.

4.1 Extensions specified by this Recommendation | International Standard

The following extensions are specified:

- An extension which provides for arithmetic coding. This extension will provide higher compression ratio, especially with high-skew images.
- An extension which provides for variable near-lossless coding. This extension will provide a wider variety of possible nearly lossless reconstructions of a source image than ITU-T Rec. T.87 | ISO/IEC 14495-1. There are two types of variable near-lossless coding, depending on whether the difference bound is changed:
  a) according to its context; or
  b) in vertical direction.
- An extension which provides for a modified prediction procedure in source images in which sample values are sparsely distributed.
- An extension which provides for a modified Golomb coding procedure. This modification avoids possible expansion of compressed image data, and improves coding efficiency by eliminating code words that are not used.
- An extension which provides for fixed length coding.
- An extension which provides for a modified sample transformation process. This extension can be used to define inverse colour transforms in order to achieve greater efficiency by compressing a source image in a different colour representation.

The following subclauses describe these extensions in greater detail.

4.1.1 Encoding with arithmetic coding

In JPEG-LS baseline, specified in ITU-T Rec. T.87 | ISO/IEC 14495-1, simple but efficient coding is achieved by the combination of Golomb coding (regular mode) and the run mode. However, for some very high-skewed image data such as computer generated images, the compression efficiency is affected by the use of symbol-by-symbol coding in contexts that present highly skewed distributions. Therefore, coding procedures based on arithmetic coding are specified in this Recommendation | International Standard as an extended function, which enables alphabet extension for every context (rather than only in run mode) and provides higher compression performance with a moderate increase of the coder complexity.

The arithmetic coder adopted in this Recommendation | International Standard is characterized by its fast multiplication-free arithmetic operation and radix-255 representation. The details are described in Annex A and B.

4.1.2 Extension of near-lossless coding

The extension of the near-lossless coding capabilities of ITU-T Rec. T.87 | ISO/IEC 14495-1 is to allow the **NEAR** parameter to vary during the process of encoding a source image. There are two types of variable near-lossless coding, serving two different purposes.

4.1.2.1 Visual quantization

Visual quantization takes into account the human visual system by primarily performing the quantization in high activity regions of the image where the activity may mask for the quantization noise. Therefore, by extending the near-lossless coding capabilities of ITU-T Rec. T.87 | ISO/IEC 14495-1 so that the **NEAR** parameter can change according to its context, it becomes possible to provide reconstructed images whose distance from the source image is between those obtained with compression schemes using **NEAR** = n and **NEAR** = $n + 1$, where n denotes a non-negative integer.

4.1.2.2 Re-specification of NEAR value

This type of variable near-lossless coding can vary the **NEAR** parameter according to the vertical direction. The main purpose of this extension of the near-lossless coding capabilities of ITU-T Rec. T.87 | ISO/IEC 14495-1 is to provide a mechanism by which an encoder can change the value of **NEAR** according to the observed compressibility of the source image, which is useful to control the total length of compressed image data within some specified amount. By this extension, an encoder can compress a source image to less than a pre-specified size with a single sequential pass over the image. The capability is valuable to applications which utilize a fixed-size compressed image memory.

4.1.3 Extension of prediction

The prediction and error coding procedure specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 is not suitable for images with sparse histograms, such as limited colour images or fewer-bit images expressed by byte form. These images contain only a subset of the possible sample values in each component, and the edge-detecting predictor specified in code segment A.5 of ITU-T Rec. T.87 | ISO/IEC 14495-1 would tend to concentrate the value of the prediction errors into a reduced set. However, the prediction correction procedure specified in code segment A.6 of ITU-T Rec. T.87 | ISO/IEC 14495-1 tends to spread these values over the entire range. In addition, the Golomb coding procedure would assign short code words to small prediction errors, even those that do not occur in the image component.

The goal of this extension is to modify the prediction procedure in order to alleviate this unwanted effect by checking the occurrence of the corrected predicted value *Px* in the past. This extension also provides a modified coding procedure to improve the coding efficiency for these images.

4.1.4 Extension of Golomb coding

In addition to the specification of Golomb coding defined in ITU-T Rec. T.87 | ISO/IEC 14495-1, two modifications are incorporated in this Recommendation | International Standard as follows:

4.1.4.1 Golomb code completion

More effective usage of Golomb code words is specified in this Recommendation | International Standard, in which the final bit "1" in the longest possible unary representation used in a Golomb code, which is redundant, shall be omitted. This procedure improves the coding efficiency especially in cases in which this Recommendation | International Standard is also applied to bi-level images.

4.1.4.2 Omission of run interruption sample coding

In cases in which this Recommendation | International Standard is also applied to bi-level images and the mode is not sample interleaved, the encoding of the run interruption sample is superfluous and shall be omitted.

4.1.5 Fixed length coding

In this Recommendation | International Standard, a procedure to avoid situations in which the compressed image data is larger than the source image data is incorporated. The values of image samples, or the quantized values in near-lossless coding, are encoded with a fixed length code. The region in a scan to be encoded with a fixed length code is specified by appending a marker indicating the beginning of the fixed length coding and the end of the fixed length coding after an appropriate MCU.

4.1.6 Sample transformation for inverse colour transforms

In this Recommendation | International Standard, a procedure for sample transformation is provided, in addition to the ones defined in ITU-T Rec. T.87 | ISO/IEC 14495-1. This procedure uses corresponding values of decoded samples in the various components, to reconstruct the source image data, which is of the same precision as the encoded data. The goal of this extension is to facilitate the use of this Recommendation | International Standard in conjunction with colour transforms to improve coding efficiency.

4.2 Descriptions of extended functions

The coding procedure specified in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1 is referred to as baseline coding process. The newly introduced coding procedure, modified from the baseline coding process, is referred to as arithmetic-based process. The context modelling for the arithmetic-based coding process is described in Annex A, and the arithmetic coding procedure of a binary symbol for the given context is described in Annex B. The functions outlined in 4.1.2, 4.1.3 and 4.1.6 can be used on either arithmetic-based coding process or baseline coding process. The functions outlined in 4.1.4 and 4.1.5 can be used only in baseline coding process. All the extended functions are optional and the combinations of the extended functions are arbitrary under this general rule.

The use of the extended functions outlined in 4.1.2 and 4.1.3 in conjunction with the arithmetic-based coding process is also described in Annex A. The use of the extended functions outlined in 4.1.2 and 4.1.3 in conjunction with the baseline coding process is described in Annex D by referring to the differences with respect to the coding process of the non-extended functions described in ITU-T Rec. T.87 | ISO/IEC 14495-1. The functions outlined in 4.1.4 are also described in Annex D.

The extended functions outlined in 4.1.5 are described in Annex E. The extended functions outlined in 4.1.6 on both arithmetic-based and baseline coding are described in Annex F.

The contents of the annexes for extended functions are summarized in Table 1.

Table 1 – Combination of extended functions and corresponding annex

Extended functions / Coding process	Extension of near-lossless coding	Extension of prediction	Arithmetic coding procedure	Extension of Golomb coding	Fixed length coding	Colour transform
Baseline coding process	Annex D	Annex D		Annex D	Annex E	Annex F
Arithmetic-based coding process	Annex A	Annex A	Annex B			

5 Interchange format requirements

The interchange format is the coded representation of compressed image data for exchange between application environments.

The interchange format requirements are that any compressed image data represented in interchange format shall comply with the syntax and code assignments appropriate for the coding process and extensions selected, as defined in Annex C of ITU-T Rec. 87 | ISO/IEC 14495-1 and Annex G.

6 Encoder requirements

An encoding process converts source image data to compressed image data. ITU-T Rec. T.87 | ISO/IEC 14495-1 specifies baseline coding processes. This Recommendation | International Standard defines arithmetic-based coding process and encoding extensions which may be used in combination with baseline coding process or arithmetic-based coding process.

An extended encoder is an embodiment of one (or more) of the encoding processes specified in this Recommendation | International Standard or ITU-T Rec. T.87 | ISO/IEC 14495-1 used in combination with one or more of the encoding extensions specified herein. In order to comply with this Recommendation | International Standard, an extended encoder shall satisfy at least one of the following two requirements.

An extended encoder shall:

a) convert source image data to compressed image data which conform to the interchange format syntax specified in Annex G;

b) convert source image data to compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex G.

Conformance tests for the above requirements are specified in clause 8.

NOTE – There is no requirement in this Recommendation | International Standard that any encoder which embodies one of the encoding processes and extensions shall be able to operate for all ranges of the parameters which are allowed. An encoder is only required to meet the applicable conformance tests, and to generate the compressed data format according to Annex G for those parameter values which it does use.

7 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Since the decoding process is uniquely defined by the encoding process, there is no separate normative definition of the decoding processes. The values of samples output by the decoding process are used as vector components in an inverse colour transform. The inverse colour transform is specified in Annex F. If no transform is specified for a sample component, then the colour-transformed sample value is identical to the sample value output by the decoding process. In this case, an inverse point transform may also be applied (see 4.3.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1). A subsequent sample mapping procedure uses the value of each sample output by the inverse colour transform procedure to map each sample value to sample-mapped value using the mapping tables specified for that sample component in Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1. Again, if no table is specified for that sample component, then the sample-mapped value is identical to the colour-transformed sample value (after possible inverse point transform).

A decoder is an embodiment of the decoding process implicitly specified by the encoding process as specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 and this Recommendation | International Standard, followed by the embodiment of the sample transformation process defined above. In order to comply with this Recommendation | International Standard, an extended decoder shall satisfy all three of the following requirements.

An extended decoder shall:

a) convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which comply with the interchange format syntax specified in Annex G. In the reconstructed image data output by the embodiment of the decoding process (before sample transform), the values of each sample shall be identical to the reconstructed values defined in the encoding process;

b) accept and properly store any table-specification data which conform to the abbreviation format for table-specification data syntax specified in Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1;

c) convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which conforms to the abbreviated format for compressed image data syntax specified in Annex G, provided that the table specification data required for sample mapping has previously been installed in the decoder.

NOTE – There is no requirement in this Recommendation | International Standard that any decoder which embodies one of the decoding processes and extensions shall be able to operate for all ranges of the parameters which are allowed. A decoder is only required to meet the applicable conformance tests, and to decode the compressed image data format specified in Annex G for those parameter values which it does use.

8 Conformance testing for extensions

8.1 Purpose

The conformance tests specified in this Recommendation | International Standard are intended to increase the likelihood of compressed image data interchange by specifying a range of tests for both encoders and decoders. The tests are not exhaustive tests of the respective functionality, and hence do not guarantee complete interoperability between independently implemented encoders and decoders. The main purpose of these conformance tests is to verify the validity of encoding and decoding process implementations, and the corresponding compressed image data. It is not an objective of these tests to carry out extensive verification of the interchange format or marker segment syntax. The marker segment syntax follows closely the interchange formats specified in Annex B of ITU-T Rec. T.81 | ISO/IEC 10918-1, and testing procedures similar to those specified in ITU-T Rec. T.83 | ISO/IEC 10918-2 can be used for the purpose of verifying interchange format and marker segment syntax.

The tests are based on a set of test images which are incorporated into this specification in digital form.

8.2 Encoder conformance tests

Encoders are tested by encoding a source test image (see Annex I) using the encoder under test, and then comparing the compressed image data produced by the encoder to the compressed image data listed in Table I.2. The coded data segments of the compressed image data shall match those of the compressed image data in Table I.2.

The encoding shall be carried out for each of the tests listed in Table I.2 using the test images listed in the "Source Image" column, and using the parameters specified in the table. Restart markers shall not be inserted. The encoder testing procedure is illustrated in Figure 1.

NOTE – This testing is restricted to conformance of the coded data segments only, excluding marker segments (as different marker segments may represent the same coding parameters).

The above conformance tests shall be performed without sample mapping and with **Pt** = 0.

8.3 Decoder conformance tests

Decoders are tested by decoding compressed test image data (see Annex I) using the decoder under test and comparing the reconstructed image to the corresponding source test image. The image reconstructed by the decoder under test shall exactly match the source test image in the case of lossless coding (**NEAR** = 0). In the case of near-lossless coding (**NEAR** > 0), the image reconstructed by the decoder under test shall be the source image data with **NEAR** for all samples.

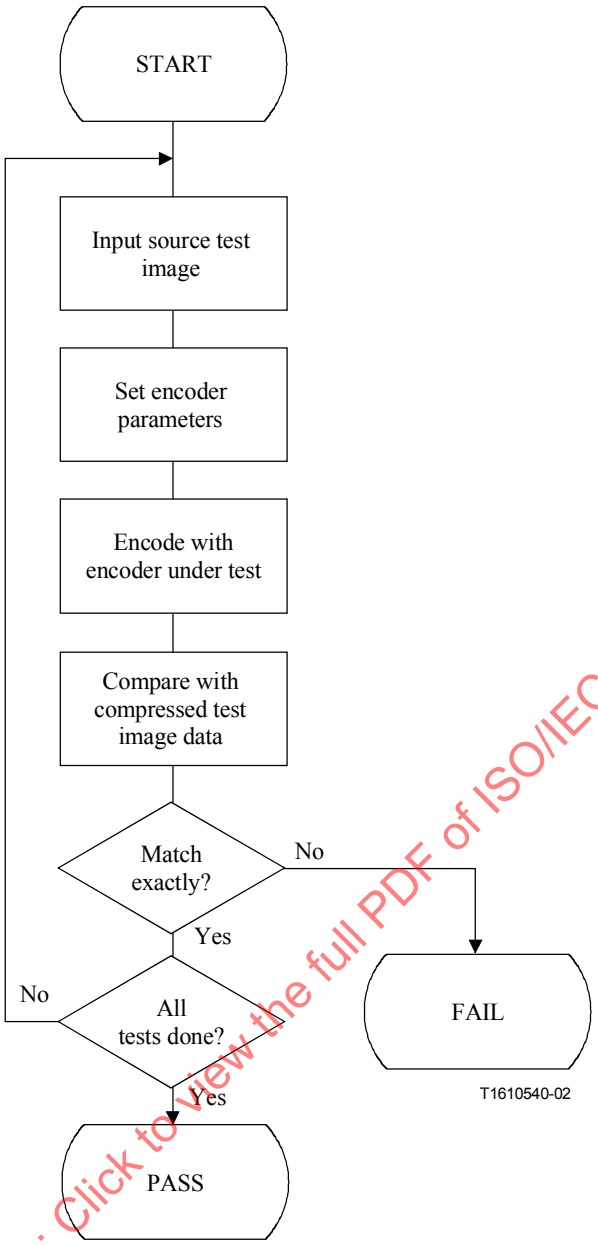

Figure 1 – Encoder testing procedure

The decoding conformance tests shall be carried out for each of the tests listed in Table I.2, using as an input the compressed test image data listed in the "Compressed file name" column, with the parameters specified in the table. The source test images used for the comparison are listed in the "Source image" column of Table I.2. The decoder testing procedure is illustrated in Figure 2.

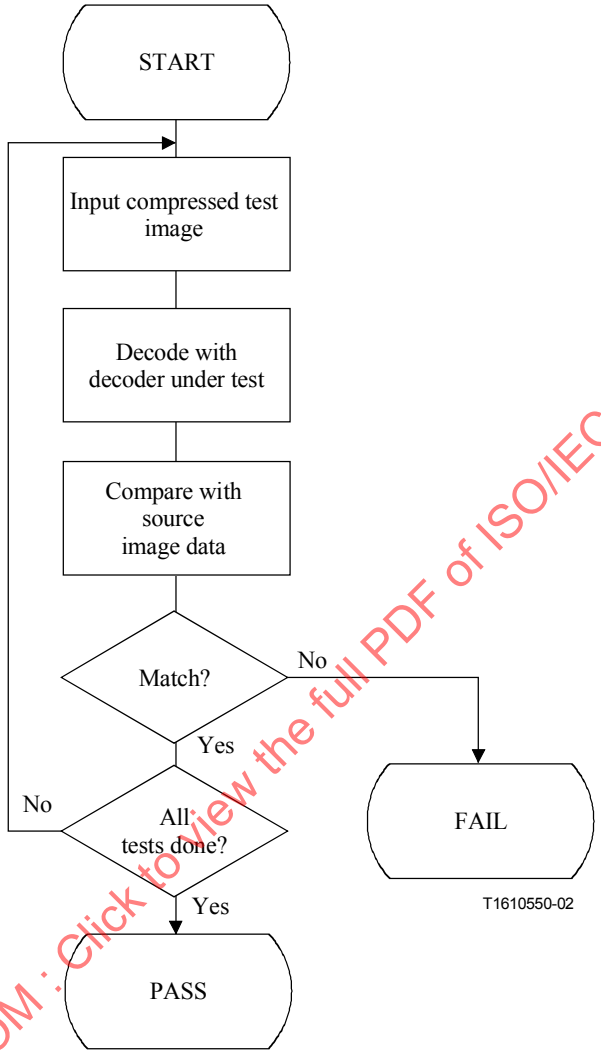

Figure 2 – Decoder testing procedure

Annex A

Encoding procedures with arithmetic coding for a single component

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies the encoding procedures using arithmetic coding. The encoding procedures using arithmetic coding (arithmetic-based coding process) is similar to those using Golomb coding (baseline coding process), which is specified in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1. However, since there are many differences in detail, this independent annex is provided for its description. The main differences are as follows:

- Binary arithmetic coding is used in stead of Golomb coding.
- No classification for regular mode and run mode.
- Binarization of mapped-error and context modelling for binary arithmetic coding are performed.
- The number of samples used in context modelling is increased from four to five.

This annex presumes a single component. The necessary modifications for dealing with multiple-component scans are specified in Annex C.

NOTE – There is **no requirement** in this Recommendation | International Standard that any encoder or decoder shall implement the procedures in precisely the manner specified in this annex. It is necessary only that an encoder or decoder implement the function specified in this annex. The sole criterion for an encoder or a decoder to be considered in conformance with this Recommendation | International Standard is that it satisfy the requirements determined by the conformance tests given in clause 8.

A.1 Coding parameters and compressed image data

A number of parameters are necessary to specify the arithmetic-based coding process in this Recommendation | International Standard. The coding of these parameters in the compressed image data, and a normative set of default values for some of these parameters are specified in Annex G. This Recommendation | International Standard does not specify how these parameters are set in the encoding process by any application using it, if non-default values are used.

The bits generated by the encoding process forming the compressed image data shall be packed into 8-bit bytes. These bits shall fill bytes in decreasing order of significance. As an example, when outputting a binary code a_n, a_{n-1}, a_{n-2},....a_0, where a_n is the first output bit, and a_0 is the last output bit, a_n will fill the most significant available bit position in the currently incomplete output byte, followed by a_{n-1}, a_{n-2}, and so on. When an output byte is completed, it is placed as the next byte of the encoded bit stream, and a new byte is started.

Marker segments are inserted in the data stream as specified in Annex G. In the coded data segment, a bit '0' is inserted after the X'FF' byte like the procedures specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 in the baseline coding process, but in the arithmetic-based coding process, which is specified in Annex B, no such treatment is necessary because every X'FF' byte, if happened, shall be always followed by X'00' byte in the data stream and will not be mistaken as marker code.

A.2 Initializations and conventions

A.2.1 Initializations

The differences from the initializations specified in A.2.1 of ITU-T Rec. T.87 | ISO/IEC 14495-1 are as follows :

- initialization of reconstruction values outside the border of an image;
- initialization of additional counters of A, B, C and N, which are caused by the increase of samples used for context modelling;
- initialization of variables for arithmetic coding;
- elimination of initialization of variables for the run mode.

The context modeling procedure specified in this annex uses the causal template *a*, *b*, *c*, *d* and *e* depicted in Figure A.1. When encoding the first line of a source image component, the samples at positions *b*, *c*, and *d* are not present, and their reconstructed values are defined to be zero. If the sample at position *x* is at the start or end of a line so that either *a*, *c* and *e*, or *d* is not present, the reconstructed value for a sample (samples) in position *a* and *e*, or *d* is defined to be equal to *Rb*, the reconstructed value of the sample at position *b*, or zero for the first line in the component. The reconstructed value at a sample in position *c*, in turn, is copied (for lines other than the first line) from the value that was assigned to *Ra* when encoding the first sample in the previous line. If the sample at position *x* is at the second column of a line so that *e* is not present, the reconstructed value for a sample in position *e* is copied from the value that was assigned to *Ra* when encoding the previous sample.

The following initializations shall be performed at the start of the encoding process of a scan, as well as in other situations specified in Annex H. All variables are defined to be integers with sufficient precision to allow the execution of the required arithmetic operations without overflow or underflow, given the bounds on the parameters indicated in Annex G:

1) Compute the parameter **RANGE**: For lossless coding (**NEAR** = 0), **RANGE** = **MAXVAL** +1. For near-lossless coding (**NEAR** > 0):

$$\mathbf{RANGE} = \left\lfloor \frac{\mathbf{MAXVAL} + 2 * \mathbf{NEAR}}{2 * \mathbf{NEAR} + 1} \right\rfloor + 1$$

NOTE 1 – **MAXVAL** and **NEAR** are coding parameters whose values are either default or set by the application (see Annex C of of ITU-T Rec. T.87 | ISO/IEC 14495-1).

Compute the parameters **qbpp** = $\lceil \log \mathbf{RANGE} \rceil$, **bpp** = max(2, $\lceil \log(\mathbf{MAXVAL}+1) \rceil$).

2) Initialize the variables *N*[0..1091], *A*[0..1091], *B*[0..1091] and *C*[0..1091], where the nomenclature [0..i] indicates that there are i + 1 instances of the variable. The instances are indexed by [*Q*], where *Q* is an integer between 0 and i. For example, *C*[5] corresponds to the variable C indexed by 5. Each one of the entries of *A* is initialized with the value:

$$\max\left(2, \left\lfloor \frac{\mathbf{RANGE} + 2^5}{2^6} \right\rfloor\right)$$

those of *N* are initialized with the value 1, and those of *B* and *C* with the value 0.

3) Initialize the variables for the arithmetic coder, *LPScnt*[0..**MAXS**], *MLcnt*[0..**MAXS**] and *MPSvalue*[0..**MAXS**], where **MAXS** is the maximum index of binary contexts. At the nodes for the unary representation of the Golomb code tree described in A.5.2, counters are initialized as *LPScnt*[*S*]=2 and *MLcnt*[*S*]=4, where *S* is an index of the binary context. At the nodes in sub-trees, counters are initialized as *LPScnt*[*S*]=4 and *MLcnt*[*S*]=8. The sense of MPS *MPSvalue*[*S*] is initialized as *MPSvalue*[*S*]=0 for all the binary contexts.

4) Initialize the error tolerance for near-lossless coding as *nearq* = **NEAR** (in lossless, *nearq* = 0). If an extended function of near-lossless coding is indicated, initialize the visual quantization threshold **TQ** as is specified in the LSE marker segment..

NOTE 2 – In the extended near-lossless coding specified in this Recommendation | International Standard, the error tolerance *nearq* is variable, although in coding and near-lossless coding without the extension and lossless coding, it is fixed to **NEAR** during the encoding of a scan.

5) If the extension of prediction is indicated, initialize the variable *Flag*[1..**MAXVAL**]. *Flag*[0] is initialized with the value 1, and *Flag*[1..**MAXVAL**] with the value 0.

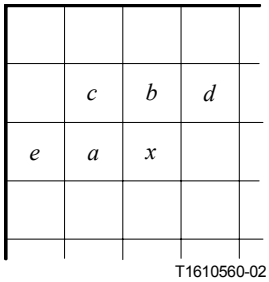

Figure A.1 – Causal template used for encoding with arithmetic coding

A.2.2 Conventions for figures

In the remaining clauses of this annex, various procedures of the encoding process are specified in software code segments, written in the C programming language, as specified in ISO/IEC 9899:1990. The syntax and semantics of C shall be assumed in all code segments contained in this annex.

All variables used in the code segments are assumed to be integer, and to have sufficient precision to allow the execution of the required arithmetic operations without overflow or underflow, given the bounds on the parameters indicated in Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1 and Annex G. When division and right shift operations are indicated, all variables used are non-negative integers so that the exact computation of quotients, remainders and shifted quantities is unambiguously specified. The figures are used to specify parts of the encoding process, and do not constitute, by themselves or in any aggregation, a full implementation of the process.

In addition to the variables and parameters specified in 3.1 for the encoding and decoding processes, the following auxiliary labels, global variables, and functions are used in the software code segments:

abs(*i*)	Function: returns the absolute value of *i* in accordance with the definition in 3.1.4 of ITU-T Rec. T.87 | ISO/IEC 14495-1.
max(*i*, *j*)	Function: returns the maximum of *i* and *j* in accordance with the definition in 3.1.47 of ITU-T Rec. T.87 | ISO/IEC 14495-1.
min(*i*, *j*)	Function: returns the minimum of *i* and *j* in accordance with the definition in 3.1.48 of ITU-T Rec. T.87 | ISO/IEC 14495-1.
AppendToBitStream(*i*, *j*)	Function: appends the non-negative number *i* in binary form to the encoded bit stream, using *j* bits. Most significant bits are appended first. The process guarantees that *j* bits are sufficient to represent *i* exactly.
ModRange(*i*, RANGE)	Function: returns the value of *i* modulo **RANGE** as described in A.4.5.
ArithmeticEncode(*Bin, S*)	Function: encodes a binary decision *Bin* conditioned with the binary context *S* by the arithmetic coding specified in Annex B.
GetBinaryContext()	Function: returns the binary context *S* for a binary decision to be encoded according to the binarization specified in A.5.2.
GetGolombk(*S*)	Function: returns the Golomb parameter k used to binarize a mapped error.
GetByte()	Function: reads the next byte from the coded image data segment and returns the byte.

A.3 Context determination

In the arithmetic-based coding process, five samples are used for context modelling instead of four samples used for the baseline coding process. In this case, an explicit run mode does not exist; however, contexts with small local gradients are recognized by a flag.

After a number of samples have been coded scanning from left to right and from top to bottom, the sample *x* positioned as in Figure A.1 shall be encoded. The context at this sample shall be determined by the previously reconstructed values *Ra, Rb, Rc, Rd,* and *Re* corresponding to the samples *a*, *b*, *c*, *d*, and *e* as shown in Figure A.1, respectively. In lossless coding, the reconstructed values are identical to those of the source image data. The steps in context determination, to be performed in the presented order, are the following:

A.3.1 Local gradient computation

The first step in the context determination procedure shall be to compute the local gradient values, *D*1, *D*2, *D*3 and *D*4 of the neighbourhood samples, as indicated in Figure A.2.

$$D1 = Rd - Rb;$$

$$D2 = Rb - Rc;$$

$$D3 = Rc - Ra;$$

$$D4 = Ra - Re;$$

Figure A.2 – Local gradient computation for context determination

A.3.2 Flat region detection

In the arithmetic-based coding process, an explicit run mode does not exist; however, when the local gradients are all zero (for lossless), or their absolute values are less than or equal to **NEAR**, the context that meets the above condition is recognized by a flag.

```
if ((abs(D1) <= NEAR) && (abs(D2) <= NEAR) && (abs(D3) <= NEAR) )
    Zerograd = 1;
else
    Zerograd = 0;
```

Figure A.3 – Flat region detection procedure

```
if ( (D1 == 0) && (D2 == 0) && (D3 == 0) )
    Zerograd = 1;
else
    Zerograd = 0;
```

Figure A.4 – Flat region detection procedure for lossless coding

A.3.3 Local gradient quantization

The context determination procedure shall continue by quantizing $D1$, $D2$, $D3$, and $D4$ according to the procedure specified in Figures A.5 and A.6. For this purpose, non-negative thresholds, **T1**, **T2**, **T3** and **T4** are used. The default values of these thresholds, and ways to explicitly override these defaults are specified in Annex G. In Figure A.5, the entry Di to the procedure is one of the values $D1$, $D2$, or $D3$ from the local gradient computation step. According to their relation with the thresholds, a region number Qi is obtained ($Q1$, $Q2$, $Q3$ and $Q4$ respectively). The procedures in Figures A.5 and A.6 form a vector ($Q1$, $Q2$, $Q3$, $Q4$) representing the context for the sample x.

```
if (Di <= –T3)  Qi = –4;
else if (Di <= –T2)  Qi = –3;
else if (Di <= –T1)  Qi = –2;
else if (Di < –NEAR)  Qi = –1;
else if (Di <= NEAR)  Qi = 0;
else if (Di < T1)  Qi = 1;
else if (Di < T2)  Qi = 2;
else if (Di < T3)  Qi = 3;
else  Qi = 4;
```

Figure A.5 – Quantization of the most significant gradients

```
if (D4 <= –T4)  Q4 = –1;
else if (D4 < T4)  Q4 = 0;
else  Q4 = 1;
```

Figure A.6 – Quantization of the least significant gradients

A.3.4 Quantized gradient merging

If the first non-zero element of the vector ($Q1$, $Q2$, $Q3$) is negative, then all the signs of the vector ($Q1$, $Q2$, $Q3$, $Q4$) shall be reversed to obtain ($-Q1$, $-Q2$, $-Q3$, $-Q4$), and the variable *SIGN* shall be set to –1, otherwise it shall be set to +1. By this possible "merging", the total of $9 \times 9 \times 9 \times 3 = 2187$ possible vectors, defined by the procedure in Figures A.5 and A.6, is merged into $\{(9 \times 9 \times 9+1)/2\} \times 3 = 1095$. The vectors (0, 0, 0, –1), (0, 0, 0, 0) and (0, 0, 0, 1) are also merged into one context at the same time. By these possible merging, the possible number of single component contexts will be 1093.

The function to map the vector ($Q1$, $Q2$, $Q3$, $Q4$) except (0, 0, 0, –1), (0, 0, 0, 0) and (0, 0, 0, 1) to the integer Q representing the context for the sample x on a one-to-one basis is not specified in this Recommendation | International Standard. This Recommendation | International Standard only requires that the mapping shall be one-to-one to produce an integer in the range of [0..1091].

A.3.5 Adjustment of error tolerance for near-lossless coding with visual quantization

If near-lossless coding with visual quantization is indicated as an extended function, adjust the error tolerance *nearq* as in Figure A.7.

```
if(| Q1 | + | Q2 | + | Q3 |>= TQ)
    nearq = NEAR + 1;
else
    nearq = NEAR;
```

Figure A.7 – Adjustment of NEAR value

A.4 Prediction

The procedures of prediction is the same as in baseline coding process except that sign flipping specified in Figure A.12 is performed. The following steps shall be performed in the order specified.

A.4.1 Edge-detecting predictor

An estimate *Px* of the value at the sample at x to be encoded shall be determined from the values *Ra, Rb,* and *Rc* at the positions *a, b,* and *c* specified in Figure A.1, as indicated in Figure A.8.

```
if (Rc >= max(Ra, Rb))
    Px = min(Ra, Rb);
else {
    if (Rc <= min(Ra, Rb))
        Px = max(Ra, Rb);
    else
        Px = Ra + Rb – Rc;
}
```

Figure A.8 – Edge-detecting predictor

A.4.2 Prediction correction

After *Px* is computed, if *Zerograd* is zero, the prediction shall be corrected according to the procedure depicted in Figure A.9 or Figure A.10, which depends on *SIGN,* the sign detected in the context determination procedure. The new value of *Px* shall be clamped to the range [0..**MAXVAL**]. The prediction correction value $C[Q]$ is derived from the bias as specified in A.6.2.

```
if (Zerograd == 0) {
    if (SIGN == +1)
        Px = Px + C[Q];
    else
        Px = Px – C[Q];

    if (Px > MAXVAL)
        Px = MAXVAL;
    else if(Px < 0)
        Px = 0;
}
```

Figure A.9 – Prediction correction from the bias

If the extension of prediction is indicated, the procedure in Figure A.10 is performed instead of Figure A.9.

```
if (Zerograd == 0) {
    Pmed = Px;
    if (SIGN == +1)
        Px = Pmed + C[Q];
    else
        Px = Pmed - C[Q];

    if (Px > MAXVAL)
        Px = MAXVAL;
    else if(Px < 0)
        Px = 0;

    if(Flag[Px] == 0) {
        if (Px < Pmed)
            for (Px++; Flag[Px] == 0; Px++);
        else
            for (Px--; Flag[Px] == 0; Px--);
    }
}
```

Figure A.10 – Prediction correction for the extension of prediction

A.4.3 Computation of prediction error

Using the value of *Px*, corrected by the above procedure, the prediction error, *Errval*, shall be computed. If the sign of the context, given by *SIGN*, is negative, the sign of the error shall be reversed. This is shown in Figure A.11 for the sample at position *x*, with value *Ix*.

```
Errval = Ix – Px;
if (SIGN == –1)
    Errval = –Errval;
```

Figure A.11 – Computation of prediction error

Furthermore, in the arithmetic-based coding process, if *Zerograd* is zero, the sign of *Errval* shall be flipped according to the sign of $B[Q]$ as specified in Figure A.12. This process has good effect since the shapes of the distribution of the value of *Errval* for positive $B[Q]$ and negative $B[Q]$ is quite mirror like.

```
if( (Zerograd == 0) && (B[Q] > 0) )
    Errval = - Errval;
```

Figure A.12 – Sign flipping

A.4.4 Error quantization for near-lossless coding, and reconstructed value computation

In lossless coding (*nearq* = 0), the reconstructed value *Rx* shall be set to *Ix*. In near-lossless coding (*nearq* > 0), the error shall be quantized. After quantization, the reconstructed value *Rx* of the sample *x*, which is used to encode further samples, shall be computed in the same manner as the decoder computes it. These operations are shown in Figure A.13.

```
if (Errval > 0)
    Errval =  (Errval + nearq) /(2 * nearq + 1);
 else
    Errval  = – (nearq – Errval) /(2 * nearq + 1);

if( (Zerograd == 0) && (B[Q] > 0))
    Rx = Px + SIGN * Errval * (2 * nearq + 1);
else
    Rx = Px – SIGN * Errval * (2 * nearq + 1);

if (Rx < 0)
    Rx = 0;
else if (Rx > MAXVAL)
    Rx = MAXVAL;
```

Figure A.13 – Error quantization and computation of the reconstructed value in near-lossless coding

NOTE – In the extended near-lossless coding specified in this Recommendation | International Standard, the error tolerance *nearq* is variable, although in near-lossless coding without the extension, it is fixed to **NEAR** during the encoding of a scan.

A.4.5 Modulo reduction of the prediction error

The error shall be reduced to the range relevant for coding, ($\lfloor$–**RANGE**/2$\rfloor$.. $\lfloor$**RANGE**/2$\rfloor$–1). This is achieved with the steps detailed in Figure A.14 (function **ModRange**()).

```
if(Errval < 0)
        Errval = Errval + RANGE;
if(Errval >= ((RANGE + 1) / 2))
        Errval = Errval – RANGE;
```

Figure A.14 – Modulo reduction of the error

A.5 Prediction error encoding

The procedures to encode the prediction error are considerably different from those of the baseline coding process. A non-negative integer *MErrval* mapped from the variable *Errval* shall be encoded with the binary arithmetic coder. The mapped error value, *MErrval,* is binarized by Golomb code. The Golomb parameter *k* shall be decided from the activity class *Act*, which is derived from accumulated prediction error magnitudes *A*.

A.5.1 Error mapping

The prediction error, *Errval* shall be mapped to a non-negative value, *MErrval* as specified in Figure A.15.

```
if(Errval >= 0)
    MErrval = 2 * Errval;
else
    MErrval = - 2 * Errval - 1;
```

Figure A.15 – Error mapping to non-negative values

A.5.2 Binarization of MErrval with the Golomb code tree

The mapped-error shall be decomposed into the sequence of binary decisions so as to be encoded with the binary arithmetic coder specified in Annex B. The binarization is done according to a code tree, which is specified according to the activity class *Act* associated with the Golomb parameter *k* as in Figure A.16.

```
if( Zerograd == 0) {
    for(k = 0; (N[Q] << k) < A[Q] + N[Q] / 2; k++);
    if( qbpp < 10) {
        if(k > 0 )
            if( 5 * (N[Q] << k) > 7 * (A[Q] + N[Q] / 2) )
                Act = 2 * k;
            else
                Act = 2 * k + 1;
        else
            Act = 1;
    } else
        Act = k + 1;
    if( Act > 11) Act = 11;
} else
    Act = 0;
```

Figure A.16 – Computation of the activity class *Act*

The code tree is composed of a number of sub-trees, each of which represents the leaves of 2^k and the value of *k* may differ according to the activity class and also the order of the sub-trees in the code tree. If the leaf to be coded is in the sub-tree now being separated, symbol 0 is given, and otherwise symbol 1 is given. In the sub-tree, 2^k leaves are binary expressed with fixed length code. The total binarization is done combining such sub-tree separation and in-sub-tree expression. In the code tree specified here, the parameter *k* of first some sub-trees are given as $k=\lfloor Act/2 \rfloor$ if **qbpp** < 10, otherwise $k=\mathbf{max}(Act-1, 0)$. After the first some sub-trees, the *k* parameter of sub-trees may take larger value, and sub-trees may be merged into the sub-trees of the code trees of higher activity classes. Each node of the whole code trees, including both sub-tree separation and in-sub-tree binarization, is referred by its unique index, which is denoted as *S*. The index *S* ranges between 0 to **MAXS**, where **MAXS** is defined as follows :

If **qbpp** < 10,

$$\mathbf{MAXS} = \min(4, \mathbf{RANGE}) + \sum_{k=1}^{\min(qbpp-1,\,5)} 2^k \cdot \min(6, \lceil \mathbf{RANGE} / 2^k \rceil) + \sum_{k=0}^{\min(qbpp-2,\,4)} 2^k \cdot \min(8, \lceil \mathbf{RANGE} / 2^k \rceil) + 2^{\min(qbpp-1,\,5)} \cdot \lceil \mathbf{RANGE} / 2^{\min(qbpp-1,\,5)} \rceil - 1$$

otherwise,

$$\mathbf{MAXS} = 3 + \sum_{k=1}^{\min(qbpp-2,\,9)} 2^k \cdot \min(8, \lceil \mathbf{RANGE} / 2^k \rceil) + 2^{\min(qbpp-1,\,10)} \cdot \lceil \mathbf{RANGE} / 2^{\min(qbpp-1,\,10)} \rceil$$

Figure A.18 shows the structure of the code tree separated into sub-trees in case of **qbpp** < 10. For the case of **qbpp** ≥ 10, the code tree shown in Figure A.19 is applied. In the figures, the merging of the sub-trees of different activity class trees is indicated by connecting the sub-trees of lower activity classes to those of the higher activity classes. If **RANGE** is not so large as to use all the sub-trees with mapped-error values, unnecessary sub-trees will be just omitted. At the last sub-tree separation, that means the largest mapped-error is contained in the sub-trees, the decision result of the sub-tree separation is obvious. Also if the number of leaves in a sub-tree is larger than the possible mapped error being left, which may occur especially in near-lossless cases, not all the leaves of the sub-trees are assigned mapped-errors, and not all the binarization patterns within the sub-tree will happen. Even in these cases, the arithmetic coder encodes the binary decisions associated with these obvious decisions. But in such obvious cases, since the probability of a symbol is set to 1.0 or 0.0 beforehand, it does not hurt the coding efficiency.

```
MErrvalTMP = MErrval;
while(1) {
    S = GetBinaryContext();
    k = GetGolombk(S);
    if(MErrvalTMP >= (1 << k) ) {
        ArithmeticEncode(1, S);
        MErrvalTMP = MErrvalTMP – (1 << k);
    }
    else {
        ArithmeticEncode(0, S);
        while(k - -) {
            S = GetBinaryContext();
            ArithmeticEncode((MErrvalTMP >> k) & 1, S);
        }
        break;
    }
}
```

Figure A.17 – Encoding of binary decisions with arithmetic coding

A.5.3 Mapped-error encoding

The sequence of binary decisions of the mapped error value, *MErrval*, shall be encoded as in Figure A.17, with the binary arithmetic coder specified in Annex B. The probability estimate of each binary decision is conditioned on the index of the node *S*, which is called binary context. Every node in the sub-tree structure or of binarization within sub-trees in Figure A.18 or A.19 has two counters, *LPScnt*[*S*] and *MLcnt*[*S*], which are a counter for the LPS (less probable symbol) and a counter for both of the LPS and MPS (more probable symbol), respectively. According to the data of the counters, the occurrence probability of a binary symbol is estimated.

The counters are initialised as *LPScnt*[*S*]=2 and *MLcnt*[*S*]=4 at binary contexts associated with sub-tree separation, while *LPScnt*[*S*]=4 and *MLcnt*[*S*]=8 for binary contexts within the sub-trees. Notice that the counters for the decisions on which one symbol never happens are initialized as *LPScnt*[*S*]=0 and *MLcnt*[*S*]=1.

A.6 Update variables

The last step of the encoding of the sample *x* is the update of the variables *A, B, C,* and *N*. It is important to note that this update shall be performed at the end of the coding procedure, after *k* and *MErrval* are computed.

A.6.1 Update

The variables *A*[*Q*], *B*[*Q*], and *N*[*Q*] are updated according to the current prediction error, as in Figure A.20. The variables *A*[*Q*] and *B*[*Q*] accumulate prediction error magnitudes and values for context *Q*, respectively. The variable *N*[*Q*] accounts for the number of occurrences of the context *Q* since initialization.

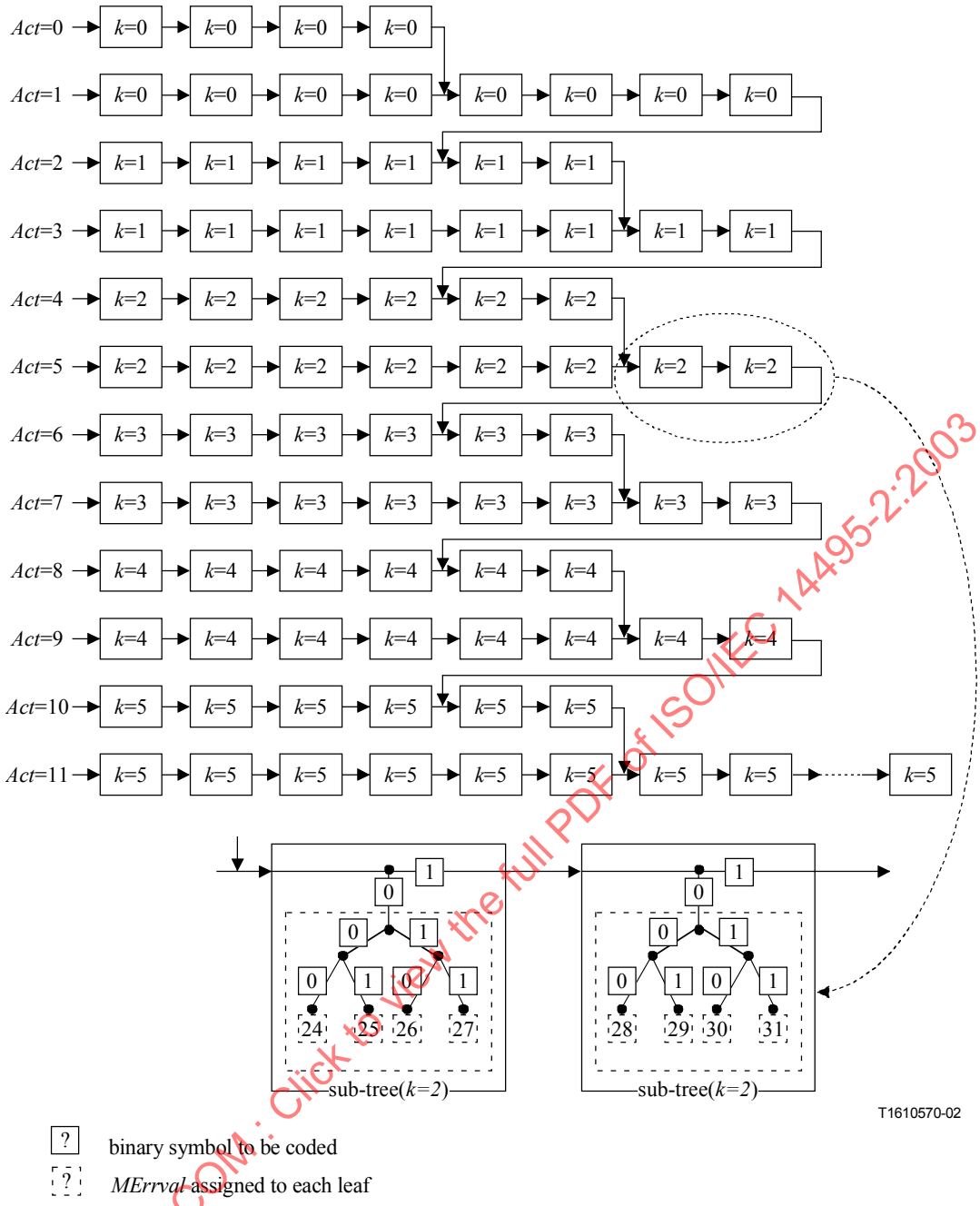

Figure A.18 – Binary decision tree for lossless coding with qbpp < 10

Figure A.19 – Binary decision tree for lossless coding with qbpp ≥ 10


```
if(Zerograd == 0){
    if(B[Q] > 0) B[Q] = B[Q] – Errval * (2 * nearq + 1);
    else         B[Q] = B[Q] + Errval * (2 * nearq + 1);
    if(Errval < 0)  A[Q] = A[Q] – 1;
    A[Q] = A[Q] + abs(Errval);
    if (N[Q] == RESET) {
            A[Q] = A[Q] >> 1;
            if (B[Q] >= 0)
                B[Q] = B[Q] >> 1;
            else
                B[Q] = -((1-B[Q]) >> 1);
            N[Q] = N[Q] >> 1;
    }
    N[Q] = N[Q] + 1;
}
```

Figure A.20 – Variables update

NOTE – In lossless coding, the value added to $B[Q]$ is the signed error, after modulo reduction.

RESET is a JPEG-LS coding parameter whose value is either default or set by the application (see Annex G).

If the extension of prediction is indicated, the procedure in Figure A.21 is performed instead of Figure A.20. In this procedure, *Flag*[*Rx*] indicating the occurrence of the reconstructed value *Rx* is also updated.

```
if(Zerograd == 0){
    if(B[Q] > 0) B[Q] = B[Q] – Errval * (2 * nearq + 1);
    else         B[Q] = B[Q] + Errval * (2 * nearq + 1);
    if(Errval < 0)  A[Q] = A[Q] – 1;
    A[Q] = A[Q] + abs(Errval);
    if (N[Q] == RESET) {
            A[Q] = A[Q] >> 1;
            if (B[Q] >= 0)
                B[Q] = B[Q] >> 1;
            else
                B[Q] = -((1-B[Q]) >> 1);
            N[Q] = N[Q] >> 1;
    }
    N[Q] = N[Q] + 1;
    Flag[Rx] = 1;
}
```

Figure A.21 – Variables update for the extension of prediction

A.6.2 Bias computation

The bias variable *B*[*Q*] allows an update of the prediction correction value *C*[*Q*] by at most one unit every iteration. The variables are clamped to limit their range of possible values. The prediction correction value *C*[*Q*] shall be computed according to the procedure in Figure A.22, which also yields an update of *B*[*Q*].

```
if(Zerograd == 0){
    if(2 * B[Q] <= - N[Q]) {
        B[Q] = B[Q] + N[Q];
        if(C[Q] > MIN_C)
            C[Q] = C[Q] - 1;
        if(2 * B[Q] <= - N[Q])
            B[Q] = - (N[Q] >> 1) + 1;
    } else if(2 * B[Q] > N[Q]) {
        B[Q] = B[Q] - N[Q];
        if(C[Q] < MAX_C)
            C[Q] = C[Q] + 1;
        if(2 * B[Q] > N[Q])
            B[Q] = (N[Q] >> 1);
    }
}
```

Figure A.22 – Update of bias related variables *B*[*Q*] and *C*[*Q*]

The constants **MIN_C** and **MAX_C** are defined in 3.3 of ITU-T Rec. T.87 | ISO/IEC 14495-1.

A.7 Flow of encoding procedures

The order in which the encoding procedures shall be performed is summarised below.

1) Initialization:

   a) Assign default parameter values to JPEG-LS preset coding parameters not specified by the application (see A.1).

   b) Initialize the non-defined samples of the causal template (see A.2.1).

   c) Compute the parameter **RANGE** (see A.2.1): For lossless coding, **RANGE** = **MAXVAL** +1. For near-lossless coding:

$$\mathbf{RANGE} = \left\lfloor \frac{\mathbf{MAXVAL} + 2 * \mathbf{NEAR}}{2 * \mathbf{NEAR} + 1} \right\rfloor + 1 .$$

   Compute the parameters **qbpp** = $\lceil \log \mathbf{RANGE} \rceil$, **bpp** = max(2,$\lceil \log(\mathbf{MAXVAL} + 1) \rceil$).

   d) For each context Q initialize four variables (see A.2.1): $A[Q] = \max\left(2, \left\lfloor \frac{\mathbf{RANGE} + 2^5}{2^6} \right\rfloor\right)$, $B[Q]$ = $C[Q] = 0$, $N[Q] = 1$, where Q is an integer between 0 and 1091.

   e) Initialize the counters for each binary decision. For each binary context S associated with unary representation of the mapped error value, *LPScnt*[S]=2, *MLcnt*[S]=4. For each binary context S associated with unary representation, *LPScnt*[S]=4, *MLcnt*[S]=8, and for each binary context S on which an LPS never happens, *LPScnt*[S]=0, *MLcnt*[S]=1. The sense of MPS *MPSvalue*[S] is initialised as *MPSvalue*[S]=0 for all the binary contexts.

   f) Initialize the error tolerance for near-lossless coding: *nearq*=**NEAR** (in lossless, *nearq*=0).

   g) If the extension of prediction is indicated, initialize the parameters: *Flag*[0]=1, and *Flag*[1..**MAXVAL**]=0.

   h) Set current sample to the first sample in the source image.

2) For the current sample, compute the local gradients according to Figure A.2.

3) Examine whether the sample to be coded is treated as to be in a uniform image area according to Figure A.3 or A.4. If the sample is judged to be in a uniform area a flag *Zerograd* is set to 1, otherwise *Zerograd* is set to 0.

4) Quantize the local gradients according to the steps detailed in Figures A.5 and A.6.

5) Check and change if necessary the signs of the components of the vector representing the context, modifying accordingly the variable *SIGN* (see A.3.4).

6) If near-lossless coding with visual-quantization is indicated, adjust the error tolerance according to Figure A.7.

7) Compute *Px* according to Figure A.8.

8) Correct *Px* using *C*[*Q*] and the variable *SIGN*, and clamp the corrected value to the interval [0..**MAXVAL**] according to the procedure in Figure A.9, or if the extension of prediction is indicated, according to Figure A.10, in which correction of *Px* is modified in order to avoid an unwanted predicted value.

9) Compute the prediction error and, if necessary, invert its sign according to the procedure in Figure A.11. Furthermore, if *Zerograd* is 0, the sign of the prediction error is flipped according to *B*[*Q*] as in Figure A.12.

10) For near-lossless coding, quantize the error and compute the reconstructed value of the current sample according to Figure A.13. For lossless coding, update the reconstructed value by setting *Rx* equal to *Ix*.

11) Reduce the error to the relevant range according to Figure A.14.

12) Perform the error mapping according to the procedure in Figure A.15.

13) Compute *Act* which determines the Golomb code structure to decompose the mapped error value according to the procedure in Figure A.16.

14) According to the procedure in Figure A.17, The mapped error value *MErrval* is decomposed into binary decisions by using the Golomb code tree determined by *Act* and the binary decisions are encoded with the arithmetic coder specified in Annex B.

15) Update the variables according to Figure A.20. If the extension of extension is indicated, the variables are updated according to Figure A.21.

16) Update the prediction correction value $C[Q]$ according to the procedure in Figure A.22.

17) Go to step 2 to process the next sample.

Annex B

Arithmetic coding

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies the method for using adaptive binary arithmetic coding by extending the syntax for lossless/near-lossless image compression specified in ITU-T Rec. T.87 | ISO/IEC 14495-1. Coding models for adaptive binary arithmetic coding are defined in Annex A. In this annex the arithmetic encoding and decoding procedures used in those models are defined.

B.1 Arithmetic encoding procedures

B.1.1 Binary arithmetic encoding principles

The arithmetic coder used in this specification encodes a series of binary symbols, zeros and ones, each symbol representing one possible result of a binary decision.

Recursive probability interval subdivision of the numerical line of [0,1) is the basis for the binary arithmetic encoding procedures. With each binary decision the current probability interval is subdivided into two sub-intervals, and the bit stream is modified (if necessary) so that it points to the base (the lower bound) of the probability sub-interval assigned to the symbol which occurred.

In the partitioning of the current probability interval into two sub-intervals, the sub-interval for the less probable symbol (LPS) and the sub-interval for the more probable symbol (MPS) are ordered such that the MPS sub-interval is closer to zero. Therefore, when the LPS is coded, the MPS sub-interval size is added to the bit stream. This coding convention requires that symbols be recognized as either MPS or LPS rather than 0 or 1. Consequently, the size of the LPS/MPS sub-interval must be known in order to encode that decision.

Figure B.1 shows an example of such interval division through an initial sequence 0, 1, 0, 0 to be coded.

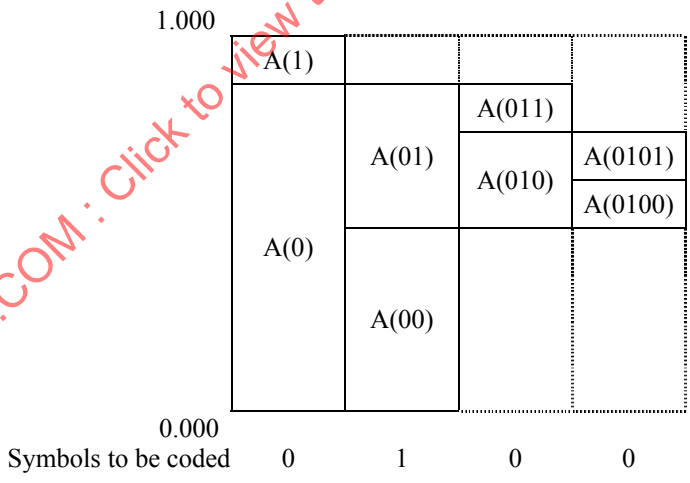

Figure B.1 – Interval subdivision of numerical line

B.1.1.1 Arithmetic operation in radix 255 representation

The encoding procedures use fixed precision integer arithmetic and an integer representation of fraction values. In this arithmetic coding, the numerical line data is treated in radix 255 representation, but one 255ary data is expressed with eight-bit binary data, which means each byte of the output data stream takes a value from X'00' to X'FE', though it could take a value up to X'FF' if the carry propagation is needed.

Consequently the probability interval, *Areg*, is kept in the integer range of X'FF' ≤ *Areg* < X'FF'×X'FF' with multiplying *Areg* by X'FF' whenever it falls less than or equal to X'FF'. The code register, *Creg*, containing the trailing bits of the bit stream, is also multiplied by X'FF' whenever *Areg* is multiplied by X'FF'.

The radix 255 representation has an advantage to let the byte of X'FF' not happen in the original data stream and let the byte of X'FF' can be used for the special case, which can avoid the accidental generation of markers in the output data stream without so-called zero insertion.

B.1.1.2 Probability estimation

An adaptive binary arithmetic coder requires a statistical model – a model for selecting conditional probability estimates to be used in the coding of each binary decision. When a given binary decision probability estimate is dependent on a particular feature or features (the context) already coded, it is "conditioned" on that feature. The conditioning of probability estimates on previously coded decisions must be identical in encoder and decoder, and therefore can use only information known to both.

In this arithmetic coding, the occurrence probability of LPS is estimated from the number of cumulative occurrences of the LPS, *LPScnt*[*S*], and the number counts of both of the LPS and MPS, *MLcnt*[*S*], which are dependent on each context, *S*.

B.1.1.3 Approximation of interval subdivision with OHP

The ideal LPS sub-interval using the estimated probability of LPS will be given by the following calculation:

Areg × *LPScnt[S] / MLcnt[S]*.

However this calculation needs multiplication and may decrease the processing speed. In the arithmetic coding used in this specification, in order to avoid such multiplication, one entry is chosen from the fixed value table data called Av table, based on the ratio of *LPScnt*[*S*] and *MLcnt*[*S*], and chosen Av is usually assigned for the LPS sub-interval independent to the current interval *Areg*. However, the degradation of the coding efficiency due to such approximation of subinterval matters especially when the probability of LPS is close to 0.5. Therefore to compensate the degradation of the coding efficiency when the probability of LPS is close to 0.5, OHP-"over-half processing", which is yet more simple than multiplication, is performed as follows.

Let us assume that *Areg* is between X'8000' to X'10000', and the value of **Av**[*i*] is chosen by comparing the *Prob* = (*LPScnt*[*S*]<<16)/*MLcnt*[*S*] with threshold value *Th*[]. If the (*Areg*–*Av*), which is the default sub-interval of MPS is greater than X'8000', the LPS interval will be *Av*. But if the (*Areg*–*Av*) is less than X'8000', half of the difference of (*Areg*–*Av*) and X'8000' is allocated to the MPS and the subinterval of the LPS will be decreased by that amount from *Av*. This process is described below and is called as OHP.

if((*Areg*–*Av*) >= X'8000') the LPS interval = *Av*;

else the LPS interval = (*Av*+*Areg*–X'8000')/2;

B.1.2 Procedures of arithmetic coding

B.1.2.1 Initialization

The following initializations shall be performed at the start of the arithmetic coding of the first binary decision of a frame.

The trailing bits of the code stream are stored in a variable *Creg* and the current probability interval is stored in a variable *Areg*. They are initialized as *Areg* = X'FF'×X'FF' and *Creg* = 0. *Buf*[0] and *Buf*[1] are two bytes of the code stream temporarily stored just after they are output from the code register.

MLcnt[*S*] and *LPScnt*[*S*] are the accumulated occurrence counts of both of the binary symbols and that of the LPS for each binary context *S*, respectively. The number of both counters shall be large enough to store the counts for all the contexts. Initial values of the counters are specified in A.2.1 and A.5.3. The maximum value of the counters **MAXcnt** is set to **MAXcnt** = 255. *MPSvalue*[*S*] is the sense of MPS for each binary context *S*, which takes 0 or 1. *MPSvalue*[*S*] is initialized as *MPSvalue*[*S*]=0 for all the binary contexts.

Preset **Av**[0...30] and **Th**[0...29]. The LPS probability is given by *Prob*=(*LPScnt*[*S*]<<16)/*MLcnt*[*S*] and by comparing *Prob* with **Th**[0...29], appropriate **Av**[0...30] is chosen.

```
Av[31] = {
0x7ab6, 0x7068, 0x6678, 0x5ce2, 0x53a6, 0x4ac0, 0x4230, 0x39f4,
0x33fc, 0x301a, 0x2c4c, 0x2892, 0x24ea, 0x2156, 0x1dd6, 0x1a66,
0x170a, 0x13c0, 0x1086, 0x0d60, 0x0b0e, 0x0986, 0x0804, 0x0686,
0x050a, 0x0394, 0x027e, 0x01c6, 0x013e, 0x0100, 0x0000};
Th[30] = {
0x7800, 0x7000, 0x6800, 0x6000, 0x5800, 0x5000, 0x4800, 0x4000,
0x3c00, 0x3800, 0x3400, 0x3000, 0x2c00, 0x2800, 0x2400, 0x2000,
0x1c00, 0x1800, 0x1400, 0x1000, 0x0e00, 0x0c00, 0x0a00,0x0800,
0x0600, 0x0400, 0x0300, 0x0200, 0x0180, 0x0001};
```

B.1.2.2 Search of suitable Av

The calculation of the subinterval is performed based on **Av**, which basically corresponds to the LPS interval. The suitable **Av** is given by comparing *Prob* = (*LPScnt*[*S*]<<16)/*MLcnt*[*S*] and **Th**.

Av are the values assuming *Areg* between X'8000' and X'10000', so **Av** and X'8000', half of full range, shall be modified by shifting *wct* bits if *Areg* is not in the range. The times of bit-shifting, *wct*, is determined from the value of *Areg*, i.e., from the most significant one-valued bit in *Areg* searched by the procedure in Figure B.2. A modified **Av** and X'8000' are denoted as *Avd* and *Hd* .

When the probability estimate of LPS reaches to the lowest probability 1/255, the *Avd* is forced to be 0x0002, which is less than the probability estimate given by *LPScnt*[*S*]/*MLcnt*[*S*].

```
Prob = (LPScnt[S] << 16) / MLcnt[S];
for(Aindex = 0; Aindex < 30; Aindex++)
        if(Prob > Th[Aindex]) break;
for(wct = 0; Areg < (0 x 8000 >> wct); wct++);
if( (MLcnt[S] == MAXcnt) && ( LPScnt[S] == 1) )  Avd = 0 x 0002;
else                                             Avd = Av[Aindex] >> wct;
Hd = 0 x 8000 >> wct;
```

Figure B.2 – Search of Av

NOTE – **Av** can be searched by a fast tree search. Since any thresholds, **Th**, are designated to be a power of two (2^i), or a point given by dividing the 2^i and 2^{i+1} into 2^j equal parts. An **Av** for the LPS can be determined by searching the number satisfying (*LPScnt*[*S*]<<*i*) <= *MLcnt*[*S*] < (*LPScnt*[*S*]<<(*i*+1)), which corresponds to $(1/2)^{i+1} < LPScnt[S]/\ MLcnt[S] <= (1/2)^i$, and then, in the range between $(1/2)^i$ and $(1/2)^{i+1}$, by choosing one of the divided parts which *LPScnt*[*S*]/ *MLcnt*[*S*] belongs to by simple shifting, subtracting and comparison operations.

B.1.2.3 Update of *Creg* and *Areg*

According to the LPS probability interval *Avd* and the binary decision *Bin*, the two registers *Creg* and *Areg* are updated by the procedure in Figure B.3. Notice that, in the calculation of *Creg* and *Areg* in the following procedure, the MPS probability interval, which may have been modified by so-called over-half processing, is temporarily stored in the variable *Avd*.

```
Avd = Areg-Avd;
if( Avd < Hd ) {          /* Over-half processing */
        Avd = (Avd + Hd)/2;
}

if (Bin==MPSvalue[s]) { /* MPS occurred */
        Areg = Avd;
} else {                /* LPS occurred */
        Creg = Creg + Avd;
        Areg = Areg - Avd;
}
```

Figure B.3 – Update of Creg and Areg

B.1.2.4 Update of counters

Basically, *MLcnt*[*S*] counter for both binary symbols are incremented after every binary decision is encoded and when it reaches to **MAXcnt**, both of the counters *MLcnt*[*S*] and *LPScnt*[*S*] are halved. However, the increment of the total counter is suspended if the LPS probability estimate reaches to the lowest probability until the LPS occurs. While the increment of the counters is being suspended, the *Avd* is forced to be 0x0002, which is a little smaller than the theoretical *Avd* corresponding to probability estimate of 1/**MAXcnt**. This is a simple treatment to improve the coding efficiency for high-skewed images.

```
if(MLcnt[S] == MAXcnt) {
    if(Bin != MPSvalue[S]) {
        MLcnt[S] = (MLcnt[S] + 1) / 2 + 1;
        LPScnt[S] = (LPScnt[S] + 1) / 2 + 1;
    } else if(LPScnt[S] != 1) {
        MLcnt[S] = (MLcnt[S] + 1) / 2 + 1;
        LPScnt[S] = (LPScnt[S] + 1) / 2;
    }
} else {
    MLcnt[S]++;
    if(Bin != MPSvalue[S]) LPScnt[S]++;
}
if(MLcnt[S] < LPScnt[S] * 2 ) {     /* change the sence of MPS */
    LPScnt[S] = MLcnt[S] - LPScnt[S];
    MPSvalue[S] = 1 - MPSvalue[S];
}
```

Figure B.4 – Update of counters

B.1.2.5 Renormalization of *Areg* and *Creg* and output data bit stream

Though the data steam is expressed in radix 255, integers in *Creg* and *Areg* are expressed in radix 2. Therefore, before putting out a byte from *Creg* to *Buf*[0] by renormalizing *Creg*, the output byte having a value expressed in radix 255 is calculated by (*Creg*>>8 + *Creg* +1) >>8 .

Propagation of carry-over into the bit stream from the code register can be stopped in the two saved bytes, *Buf*[0] and *Buf*[1], just by propagating carry-over bit into the saved bytes. If both *Buf*[1] and *Buf*[0] are X'FE' and carry-over from *Creg* occurrs, *Buf*[1] and *Buf*[0] will be exceptionally X'FF' and X'00', respectively. In this case, however, since the X'FF' byte is necessarily followed by a X'00' byte, proper detection of markers is possible.

```
if( Areg < 0x100 ) {
    if( Creg >= 0xff*0xff) {
        Creg -= 0xff*0xff;
        Buf[0]++;
        if(Buf[0] == 0xff) {
            Buf[0] = 0;
            Buf[1]++;
        }
    }
    AppendToBitStream(Buf[1], 8);
    Buf[1] = Buf[0];
    Buf[0] = ((Creg >> 8) + Creg + 1) >> 8;
    Creg += Buf[0];
    Creg = ((Creg & 0xff) << 8) - (Creg & 0xff);
    Areg = (Areg << 8) - Areg;
}
```

Figure B.5 – Renormalization of *Areg* and *Creg*

B.1.2.6 Termination of encoding

After encoding all the binary decisions, the encoder outputs four bytes of *Buf*[1], *Buf*[0] and *Creg*.

B.2 Arithmetic decoding procedures

B.2.1 Binary arithmetic decoding principles

The probability interval subdivision and sub-interval ordering defined for the arithmetic encoding procedures also apply to the arithmetic decoding procedures.

Since the bit stream always points within the current probability interval, the decoding process is a matter of determining, for each decision, which sub-interval is pointed to by the bit stream. This is done recursively, using the same probability interval sub-division process as in the encoder. Each time a decision is decoded, the decoder subtracts from the bit stream any interval the encoder added to the bit stream. Therefore, the code register in the decoder is a pointer into the current probability interval relative to the base of the interval.

The approximations and integer arithmetic defined for the probability interval subdivision in the encoder must also be used in the decoder. However, where the encoder would have added to the code register, the decoder subtracts from the code register.

B.2.2 Procedures of arithmetic decoding

B.2.2.1 Initialization for decoding

The following initializations shall be performed at the start of the arithmetic decoding.

The trailing bits of the code stream are stored in a variable *Creg* and the current probability interval is stored in a variable *Areg*. After the first two bytes of the code stream, which are always X'00', are removed, the third byte multiplied by X'FF' plus the fourth byte of the code stream shall be stored in *Creg* as its initial value. *Areg* is initialized as *Areg* = X'FF'×X'FF'.

MLcnt[*S*] and *LPScnt*[*S*], the accumulated occurrence counts of both the binary symbols and that of the LPS for each context *S*, are initialized as specified in A.2.1 and A.5.3. The maximum value of the counters **MAXcnt** are initialized as **MAXcnt** = 255.

Preset **Av**[0...30] and **Th**[0...29] by the same manner as in the encoding specified in B.1.2.1.

B.2.2.2 Search of suitable Av

A suitable **Av** is searched by the same procedure as in the encoding specified in Figure B.2.

B.2.2.3 Determination of a binary decision

A binary decision *Bin* is determined by comparing the MPS probability interval with the value of *Creg* as in Figure B.6. Also in the following procedure for decoding, the MPS probability interval, which may have been modified by over-half processing, is temporarily stored in the variable *Avd*.

```
Avd = Areg-Avd;
if( Avd < Hd ) {    /* over-half processing */
        Avd = (Avd + Hd)/2;

}

if (Creg < Avd) {    /* MPS occurred */
        Areg = Avd;
        Bin = MPSvalue[S];
} else {            /* LPS occurred */
        Creg = Creg-Avd;
        Areg = Areg-Avd;
        Bin = 1-MPSvalue[S];
}
```

Figure B.6 – Determination of a binary decision

B.2.2.4 Update of counters

The counters are updated by the same procedure as in the encoding specified in Figure B.4.

B.2.2.5 Renormalization of *Areg* and *Creg*

Whenever the current probability interval *Areg* becomes less than X'100', another byte is read from the code stream into *Creg*, and both *Areg* and *Creg* are renormalized by multiplying X'FF' as in Figure B.7.

```
if( Areg < 0 x 100 ) {
    Creg = (Creg << 8) - Creg + GetByte();
    Areg = (Areg << 8) - Areg;
}
```

Figure B.7 – Renormalization of *Areg* and *Creg*

Annex C

Encoding with arithmetic coding for multiple component images

(This annex forms an integral part of this Recommendation | International Standard)

C.1 Introduction

For encoding images with more than one component (e.g., colour images) using the arithmetic-based coding process, this Recommendation | International Standard supports combinations of single component scans and multi-component scans, as specified in Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1. For multi-component scans, two modes (described below) are supported: *line interleaved* and *sample interleaved*. The specific components per scan are specified in the scan header (see Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1), as well as the interleave mode (as specified by parameter **ILV**), which describes the structure within a single scan. The parameter **ILV** admits the values 0 (non-interleaved), 1 (line-interleaved) and 2 (sample interleaved).

For multi-component scans, a single set of context counters (*A, B, C and N*) is used across all the components in the scan. Also a single set of variables for the arithmetic coding (e.g., *MLcnt, LPScnt, MPSvalue*) is used across all the components. The prediction and context modelling procedures shall be performed as in the single component case, and are component independent, meaning that samples from one component are not used to predict or compute the context of samples from another component.

All the encoding and decoding variables (e.g., *A*[*0..1091*]) shall be set to their initial values, as described in Annex A, when a new scan is to be encoded (starting from Step 1 in A.7). The dimensions of each component are given by the information in the frame header.

C.2 Line interleaved mode

C.2.1 Description

This mode is specified by setting the parameter **ILV** in the start of scan marker segment to a value of 1. In this mode, for each component $\mathbf{C_i}$ in a scan, a set of $\mathbf{V_i}$ consecutive lines is encoded before starting the encoding of $\mathbf{V_{i+1}}$ lines of the subsequent component $\mathbf{C_{i+1}}$. The values $\mathbf{V_i}$ are specified in the start of frame marker segment as vertical sampling factors. For a scan with **Ns** components, the number of lines interleaved and encoded follows the sequence:

$\mathbf{V_1\ V_2 \ldots V_{Ns}}$, $\mathbf{V_1\ V_2 \ldots V_{Ns}}$, $\mathbf{V_1\ V_2 \ldots V_{Ns}}$, etc.

The flat region detection, prediction and context determination procedures shall be performed as in the single component mode, and do not use information from the multiple components. The flush procedure will be executed only at the end of the scan.

C.2.2 Process flow

The process flow for the line interleaved encoding mode is specified below, in terms of the general process flow given in A.7. For convenience, the steps are numbered identically to those in A.7.

1) Initialize a single set of variables as in Step 1 in the procedure described in A.7.

2) Follow Steps 2-16 in A.7 for the current sample in the current component (*i*). The reconstructed values *Ra*, *Rb*, *Rc*, *Rd* and *Re* used for context modelling and prediction correspond to the current component.

17) Return to Step 2. If all the samples of $\mathbf{V_i}$ consecutive lines of the *i*th component have been processed, continue with samples of component *i* + 1 (or component 1, if *i* was the last component). Otherwise, continue with samples from component *i*.

C.3 Sample interleaved mode

C.3.1 Description

This mode is specified by setting the parameter **ILV** in the start of scan marker segment to a value of 2. In this mode, one sample at a time per component shall be encoded. As in the line interleaved mode, the same counters shall be used across all components, and the flat region detection, prediction and context determination procedures shall be performed as in the single component mode, and shall not use information from the multiple components.

In this interleaved mode all components which belong to the same scan shall have the same dimensions.

C.3.2 Process flow

The process flow for the sample interleaved mode is described below in terms of the general process flow given in A.7. For convenience, the steps are numbered identically in this clause.

1) Initialize a single set of variables, as in step 1 in the procedure described in A.7.

2) Follow steps 2 to 16 in the procedure described in A.7 for each one of the current samples of each component. Steps 2-16 for the sample *j* of the component *i* shall be completed before the steps 2-16 for the sample *j* of the next component *i* + 1 are performed. Steps 2-16 of sample *j* + 1 of any component are not performed until these steps are completed for all the samples *j* for all the components. The same set of variables is used in these steps, but the flat region detection, context determination and prediction are performed for each component separately. The encoding of the sample *j* in component *i* + 1 uses the variables already updated by the sample *j* in the previous component *i*.

17) All the samples in the same position *j*, for all the components, have now been encoded. The encoder shall now return to step 2 above to continue with the sample in position *j* + 1 for all the components.

C.4 Minimum coded unit (MCU)

For non-interleaved mode (**Ns** = 1, **ILV** =0), the minimum coded unit is one line. For sample interleaved mode (**Ns** > 1, **ILV** = 2), the MCU is a set of **Ns** lines, one line per component, in the order specified in the scan header.

NOTE – The order in the scan header is determined by the order in the frame header.

For line interleaved mode (**Ns** > 1, **ILV** = 1), the MCU is $\mathbf{V_1}$ lines of component $\mathbf{C_1}$ followed by $\mathbf{V_2}$ lines of component $\mathbf{C_2}$ … followed by $\mathbf{V_{Ns}}$ lines of component $\mathbf{C_{Ns}}$. In addition, the encoding process shall extend the number of lines if necessary so that the last MCU is completed. Any line added by an encoding process to complete a last partial MCU shall be removed by the decoding process.

Annex D

Extended functions for the baseline coding model

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies extended functions of near lossless coding, prediction and Golomb coding, which are applied to the baseline coding process. The extensions of near-lossless coding and prediction can be also applied to the arithmetic-based coding model. The procedures on the arithmetic-based coding model are described in Annex A.

D.1 Extensions of near-lossless coding

D.1.1 Near-lossless coding with visual quantizaion

The operation of near-lossless coding with visual quantization is indicated by the LSE marker with **ID**=X'07', which may be present where tables or miscellaneous marker segments may appear (see Annex G).

To apply this extended function to the encoding procedures using Golomb coding (baseline coding process) specified in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1, the description of the procedures shall be modified as follows:

a) In A.2.1, initialize the visual quantization threshold **TQ** as is specified in the LSE marker segment.

b) After A.3.4, set the error tolerance for near-lossless coding using visual quantization as follows :

```
if(|Q1|+|Q2|+|Q3|>=TQ)
    nearq = NEAR+1;
else
    nearq = NEAR;
```

c) A.4.4 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "*nearq*" replacing "**NEAR**".

d) A.5.2 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "*nearq*" replacing "**NEAR**".

e) A.6.1 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "*nearq*" replacing "**NEAR**".

The procedure of this extended function in the arithmetic-based coding process is described in Annex A.

D.1.2 Near-lossless coding with NEAR value re-specification

The operation of near-lossless coding with NEAR value re-specification is indicated by the LSE marker segment with **ID**=X'06', which specify the new NEAR value applied to the near-lossless coding after this and the number of MCUs encoded with the previous NEAR value. The LSE marker segment with **ID**=X'06' may be present where tables or miscellaneous marker segments may appear or in the middle of the coded image data segment (see Annex G). Therefore, a X'FF' byte in a coded image data segment is not followed by a stuffed '0', if it is part of an inserted marker segment. The control procedure for encoding an MCU is also modified as described in Annex H. With this marker segment, the difference bound related to the regular mode and the one related to the run mode can be specified independently by specifying **NEAR** and **NEARRUN** as described in G.1.2.2.

To apply this extended function to the baseline coding process specified in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1, the description of the procedures shall be modified as follows :

a) A.3.2 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "**NEARRUN**" replacing "**NEAR**".

b) A.7.1.1 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "**NEARRUN**" replacing "**NEAR**".

c) A.7.2 is the same as in ITU-T Rec. T.87 | ISO/IEC 14495-1 but with "**NEARRUN**" replacing "**NEAR**". Also, in Quantize(Error) called in code segment A.19, which is specified in code segment A.8, "**NEAR**" is replaced with "**NEARRUN**".

After **NEAR** (and **NEARRUN**) are re-specified, the values of the parameters **T1**, **T2**, and **T3** are reverted to the default values corresponding to the new value of **NEAR**. If an LSE marker segment with **ID**=X'01' or **ID**=X'0C' follows the **LSE** marker segment with **ID**=X'06', these thresholds may be redefined. **RESET** and **MAXVAL** may also be changed by this segment. Next, the variables **RANGE**, **LIMIT**, **qbpp**, *A*[], *B*[], *C*[], *N*[], *Nn*[], and *RUNindex* are reset as if a restart marker was detected. If the extended prediction is used, the number of elements of *SPf*[0..**RANGE**], *SPm*[0..**RANGE**] are adjusted corresponding to the new value of **RANGE**, and *SPx*, *SPt*, *SPf*[], *SPm*[] are also reset. Note that **RANGE** and **qbpp** used in the regular mode and those in the run-interruption sample encoding are different if **NEAR** and **NEARRUN** are not identical. In this case, **RANGE** and **qbpp** for the run-interruption sample encoding shall be additionally prepared according to the re-specified **NEARRUN**.

D.2 Extensions of prediction on baseline coding model

This clause specifies the procedure of the extended prediction by describing the changes and additions to the encoding procedures defined in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1. The procedure of extension of prediction for the arithmetic-based coding process is specified in Annex A.

D.2.1 Initializations

Initializations are the same as A.2.1 of ITU-T Rec. T.87 | ISO/IEC 14495-1 but adding the following initializations.

5) Initialize the variables for symbol packing: *SPx*=1, *SPt*=1, *SPf*[0]=1, *SPf*[1..**RANGE**]=0, *SPm*[0..**RANGE**]={0, 1, 2, 3, … , **RANGE**–1,**RANGE**}

6) Initialize the variable *Flag*[0..**MAXVAL**], *Flag*[0] is initialized with the value 1, and *Flag*[1..**MAXVAL**] with the value 0.

D.2.2 Prediction correction

The procedure described in A.4.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1 is replaced by the following procedure.

After *Px* is computed, the prediction shall be corrected according to the procedure depicted in Figure D.1 which depends on *SIGN*, the sign detected in the context determination procedure. The new value of *Px* shall be clamped to the range [0..**MAXVAL**]. The prediction correction value *C*[*Q*] is derived from the bias as specified in A.6.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1.

```
Pmed = Px;
if (SIGN == +1)
   Px = Pmed +C[Q];
else
   Px = Pmed – C[Q];

if (Px>MAXVAL)
   Px=MAXVAL;
else if(Px <0)
   Px=0;

if(Flag[Px]==0) {
   if (Px<Pmed)
      for (Px++ ; Flag[Px]==0;Px++);
   else
      for (Px--; Flag[Px]==0;Px--);
}
```

Figure D.1 – Prediction correction

D.2.3 Symbol packing

After the mapped error value, *MErrval* is computed in A.5.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1, symbol packing is carried out, as described in Figure D.2.

```
TMErrval = MErrval;

MErrval = SPm[TMErrval];

if (SPf[TMErrval] == 0 && TMErrval<SPt) MErrval = MErrval + SPx;
```

Figure D.2 – Symbol packing

D.2.4 Update variables

The update of variables is the same as A.6 of ITU-T Rec. T.87 | ISO/IEC 14495-1 except for the following modifications.

a) In code segment A.12 of ITU-T Rec. T.87 | ISO/IEC 14495-1,

A[*Q*] = *A*[*Q*]+abs(*Errval*);

is replaced by:

A[*Q*] = *A*[*Q*]+((*MErrval*+1)>>1);

as described in Figure D.3.

```
B[Q] = B[Q] + Errval * (2 * NEAR + 1);
A[Q] = A[Q] + ((MErrval+1) >> 1);
if (N[Q] == RESET) {
        A[Q] = A[Q] >> 1;
        if (B[Q] >= 0)
                B[Q] = B[Q] >> 1;
        else
                B[Q] = -((1-B[Q]) >> 1);
        N[Q] = N[Q] >> 1;
}
N[Q] = N[Q] + 1;
```

Figure D.3 – Variable update

b) After bias computation specified in A.6.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1, the procedure to update parameters for the extended prediction is carried out as in Figure D.4.

```
Flag[Rx] = 1;
if (SPf[TMErrval] == 0) {
   if(TMErrval >= SPt) {
      for (i=TMErrval-1; i>=SPt; i-- ) SPm[i] =SPm[i]-SPx;
      SPt = TMErrval+1;
      SPm[TMErrval]=SPx;
   }
   else {
      for (i=SPt-1; i>TMErrval; i--) {
         if(SPf[i]) {
            SPm[TMErrval]=SPm[i];
            SPm[i]++;
         }
         else SPm[i]--;
      }
   }
   SPf[TMErrval] = 1;
   SPx ++;
}
```

Figure D.4 – Update of variables for symbol packing

D.2.5 Run interruption sample encoding

The procedure of the run interruption sample encoding is the same as A.7.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1 except for the following changes.

a) Code segment A.22 of ITU-T Rec. T.87 | ISO/IEC 14495-1 is replaced by Figure D.5 described below.

TEMErrval = 2*abs(E*rrval*) – *map*;

Figure D.5 – Errval mapping for run interruption sample

b) After the mapped error value, *TEMErrval* is computed, symbol packing is carried out, as indicated in Figure D.6.

EMErrval = *SPm*[*TEMErrval*] - *RItype*;

if (*SPf*[*TEMErrval*] == 0 && *TEMErrval*<*SPt*) *EMErrval* = *EMErrval*+*SPx*;

Figure D.6 – Symbol packing for run interruption sample

c) Change procedure 9 of A.7.2 of ITU-T Rec. T.87 | ISO/IEC 14495-1 as follows:

9) Update the variables for run interruption sample encoding, according to code segment A.23 of ITU-T Rec. T.87 | ISO/IEC 14495-1. Furthermore, in case of *TEMErrval* is the first found, variables for symbol packing shall be updated, following the same procedure as in the regular mode, Figure D.4, but using *TEMErrval* instead of *TMErrval*.

D.2.6 Flow of encoding procedures

The flow of encoding procedures is the same as A.8 of ITU-T Rec. T.87 | ISO/IEC 14495-1 except for the following changes.

In step 1, add the following steps.

h) Initialize the variables for symbol packing: *SPx*=1, *SPt*=1, *SPf*[0]=1, *SPf*[1..**RANGE**]=0, *SPm*[0..**RANGE**]={0, 1, 2, 3, … , **RANGE**–1,**RANGE**}

i) Initialize the variable *Flag*[0..**MAXVAL**], *Flag*[0] is initialized with the value 1, and *Flag*[1..**MAXVAL**] with the value 0.

Change step 7 as follows.

7) Correct *Px* using *C*[*Q*] and the variable *SIGN*, and clamp the corrected value to the interval [0..**MAXVAL**] according to the procedure in Figure D.1.

Change step 13 as follows.

13) Compute the mapped error value *MErrval* according to the procedure in Figure D.2, and encode using the limited length Golomb code function *LG*(*k*,**LIMIT**), as specified in A.5.3 of ITU-T Rec. T.87 | ISO/IEC 14495-1.

Change step 14 as follows.

14) Update the variables according to Figures D.3 and D.4.

D.3 Extension of Golomb coding

This clause specifies the two modifications of Golomb coding, Golomb code completion and run interruption handling (only for bi-level images), which will improve the coding efficiency of Golomb coding specified in Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1. The operation of the extended Golomb coding is indicated by the LSE marker segment with **ID**=X'05' and **ENT**=X'01' (see Annex G).

D.3.1 Golomb code completion

In order to complete the Golomb code for the usual case in which **RANGE** is a power of 2 (namely, **RANGE** = $2^{\mathbf{qbpp}}$), which includes the binary case (**qbpp**=1) in which an incomplete code is particularly damaging, the coding procedure of A.5.3 of ITU-T Rec. T.87 | ISO/IEC 14495-1 shall be modified by inserting the phrase: "if the unary part contains $2^{\mathbf{qbpp}-\mathbf{k}}-1$ zeros, then the terminating one shall be omitted, as shown below. The validity of this procedure is based on the fact that, in regular mode, the mapped error value never exceeds $2^{\mathbf{qbpp}}-1$ (it can reach **RANGE** only in cases in which **RANGE** is odd)."

D.3.1.1 Mapped-error encoding with Golomb code completion

The mapped error value, *MErrval,* shall be encoded with the limited length Golomb code function *LG*(*k*, **LIMIT**) defined by the following procedure:

1) If the number formed by the high order bits of *MErrval* (all but the *k* least significant bits) is less than **LIMIT** – **qbpp** – 1, this number shall be appended to the encoded bit stream in unary representation, that is, by as many zeros as the value of this number, followed by a binary one. The *k* least significant bits of *MErrval* shall then be appended to the encoded bit stream without change, with the most significant bit first, followed by the remaining bits in decreasing order of significance.

2) Otherwise, **LIMIT** – **qbpp** –1 zeros shall be appended to the encoded bit stream, followed by a binary one. The binary representation of *MErrval* – 1 shall then be appended to the encoded bit stream using **qbpp** bits, with the most significant bit first, followed by the remaining bits in decreasing order of significance.

3) If the unary part contains $2^{\mathbf{qbpp-k}}-1$ zeros, then the terminating one shall be omitted.

For run interruption coding, the mapped error value can be **RANGE** also in cases in which **RANGE** is even. To avoid this situation, the procedure in code segment A.21 of ITU-T Rec. T.87 | ISO/IEC 14495-1 shall be modified adding an additional "else if" statement just before the final "else" statement as in Figure D.7. Notice that by this modification, it is no longer valid that every code word contains at least one '1' except for runs of length zero. This may render the handling of zeros inserted for byte completion before a marker somewhat more intricate.

```
if ((k == 0) && (Errval > 0) && (2 * Nn[Q] < N[Q]))
        map = 1;
else if ((Errval < 0) && (2 * Nn[Q] >= N[Q]))
        map = 1;
else if ((Errval < 0) && (k != 0))
        map = 1;
else if ((2 * Errval == - RANGE) && (RItype == 0))
        map = 1;
else
        map = 0;
```

Figure D.7 – Computation of map for Errval mapping

D.3.2 Run interruption handling for qbpp=1

In case the mode is not sample interleaved and **qbpp**=1, the encoding of the run interruption sample is superfluous and shall be omitted. Notice that in sample interleaved mode this is not the case, as a run may be interrupted by a sample of a different value in some component, while in another component the corresponding sample value equals that of the run. Hence, run interruption coding is still required in that case. When this option is specified, the procedure 18 in the flow of encoding procedure described in A.8 of ITU-T Rec. T.87 | ISO/IEC 14495-1 is replaced by "Go to Step 16".

Annex E

Fixed length coding

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies a method to encode image samples by using a fixed length code instead of a Golomb code.

E.1 Introduction

The operation of fixed length coding is indicated by specifying a corresponding LSE marker segment. The LSE marker segments may be present where tables or miscellaneous marker segments may appear, or in the middle of coded image data segments, that is, a number of MCUs in a scan can be encoded with fixed length code. When fixed length coding is performed partially in a scan, the start and the end of MCUs encoded with fixed length coding are specified by the LSE marker segments with **ID**=X'09' and **ID**=X'0A', respectively (see Annexes G and H).

NOTE – It is not allowed to use a fixed length code in the arithmetic-based coding process.

E.2 Fixed length coding

To encode samples with a fixed length code, the following procedures are performed instead of the procedures specified in A.3 to A.7 of Annex A of ITU-T Rec. T.87 | ISO/IEC 14495-1.

a) Compute $Qx = (Ix + \mathbf{NEAR})/(2*\mathbf{NEAR} + 1)$;

b) Encode Qx in binary using $\lceil \log \left(\left\lfloor \frac{\mathbf{MAXVAL} + \mathbf{NEAR}}{2 * \mathbf{NEAR} + 1} \right\rfloor + 1 \right) \rceil$ bits (Note that this number differs from **qbpp**). To this end, use the **AppendToBitStream** function.

c) Compute $Rx = Qx * (2*\mathbf{NEAR}+1)$ (Note that Rx is needed if the next MCU is not encoded with a fixed-length code, and also to maintain the definition of the decoder requirements, which relies on Rx).

d) When needed, perform bit stuffing as in the rest of Annex A.

Annex F

Sample transformation for inverse colour transform

(This annex forms an integral part of this Recommendation | International Standard except F.2)

F.1 Inverse colour transform

In this Recommendation | International Standard, a procedure to reconstruct source image data from corresponding values of decoded samples in a subset $\mathbf{C_{i(1)}}$, $\mathbf{C_{i(2)}}$, ... , $\mathbf{C_{i(Nt)}}$, of **Nt** components of a frame is specified. Let R_1, R_2, … , R_{Nt} denote the samples output by the decoding process for a given sample location in the respective components $\mathbf{C_{i(1)}}$, $\mathbf{C_{i(1)}}$, $\mathbf{C_{i(2)}}$, ... , $\mathbf{C_{i(Nt)}}$. These samples are inverse-transformed into output image samples S_1, S_2, … , S_{Nt} having the same precision **P** as the decoded samples. The participating components and a description of the inverse transform are specified in an inverse colour transform specification marker segment (see G.1.2.8).

Let

$$\mathbf{HALFTRANS} = \lfloor (\mathbf{MAXTRANS}+1)/2 \rfloor .$$

The inverse-transformed values S_i, $1 \le i \le \mathbf{Nt}$, are computed as follows:

For $i = 1, 2, \ldots, \mathbf{Nt}$ do

$$\hat{R}_i = R_i - \mathbf{HALFTRANS} \cdot (1 - \mathbf{CENTER}_i)$$

For $i = 1, 2, \ldots, \mathbf{Nt}$ do

If $\mathbf{CENTER_i} = 0$ then

$$S_i = \hat{R}_i + \left\lfloor \frac{\sum_{j=1}^{i-1} \mathbf{A}_{i,j} S_j + \sum_{j=i+1}^{Nt} \mathbf{A}_{i,j-1} \hat{R}_j}{2^{\mathbf{NORM}_i}} \right\rfloor$$

Else

$$S_i = \hat{R}_i - \left\lfloor \frac{\sum_{j=1}^{i-1} \mathbf{A}_{i,j} S_j + \sum_{j=i+1}^{Nt} \mathbf{A}_{i,j-1} \hat{R}_j}{2^{\mathbf{NORM}_i}} \right\rfloor$$

Reduce S_i modulo **(MAXTRANS+1)** to the range S_i, $0 \le S_i \le \mathbf{MAXTRANS}$.

NOTE – Usually, the specified inverse transform shall be used to reverse (in a lossless manner) a corresponding forward transform which is applied to an image and generates source image data. The corresponding forward transform must be amenable to a "lifting" expression of the form $R_i = S_i + f(S_1, S_2, \ldots, S_{i-1}, R_{i+1}, R_{i+2}, \ldots, R_{Nt})$. This condition ensures lossless reconstruction without any restriction on the function f( ), since the function involves sample values that are recursively available to both the encoding and the decoding process. The transform description specified in this Recommendation | International Standard restricts the functions f( ) to be normalized linear combinations (with coefficients $\mathbf{A_{i,j}}$ normalized by a power-of-2), with integer constraints guaranteed by the "floor" brackets. The linear combinations and normalizations (namely, the operations inside the brackets) are carried out over the integers, whereas the additions/subtractions outside the brackets are carried out modulo a specified value **(MAXTRANS+1)** to preserve the precision. The lossless property requires that **MAXTRANS** be at least as large as the maximum value of the samples in the participating components. In addition, the lossless property assumes that the range of possible values adopted by the function f( ) is no wider than **(MAXTRANS**+1**)**, and it is centered either at 0 or at **HALFTRANS**. Function value ranges that are centered at 0 are assumed to be added and the result reduced to the range [0..**MAXTRANS**] in the forward transform, to generate source image data. Function value ranges that are centered at **HALFTRANS** are assumed to be subtracted and the result reduced to the range [–**HALFTRANS** .. **HALFTRANS**–1]. Further addition of **HALFTRANS** after forward transformation ensures that the source image data take non-negative values. The parameter $\mathbf{CENTER_i}$ specifies the center of each interval (0 for 0, 1 for **HALFTRANS**). This inverse transform description covers approximations of all commonly used forward lossless transforms.

F.2 Example and guideline (Informative)

Assume an image with three components labeled R, G and B. The precision is **P**=8, and **MAXVAL**=255 for all components. The goal is to approximate the following forward transform, where R, G and B are values in the range $0 \le R, G, B \le 255$:

$$R' = R - G,$$

$$B' = B - G,$$

$$G' = G + \frac{R' + B'}{4}.$$

A "lifting" approximation of the above transform, which also preserves precision, is given by:

$R' = R - G$ (reduced modulo 256 to the range $-128 \le R' \le 127$),

$B' = B - G$, (reduced modulo 256 to the range $-128 \le B' \le 127$),

$$G' = G + \left\lfloor \frac{R' + B'}{4} \right\rfloor$$ (reduced modulo 256 to the range $0 \le G' \le 255$).

In the last assignment, the operations inside the "floor" brackets are carried out over the integers, whereas the addition outside the brackets is carried out modulo 256. The values R' and B' are in the range $-128 \le R', B' \le 127$, but for the purpose of encoding, all components are shifted to the non-negative range by the following transformation:

$$R'' = R' + 128,$$

$$B'' = B' + 128,$$

$$G'' = G''.$$

The encoding process described in this Recommendation | International Standard is applied to the transformed image given by the components R'', G'' and B''. In this example, it is assumed that the encoding is lossless, i.e., **NEAR**=0. On the decoding side, the inverse transform, which recovers R, G and B from R'', G'' and B'', is given in two stages. In the first stage, R', B' and G' are recovered as follows:

$$R' = R'' - 128,$$

$$B' = B'' - 128,$$

$$G' = G''.$$

In the second stage, R, G and B are recovered as follows:

$$G = G' - \left\lfloor \frac{R' + B'}{4} \right\rfloor$$ (reduced modulo 256 to the range $0 \le G \le 255$),

$R = R' + G$ (reduced modulo 256 to the range $0 \le R \le 255$),

$B = B' + G$ (reduced modulo 256 to the range $0 \le B \le 255$),

In the computation of G, the operations inside the "floor" brackets are carried out over the integers, whereas the subtraction outside the brackets is carried out modulo 256.

The inverse transform is described in an inverse transform specification marker segment. To describe the marker segment, assume first that the components R'', G'' and B'' were labeled, in the SOF marker segment, with the ID numbers X'01', X'02' and X'03' respectively. Then, the inverse transform specification marker segment is constructed as follows:

FF F8 # LSE marker

00 18 # Length of marker segment = 24 bytes including the length field

0D # **ID**=X'0D', inverse transform specification marker segment

00 FF # **MAXTRANS**=255

03 # **Nt**=3

02 # **ID** of first component in the transform (G'')

01 # **ID** of second component in the transform (R'')

03 # **ID** of third component in the transform (B'')

82 # $\mathbf{F}_1$: $\mathbf{CENTER}_1$=1, $\mathbf{NORM}_1$=2

00 01 # $\mathbf{A}_{1,1}$=1

00 01 # $\mathbf{A}_{1,2}$=1

00 # $\mathbf{F}_2$: $\mathbf{CENTER}_2$=0, $\mathbf{NORM}_2$=0

00 01 # $\mathbf{A}_{2,1}$=1

00 00 # $\mathbf{A}_{2,2}$=0

00 # $\mathbf{F}_3$: $\mathbf{CENTER}_3$=0, $\mathbf{NORM}_3$=0

00 01 # $\mathbf{A}_{3,1}$=1

00 00 # $\mathbf{A}_{3,2}$=0

The components R_1, R_2 and R_3 correspond to G'', R'' and B'', respectively, in this example. We have **HALFTRANS**=128, and:

$$G' = \hat{R}_1 = R_1 - 128 \cdot (1 - \mathrm{CENTER}_1) = G'',$$
$$R' = \hat{R}_2 = R_2 - 128 \cdot (1 - \mathrm{CENTER}_2) = R'' - 128,$$
$$B' = \hat{R}_3 = R_3 - 128 \cdot (1 - \mathrm{CENTER}_3) = B'' - 128.$$

Now,

$$G = S_1 = \hat{R}_1 - \left\lfloor \frac{\mathbf{A}_{1,1}\hat{R}_2 + \mathbf{A}_{1,2}\hat{R}_3}{2^{\mathbf{NORM}_1}} \right\rfloor = G' - \left\lfloor \frac{R' + B'}{4} \right\rfloor \text{ (reduced modulo 256 to } 0 \le G \le 255),$$

$$R = S_2 = \hat{R}_2 + \left\lfloor \frac{\mathbf{A}_{2,1}S_1 + \mathbf{A}_{2,2}\hat{R}_3}{2^{\mathbf{NORM}_2}} \right\rfloor = R' + G \text{ (reduced modulo 256 to } 0 \le R \le 255),$$

$$B = S_3 = \hat{R}_3 + \left\lfloor \frac{\mathbf{A}_{3,1}S_1 + \mathbf{A}_{3,2}S_2}{2^{\mathbf{NORM}_3}} \right\rfloor = B' + G \text{ (reduced modulo 256 to } 0 \le B \le 255).$$

Numerical example:

Assume $R = 200$, $G = 10$, $B = 55$ in the original image. The following computations are performed on the encoding side, where "→" denotes reduction modulo 256 to the range specified by the procedure.

$$R' = 200 - 190 = 190 \rightarrow -66,$$
$$B' = 55 - 10 = 45 \rightarrow 45,$$
$$G' = 10 + \left\lfloor \frac{-66 + 45}{4} \right\rfloor = 4 \rightarrow 4,$$
$$R'' = -66 + 128 = 62,$$
$$B'' = 45 + 128 = 173,$$
$$G'' = 4.$$

The values R'', G'', B'' are input to the encoding process.

On the decoding side, the values R'', G'', B'' are output by the decoding process. The following computations effect the inverse colour transform:

$$G' = G'' = 4,$$
$$R' = R'' - 128 = -66,$$
$$B' = B'' - 128 = 45,$$
$$G = 4 - \left\lfloor \frac{-66 + 45}{4} \right\rfloor = 4 + 6 = 10 \rightarrow 10,$$
$$R = -66 + 10 = -56 \rightarrow 200,$$
$$B = 45 + 10 = 55 \rightarrow 55.$$

Annex G

Compressed data format

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies three compressed data formats for the extended JPEG-LS processes:

1) the interchange format;
2) the abbreviated format for compressed image data;
3) the abbreviated format for mapping tables and parameters specification data.

These compressed data formats closely follow the compressed format specified in Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1, and this annex specifies only the changes and additions to Annex C of ITU-T Rec. T.87 | ISO/IEC 14495-1.

G.1 General aspects of the compressed data format specification

G.1.1 Marker assignments

All the marker segments specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 are compatible in this Recommendation | International Standard. In order to indicate to use extended functions specified in this Recommendation | International Standard, a new frame marker, **SOF$_{57}$** (X'FFF9') assigned from the **JPEG$_n$** set, is used. The syntax of the new frame marker segment is identical to the start of frame marker specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 (**SOF$_{55}$**, X'FFF7'). Extended functions of JPEG-LS extension are identified by using JPEG-LS preset parameters marker, **LSE** (X'FFF8'), but with the table **ID** greater than X'04'. The details are described below.

G.1.2 JPEG-LS preset parameters specification syntax

The **LSE** marker segment may be present where tables or miscellaneous marker segments may appear, except for the cases the table ID=X'06', X'09' and X'0A', and **ID**=X'01' or **ID**=X'0C' followed by an LSE marker segment with **ID**=X'06'. If tables specified in this marker segment for a given table ID appear more than once, each specification shall replace the previous specification. Figure G.1 specifies the marker segment following an **LSE** marker which defines parameters specified in ITU-T Rec. T.87 | ISO/IEC 14495-1 and specified only in this Recommendation | International Standard, and are collectively called "JPEG-LS preset parameters".

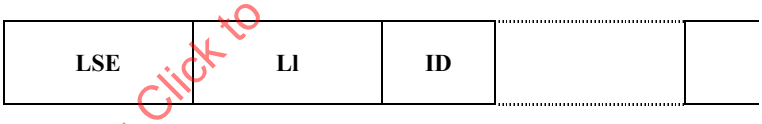

Figure G.1 – JPEG-LS preset parameters marker segment syntax

The marker and parameters, shown in Figure G.1 are defined below.

LSE: JPEG-LS preset parameters marker; marks the beginning of the JPEG-LS preset parameters marker segment.

LI: JPEG-LS preset parameters length; specifies the length of the JPEG-LS preset parameters marker segment shown in Figure G.1.

ID: parameter ID; specifies which JPEG-LS preset parameters follow. The specifications for **ID**=X'01' to **ID**=X'04' are identical to those of ITU-T Rec. T.87 | ISO/IEC 14495-1. Specifications for **ID**=X'05' to **ID**=X'0D' are defined only for JPEG-LS extension.

G.1.2.1 Coding method specification

Figure G.2 specifies the entropy coder when **ID** is equal to X'05'.

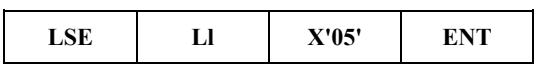

Figure G.2 – LSE marker segment for coding method specification

The new parameters shown in Figure G.2 are defined below. The size and allowed values of each parameter are given in Table G.1.

ENT: **ENT**=0 indicates to use the Golomb coding (baseline coding process) specified in ITU-T Rec. T.87 | ISO/IEC 14495-1. **ENT**=1 indicates to use the baseline coding process with the extended Golomb coding newly specified in this Recommendation | International Standard. **ENT**=2 indicates to use the arithmetic coding (arithmetic-base coding process) newly specified in this Recommendation | International Standard.

Table G.1 – LSE marker segment parameter sizes and values for coding method specification

Parameter	Size(bits)	Values
Ll	16	4
ID	8	X'05'
ENT	8	0, 1, 2

The default value of **ENT** is 0, which indicates Golomb coding specified in ITU-T Rec. T.87 | ISO/IEC 14495-1.

G.1.2.2 NEAR value re-specification (floating marker)

Figure G.3 specifies the re-specification of the NEAR parameter when **ID** is equal to X'06'.

LSE	Ll	X'06'	NEAR	NEARRUN	NMCU

Figure G.3 – LSE marker segment for NEAR value re-specification

The new parameters shown in Figure G.3 are defined below. The size and allowed values of each parameter are given in Table G.2.

NEAR: Re-specified **NEAR** value.

NEARRUN: **NEAR** value applied for the run mode.

NMCU: The number of MCUs encoded with the previous NEAR value.

Table G.2 – LSE marker segment parameter sizes and values for NEAR value re-specification

Parameter	Size(bits)	Values
Ll	16	8
ID	8	X'06'
NEAR	8	$0 \le \mathbf{NEAR} \le \min(255, \left\lceil \frac{\mathbf{MAXVAL}}{2} \right\rceil)$
NEARRUN	8	$0 \le \mathbf{NEARRUN} \le \min(255, \left\lceil \frac{\mathbf{MAXVAL}}{2} \right\rceil)$
NMCU	32	$0 \le \mathrm{NMCU} \le 2^{32}-1$

An LSE marker segment with **ID**=X'06' may be present where tables or miscellaneous marker segments may appear or in a coded image data segment. An **LSE** marker segmetn with **ID**=X'01' or **ID**=X'0C' may also be present in a code image data segment immediately following an LSE marker with **ID**=X'06'. The values of the parameters **NEAR** and **NEARRUN** overrides values specified in previous **LSE** marker segments with **ID**=X'06' or in the scan header. The values of the parameters **T1**, **T2**, and **T3** (and **T4** if the arithmetic-based coding process is used) are reverted to the default values corresponding to the new value of **NEAR**. If an LSE marker segment with **ID**=X'01' or **ID**=X'0C' follows the **LSE** marker segment with **ID**=X'06', these thresholds may be redefined. **RESET** and **MAXVAL** may also be changed by this segment. Next, the variables **RANGE**, **LIMIT**, **qbpp**, *A*[], *B*[], *C*[], *N*[], *Nn*[], and *RUNindex* are reset as if a restart marker was detected. If the extended prediction is used, the number of elements of *SPf*[0..**RANGE**], *SPm*[0..**RANGE**] are adjusted corresponding to the new value of **RANGE**, and *SPx*, *SPt*, *SPf*[], *SPm*[] are also reset.

NOTE – To apply this extended function to the arithmetic-based coding process, **NEARRUN** shall be ignored.

G.1.2.3 Visually oriented quantization specification

Figure G.4 specifies the visually oriented quantization threshold contained in the **LSE** marker segment when **ID** is equal to X'07'.

LSE	LI	X'07'	TQ

Figure G.4 – LSE marker segment for visually oriented quantization

The new parameter shown in Figure G.4 is defined below. The size and allowed values of each parameter are given in Table G.3.

TQ: Visually oriented quantization threshold.

Table G.3 – LSE marker segment parameter sizes and values for visually oriented quantization

Parameter	Size(bits)	Values
LL	16	4
ID	8	X'07'
TQ	8	$0 \leq$ **TQ** ≤ 13

G.1.2.4 Extended prediction specification

Figure G.5 specifies whether the extended prediction techniques, i.e., prediction value control and symbol packing, are used when **ID** is equal to X'08'.

LSE	LI	X'08'

Figure G.5 – LSE marker segment for extended prediction

Although this marker segment contains no additional parameter, appearance of the marker segment indicates that the extended prediction shall be applied. The size and allowed values of each parameter are given in Table G.4. Note that in the baseline coding process with the extended prediction, both of the prediction correction specified in D.2.2 and the symbol packing specified in D.2.3 are used, while in the arithmetic coding process with the extended prediction, only the prediction correction specified in A.4.2 is used.

Table G.4 – LSE marker segment parameter for extended prediction

Parameter	Size(bits)	Values
LI	16	3
ID	8	X'08'

G.1.2.5 Specification of the start of fixed length coding (floating marker)

Figure G.6 specifies the start of fixed length coding when **ID** is equal to X'09'.

LSE	LI	X'09'	NMCU

Figure G.6 – LSE marker segment to indicate the start of FLC