ISO/IEC JTC 1/SC29/WG1 N76037
76th Meeting, Turin, Italy, June 17-21, 2017

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG16)

Coding of Still Pictures

JBIG
Joint Bi-level Image Experts Group

JPEG
Joint Photographic Experts Group

TITLE: High Throughput JPEG 2000 (HTJ2K): Call for Proposals

SOURCE: WG1

PROJECT: ISO/IEC 15444 (JPEG 2000)

STATUS: Final

REQUESTED ACTION: Public Distribution

DISTRIBUTION: Public

Contact:
ISO/IEC JTC 1/SC 29/WG 1 Convener – Prof. Touradj Ebrahimi
EPFL/STI/IEL/GR-EB, Station 11, CH-1015 Lausanne, Switzerland
Tel: +41 21 693 2606, Fax: +41 21 693 7600, E-mail: Touradj.Ebrahimi@epfl.ch

ISO/IEC JTC 1/SC29/WG1 N76037
76th Meeting, Turin, Italy, June 17-21, 2017

High Throughput JPEG 2000: Call for Proposals

The JPEG Committee has launched the High Throughput JPEG 2000 (HTJ2K) activity, which aims to develop an alternate block coding algorithm that can be used in place of the existing block coding algorithm specified in ISO/IEC 15444-1 (JPEG 2000 Part 1). The objective is to increase throughput of JPEG 2000 while otherwise maintaining its unique combination of features, including minimizing the impact of changes on existing codestream syntax and structure, implementations, workflows and content libraries. The output of HTJ2K activity is intended to be published as Part 15 of the JPEG 2000 family of specifications (ISO/IEC 15444).

This document is a Call for Proposals (CfP) that invites proponents to submit technology contributions that fulfil the scope, objectives, requirements, and use cases herein. Registration of intent is due no later than October 1, 2017 as detailed at Section 2.4.

ISO/IEC JTC 1/SC29/WG1 N76037
76th Meeting, Turin, Italy, June 17-21, 2017

1 Background

1.1 Introduction

The success of JPEG 2000 (ISO/IEC 15444-1) across many high-performance applications (D-Cinema, video post-production, film preservation, medical, satellite, etc.) has been due to its unique combination of features.

A recurring issue in these high-performance applications has been the computational complexity of the JPEG 2000 block coding algorithm. The resulting negative impact on throughput can present an obstacle to:

- applications adopting increasing sampling rates (e.g. higher resolutions, or higher frames rates) and sample depths, especially at high-bit rates.
- implementation on power-constrained platforms.

It is therefore desirable to make incremental improvements to JPEG 2000 to increase its throughput, while maintaining its unique combination of features. This includes minimizing the impact of changes on existing codestream syntax and structure, implementations, workflows and content libraries. In particular, the JPEG 2000 block coding algorithm has a substantial negative impact on throughput, and thus a primary candidate for improvement.

1.2 Scope

The High Throughput JPEG 2000 (HTJ2K) activity aims to develop an alternate block coding algorithm that can be used in place of the existing block coding algorithm[1] specified in ISO/IEC 15444-1.
The alternate block coding algorithm is intended to:

- provide substantially higher throughput than the existing block coding algorithm on a given platform (CPU, GPU, FPGA, ASIC…) without substantially reducing compression performance. Software platforms are of primary interest, although others are also important.
- allow transcoding to/from codestreams generated by the existing block coding algorithm, such that the transcoding is mathematically lossless and computationally efficient.
- improve power efficiency such that higher throughput on a given platform should be equivalent to reduced energy consumption per sample.
- be used within all existing parts of the JPEG 2000 standard, with minimal syntactic and structural impact.

The output of HTJ2K activity is intended to be published as a part of the JPEG 2000 family of specifications (ISO/IEC 15444).

[1] The same block coding algorithm is currently used in all parts of the JPEG 2000 work item, except that some optional modifications are introduced by Part 2 (the ultra-fast mode is a very minor extension) and by Part 10 (JP3D allows for the coding of extended blocks).

1.3 Use Cases

1.3.1 General

Unless specified otherwise, HTJ2K is intended to enable the same use cases as those enabled by JPEG 2000 Part 1.

For instance, HTJ2K applies to the JPEG 2000 use cases involving resolution scalability and spatial region-of-interest accessibility. These use cases are particularly relevant to applications that involve some interactive access to a part of the compressed content.

The following summarizes use cases specifically enabled by HTJ2K, while Annex A discusses selected use cases in more detail.

1.3.2 Transcode from J2K to HTJ2K for Efficient Processing

In some applications, an image or image sequence that has already been encoded using JPEG 2000 Part 1 is transcoded to HTJ2K for storage and processing, benefiting from an increase in throughput. This transcoding can be performed on a separate platform or even on the rendering platform itself; the latter is valuable whenever content may need to be decoded or rendered multiple times, as often happens in interactive image communication and rendering applications based on the mechanisms and protocols defined by ISO/IEC 15444-9 (JPIP).

1.3.3 Transcode from HTJ2K to J2K for Legacy Interchange

In some applications, an image or an image sequence is encoded using HTJ2K for local storage and processing, but transcoded to JPEG 2000 Part 1 for interchange with legacy implementations. Such transcoding can be performed incrementally or on-demand.

1.3.4 Transcode from HTJ2K to J2K for Higher Coding Efficiency or Improved Scalability

Some applications may find it useful to transcode from HTJ2K to JPEG 2000 Part 1 for communication to a remote location, taking advantage of quality/SNR scalability and the high coding efficiency of the block coding algorithm specified in JPEG 2000 Part 1.

1.3.5 HTJ2K Native Applications

Some applications will use HTJ2K for both encoding and decoding, without any intermediate transcoding to/from the original JPEG 2000 block bit-stream representation.

1.4 High-level Technical Requirements

1.4.1 Codesteam Syntax

HTJ2K shall reuse the JPEG 2000 Part 1 codestream syntax, as specified in Annex A of JPEG 2000 Part 1, unless necessary to fulfil other requirements therein.

NOTE: The codestream syntax is expected to be modified to accommodate the changes to the decoding

process, as permitted in Section 1.4.2.

1.4.2 Minimal Changes to the J2K Decoding Process

Modifications to the JPEG 2000 Part 1 decoding process shall be limited to the Arithmetic Coding and Coefficient Bit Modelling processes specified in Annex C and D of JPEG 2000 Part 1, as illustrated in Figure 1.

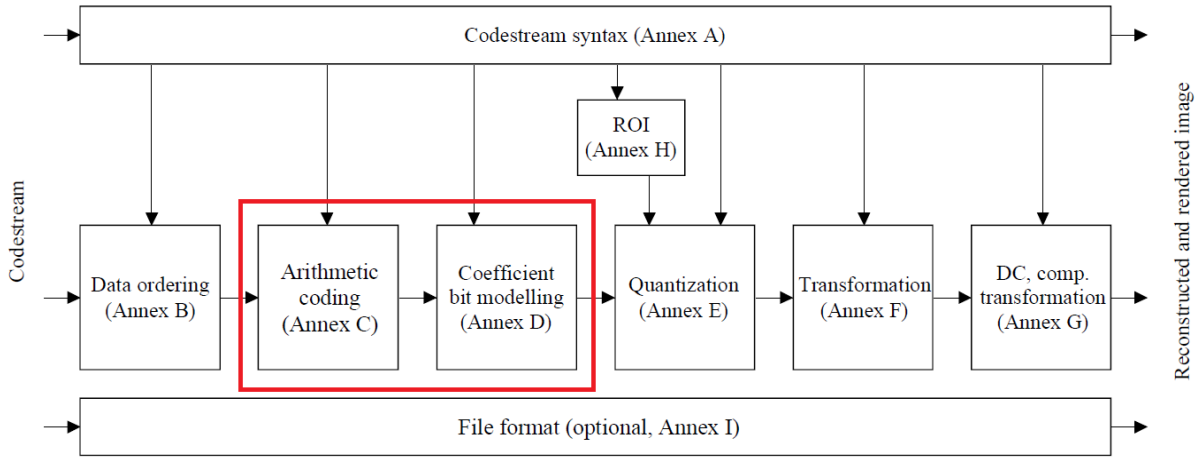

Figure 1. Minimal Changes to the J2K Decoding Process. The modifications introduced by HTJ2K are limited to the processes within the red box.

1.4.3 Mathematically Lossless Transcoding

The transcoding between codestreams produced by the existing block coding algorithm and those produced by the HTJ2K block coding algorithm shall be mathematically lossless.

1.4.4 Block-based Decoding

Decoding of one HTJ2K block shall not depend on the decoding of any other HTJ2K block.

NOTE: This enables applications in which processing is performed incrementally or on-demand.

1.4.5 Increase in Block Decoder Throughput

Over a range of bitrates and on a given software platform, the throughput of the HTJ2K block decoder should be on average no less than 10 times greater than the JPEG 2000 Part 1 block decoder of the reference specified in Annex D.

Increase of throughput of the HTJ2K block decoder is also desirable on hardware and GPU platforms.

1.4.6 Increase in Block Encoder Throughput

Over a range of bitrates, on a given software platform and with identical decoded image, the throughput of the HTJ2K block encoder should be on average no less than 10 times greater than the JPEG 2000 Part 1 block encoder of the reference specified in Annex D.

Increase of throughput of the HTJ2K block encoder is also desirable on hardware and GPU platforms.

NOTE 1: Achieving identical decoded images is possible since only the Arithmetic Coding and Coefficient Bit Modelling processes from JPEG 2000 Part 1 are modified by HTJ2K.

NOTE 2: JPEG 2000 Part 1 supports a wide range of encoder modalities. Like JPEG2000 Part 1, it is expected that systems incorporating HTJ2K block coder will support a similar range of modalities, and that the end-to-end throughput will vary for each such modality.

1.4.7 Coding Efficiency Target

On a given platform and with identical decoded image, the HTJ2K codestream should be on average no more than 15% larger than the corresponding JPEG 2000 Part 1 codestream.

1.5 Optional Features

1.5.1 Quality scalability

HTJ2K may preserve quality scalability features of JPEG 2000 Part 1, but is not required to do so.

NOTE: It is expected that the increase in throughput will result in a loss of coding efficiency and quality/SNR scalability, since the existing block coding algorithm obtains a very high degree of quality scalability through the use of multiple coding passes – a primary source of complexity. While preserving quality/SNR scalability is desirable, it is less so than achieving higher throughput and preserving coding efficiency. Applications requiring quality/SNR scalability can continue using JPEG 2000 Part 1 in its present form.

1.6 Royalty-Free Goal

The royalty-free patent licensing commitments made by contributors to JPEG 2000 Part 1 has arguably been instrumental to its success. JPEG expects that similar commitments would be helpful for the adoption of HTJ2K.

2 Call for Proposals

2.1 Introduction

This CfP invites proponents to submit technology contributions that fulfil the scope, objectives, requirements and use cases therein.

Proponents are reminded that they are expected to contribute to the standardisation process, as described in Section 2.6, and attend meetings and present their findings, as specified in Table 1.

2.2 Submission requirements

A submission shall consist of the elements specified in Annex B.

Elements of a submission are provided according to the timeline specified in Section 2.4.

2.3 Evaluation of proposals

The committee plans to select technologies to be included in the standard based on satisfying the requirements and evaluating the results obtained through the evaluation procedure documented in Annex C.

2.4 Workplan

Table 1 specifies the HTJ2K workplan, including submission deadlines. The technical committee intends to prioritize consideration of those submissions that meet these deadlines.

Table 10.HTJ2K Workplan.

27-31 March 2017	Meeting 75. Draft CfP ready for publication.
1 July 2017	Deadline for feedback on Draft CfP.
17-21 July 2017	Meeting 76. Review feedback from Draft CfP and develop Final CfP for publication.
1 September 2017	Test Bench source code, Anchor Block Code Library and Reference Block Coder Library made available.
October 1 2017	Registration of intent. Proponents that intend to provide a complete submission shall post element B.1 of their submission as input document to Meeting 77, and indicate their intent to provide a complete submission and contribute to the standardization process.
23-27 October 2017	Meeting 77. Review of registrations of intents. Proponents are required to present element B.1 of their submission. A teleconference bridge is expected to be available. Finalize set of Test Images.
5 January 2018	Proponents are required to post element B.2 of their submission.
January 2018, dates TBC	Meeting 78. Review proposed modifications to the Test Bench.
2 February 2018	Test Bench frozen
1 March 2018	No later than this date, proponents shall post elements B.1 through B.3 of their submission as an input document to Meeting 79.
April 2018	Meeting 79. Evaluation of submissions. Selection of one or more submissions as basis for standardization process. If selected, a proponent shall post elements B.4 through B.5 of their submission as an input document to Meeting 79. Proponents are required to attend this meeting in person.
End of Meeting 78 + 15 days	WD
October 2018	Meeting 80. CD
February 2019	Meeting 81. DIS
June 2019	Meeting 82. IS

The above schedule is subject to change, depending on the nature of proposals that are received and the possible need to integrate or merge elements from separate proposals.

2.5 IPR conditions (ISO/IEC Directives)

Proponents are advised that this call is being made in the framework and subject to the common patent policy of ITU-T/ITU-R/ISO/IEC and other established policies of these standardization organizations. The persons named below as contacts can assist potential submitters in identifying the relevant policy information.

2.6 Contribution to Standardization

Proponents are informed that based on the submitted proposals, a standard specification will be created. If they submit a proposal and (part of) the proposed technology is accepted for inclusion in the standard, they will be expected to attend subsequent WG1 meetings and contribute to the creation of the relevant documents. Within this process, evolution and changes are possible as several technologies may be combined to obtain a better performing solution.

2.7 Submission and materials

Materials submitted to, or made available as part of, the HTJ2K activity:

- shall be posted to the JPEG document repository[2]; and
- shall not be used for any purpose other than strictly necessary to develop the HTJ2K standard. Unsanctioned uses include competitive comparison, marketing, external publication, etc.

2.8 Further information

2.8.1 HTJ2K Ad hoc Group

The HTJ2K Ad Hoc Group was established at the 73rd JPEG meeting in October 2016, following the demonstration of a candidate technical solution. The HTJ2K Ad Hoc Group is charged with developing this CfP, collecting use cases and evidence for both requirements and technologies, and facilitating the process leading to the evaluation of proposals, as identified in the above timeline. All interested parties are requested to register to the email reflector of the HTJ2K Ad Hoc Group at:

https://listserv.uni-stuttgart.de/mailman/listinfo/jpeg-htj2k

2.8.2 Contacts

Touradj Ebrahimi (JPEG Convener)
Email: Touradj.Ebrahimi@epfl.ch

David Taubman (HTJ2K Ad hoc Group Chair)
Email: d.taubman@unsw.edu.au

Siegfried Foessel (HTJ2K Ad hoc Group Co-Chair)
Email: siegfried.foessel@iis.fraunhofer.de

[2] isotc.iso.org/livelink/livelink/open/jtc1SC29wg1

ISO/IEC JTC 1/SC29/WG1 N76037
76th Meeting, Turin, Italy, June 17-21, 2017

ANNEX A – DETAILED USE CASES

A.1. Encoding Use Cases

A.1.1. Lightweight compression on cinematography equipment

The ultimate target in cinematography is often JPEG 2000, but complexity constraints usually prevent JPEG 2000 compression being applied immediately during acquisition. It is desirable for the industry to converge on a common standard output format for cinematography cameras and auxiliary equipment. HTJ2K is a compelling candidate for this format, allowing both high throughput/low power encoding and seamless integration with workflows that involve JPEG 2000, for capture, editing, archival and ultimately distribution.

This use case requires compression of content ranging from full HD at 60fps through to 8K at 120fps, with an important target being 4K at 60fps. Input formats are typically 444 at 12 bits/sample.
Compression ratios of interest for this content are around 10:1. Specifically, bit-rates in the range 2bpp to 4bpp are of interest, but higher bit-rates up to 8bpp may also be required depending on the efficiency of the coding algorithm. Current on-camera encoding strategies involve quantization-based quality control, where quantizer settings are adjusted to ensure a sufficient level of quality, leading to a variable compressed size. For this application, both hardware (FPGA) and software (CPU/GPU) implementations are important.

A.1.2. Virtual Reality Cameras (including 360 degree cameras)

At least one virtual reality capture device already uses JPEG 2000. Low power and low complexity are important for this use case, especially considering that multiple cameras are employed by the acquisition system. At the same time, not all the captured content from a VR camera can generally be viewed at the same time. This implies that the encoded format for VR applications should support region-of-interest access, and perhaps other forms of scalability that are existing strengths of the JPEG 2000 standards.

In this use case, HTJ2K would be used to encode original content produced during image acquisition. Selective communication (e.g., via JPIP) and decoding of content of interest is enabled immediately by this approach, but an interactive browsing experience can be improved further by transcoding to the bit-stream format of JPEG 2000 Part 1. It is expected that this would result in somewhat higher coding efficiency and superior quality scalability, which are both important for highly responsive interactive access to image sources.

This application requires compression of multiple (e.g. 6-8) video streams, typically at full HD resolution. Target compressed bit-rates for this application are 1bpp to 4bpp. Some applications can be latency sensitive, in which case the end-to-end latency may need to be smaller than one frame period.

A.1.3. Compression of earth observation imagery, including hyperspectral imagery

Current commercial satellites are capable of collecting large quantities of imagery. For example, DigitalGlobe's WoldView-4 satellite sensor will be capable of acquiring imagery over 680,000 sqm per day, with permission recently granted to adopt resolutions down to 25cm. This amounts to a data rate of 126 Msamples/s per channel. Hyperspectral sensing at these resolutions will clearly be capable of

generating data well into the giga-sample/s range.

At the same time, satellite communication capabilities are limited. At the high end, satellites operating in the X band include those from Satellite Imaging Corporation (800 Mbit/s) and Google's Terra Bella satellites (400Mbit/s). Satellites operating in the S band experience much lower communication bandwidths, in the tens of Mbits/s.

Evidently, compression of satellite imagery is very important. For hyperspectral content, especially, one cannot expect to communicate all content that can be resolved to a base station. As a result, the ability to selectively store and communicate regions of interest and resolutions of interest is expected to be increasingly important. These are capabilities of JPEG 2000 Part 1 as it stands today, and indeed JPEG 2000 is used in some current satellites. However, satellites also have limited power available for performing the compression itself.

HTJ2K addresses the requirements of future satellite imaging applications by providing both a low complexity encoding path and the ability to transcode to the Part 1 JPEG 2000 bit-stream format on demand, on a block-by-block basis if desired. Content transcoded to JPEG 2000 Part 1 is expected to have superior quality scalability and coding efficiency, which can facilitate progressively degradable storage of content and efficient communication of content of interest to a base station.
Similar considerations apply to other platforms for earth observation.

A.1.4. Compression and Distribution of Astronomical Imagery

The planned Square Kilometre Array (SKA) will produce data products, some of which will be multi-dimensional imagery of significant size – multiple TBs up to 1 PB. Current telescopes such as VLA, ASKAP, Arecibo are already producing scientific imagery in similar regimes. The current implementation of JPEG 2000 and its JPIP has enabled the remote visualization to 0.5 and 1.1 TB spectral imaging cubes.

In a current workflow, 4K images with originally 32-bit floating-point intensity data are losslessly encoded and stacked into cubes of 256 components each. Such sub-cubes are generated naturally due to the parallelization of data reduction processing. The sub-cubes are further linked to form a 30,720 component super-cube that can be encapsulated within a single JPX file and remotely accessed via JPIP. The number of components is not limited to these current values and may indeed be required to be much larger. Very often some data has to be reprocessed and new sub-cubes have to be produced. Faster encoding and a lower computational footprint for the encoding will hugely benefit the SKA.

The ability to retrieve and manipulate any arbitrary part of volumetric imagery is crucial to scientific exploration of PB scale data sets. Natural use of multi-component encoding could be a good aid for this, as supported by Part 2 of JPEG 2000.

Another aspect is accurate preservation of the true data. Scientific exploration of astronomy data is done at the lowest signal-to-noise ratio. Some techniques involve aggregation of data to detect the signal from e.g. a radio galaxy. This requirement is often in contrast to the requirement to enable remote exploration. Some optimal solution is desired here.

While the nature of radio telescope data is linear, the visual exploration of the data is usually done in a

logarithmic domain. So, the ability to use non-linear transforms is certainly a benefit, as offered by Part 2 of JPEG 2000. Also, because telescopes are taking the images of the sky, non-liner mapping would ideally be defined as part of the compressed representation, as currently it can only be done at the client side with some significant computational cost.

The bandwidth optimization for remote data visualization in astronomy traditionally is achieved using a multi-resolution approach, but the multi-quality approach preserves spatial features in far higher compression regimes and is thus beneficial for studies requiring high spatial resolution.

The aggregated throughput required will be close to 3 Tera Voxels/s; however this won't be done using a single processor.

A.2. Decoding Use Cases

A.2.1. Efficient local rendering from a JPIP client cache

JPIP clients receiving content in JPEG 2000 form typically cache this content and render dynamically from their cache. In interactive applications, where the user pans and zooms across a potentially very large image (or video) surface, the same code-blocks tend to be decoded over and over again. While techniques to cache decoded content are possible, they are clumsy and very memory intensive. As a result it is preferable to render directly from the compressed data, retrieving the relevant code-block bit-streams and reconstructing the viewport of interest on the fly. This can be done in a highly responsive manner, for seemless interactive navigation, even on mobile devices. However, power consumption is non-negligible.

A preferable approach is for the cached code-block bit-streams to be transcoded to HTJ2K on the fly. In this use case, each time a code-block is accessed and decoded, it is transcoded to the HTJ2K format. The next time the code-block is accessed, it can be decoded with much higher throughput, and hence lower resource utilization.

The primary requirement in this case is for decoding with minimal energy consumption, while being able to represent the transcoded content economically. Since compressed content occupies space in a client cache, it is important that the transcoded representation for a code-block be not too much larger than the cached content received originally.

Coded original JPEG 2000 content could have a wide range of bit-rates, extending to lossless.

A.2.2. Medical image compression, including volumetric imagery

Within a hospital environment, networking infrastructure may render the quality scalability feature of JPEG 2000 less important. High throughput / low power compression and decompression have been identified as the most important requirements for this setting.

When the same content is communicated outside the hospital environment, however, coding efficiency and quality scalability both become much more important.

This dichotomy naturally suggests the benefits of a high throughput option to JPEG 2000 that supports reversible transcoding.

In this use case, content is initially compressed using JPEG 2000 with a quality scalable lossy to lossless representation. Content can then be dynamically transcoded to HTJ2K by server equipment so that browsing devices in the hospital environment can be as energy-efficient as possible. At the same time, the original JPEG 2000 content can be served to browsing clients located outside the hospital environment, allowing maximally responsive browsing in lower bandwidth contexts.

This application involves both images and medical volumes, which can be efficiently handled using JPEG 2000 transform extensions found in Part 2 and Part 10 (JP3D).

Lossless compression is likely to be important for this application.

A.2.3. Cost effective visualization of high resolution JPEG 2000 media

An Interoperable Master Format (IMF) conforming to SMPTE standards ST 2067-20 or ST 2067-21 represents a Home Video Master version of a Motion Picture. The SMPTE standards ST 2067-20 and ST 2067-21 require the use of either the Broadcast Profile or IMF Profiles of the JPEG 2000 Part 1 standard using either lossy or lossless compression modes. The content owner has a desire to review the IMF after a Mastering Facility creates it before distributing it to the content licensees worldwide.

On an IMF review workstation, the IMF is transcoded losslessly from JPEG 2000 Part 1 to HTJ2K and is stored on the review workstation's local storage system. The content owner staff "golden eye" reviews the content carefully when the transcoding operation has completed. The use of HTJ2K on the playback systems reduces the number of CPU devices needed.

A.2.4. Cost effective transcoding of high resolution JPEG 2000 media

An Interoperable Master Format (IMF) conforming to SMPTE standards ST 2067-20 or ST 2067-21 represents a Home Video Master version of a Motion Picture Film. The SMPTE standards ST 2067-20 and ST 2067-21 require the use of either the Broadcast Profile or IMF Profiles of the JPEG 2000 Part 1 standard using either lossy or lossless compression modes. A licensee receives an IMF from a content owner and stores the content using Cloud.

The licensee uses the Cloud Computing to losslessly transcode the IMF to HTJ2K. The licensee then decodes the HTJ2K content and encodes to H.264/AVC using 8 different encoding profiles at different resolutions and bitrates for an OTT video distribution service. One year passes and new devices become available that have VP9 decoding capability, and the content licensee decodes the HTJ2K content again and encodes to VP9 using 10 different encoding profiles at different resolutions and bitrates for the OTT video distribution service. The use of HTJ2K decoding instead of JPEG 2000 Part 1 decoding reduces the content licensee's Cloud Computing CPU time expense.

A.3. End-to-End Use Cases

A.3.1. Live performances and view finding

Similar to the above use case, this one involves live video, but the purpose and viewing locations are different. In this use case, video content is exchanged between the main stage and mixing/control locations, carried over the same infrastructure that is currently used for audio. Typically, audio is carried over Gigabit Ethernet and the available bandwidth for the carriage of video is between 200 Mbit/s and 800 Mbit/s. HD and 4K video resolutions at 30 or 60 Hz are of interest.

This application requires low power encoding and decoding, with the option to use either hardware FPGA platforms or software within modest CPU platforms. The specific combination of features that are particularly relevant to HTJ2K here are real-time software processing and relatively low compressed bit-rates.

A.3.2. Telepresence Systems

This use case can include large displays and image capture configurations, such as display walls. Current display wall resolutions can range into the hundreds of Mega pixels. Some applications for display walls involve local rendering of data sets for visualization. Other applications, however, involve data sharing and interactive communication between sites. For the latter set of applications, low (sub-frame) latency high throughput encoding and decoding tools are required. Moreover, in interactive display applications, it is expected that some parties in the interaction will connect via conventional display systems, which are capable of displaying only a portion of the overall content.

Features of the use case include very high spatial resolutions, the need for low (sub-frame) latency compression, and the need for efficient communication and decoding of windows of interest into the content. These features are all current strengths of JPEG 2000 Part 1, especially when coupled with JPIP (JPEG 2000 Part 9), but it is desirable to reduce the processing load, especially for encoding or decoding of the full content at display-wall resolutions.

For this use case, acceptable bit-rates are expected to lie in the range 0.5bpp to 2bpp, but interactive communication via large display walls is a new area that has not yet been widely explored.

ISO/IEC JTC 1/SC29/WG1 N76037
76th Meeting, Turin, Italy, June 17-21, 2017

ANNEX B – SUBMISSION REQUIREMENTS

B.1. High-level description and analysis

The submission shall include:

- A high-level technical description of the submission including block diagrams of the block encoder and block decoder.
- An indication that the submission meets all requirements.

Proponents that believe that their proposal is substantially affected by the Test Bench API as defined in Annex C.4 are invited to submit a description of their concerns as well as a proposal that resolves their concerns.

B.2. Modifications to the Test Bench

The submission may include modifications to the Test Bench (see Annex C.4) in the form of a patch against its source code.

B.3. Executables

The submission shall include a Block Coder Library as specified in Annex C.

The Block Coder Library shall:

- target the x86-64 instruction set
- be a shared library built for the platform specified at Annex C.5.
- not require external libraries other than glibc/glibc++
- not modify the thread affinity or create its own threads
- not store quantized sample values between calls
- be named libhtj2k.so

Executable compression or similar tools may be used to prevent reverse engineering or disassembly of the submitted executable.

NOTE: While the approach used to evaluate the submissions removes the need for proponents to provide complete codecs, proponents who have developed complete codecs are nonetheless invited to submit additional executables and/or performance measurements of potential interest to the committee.

B.4. Algorithm and design description

B.4.1. General

The submission shall include a detailed description of the proposed algorithm, including:

- a theoretical discussion on the mechanism used by proposed algorithm to satisfy requirements of Section 1.4; and

- any specific CPU instruction sets from which the implementation would benefit, including vector processing instructions.

B.4.2. Beyond CPU implementations

An analysis of GPU implementations may be provided and, if so, focus on available parallelism. More concrete evidence of GPU implementation complexity is also of interest.

An analysis of hardware implementations may be provided. A minimal analysis should consider memory, use of complex logic units such as multipliers and barrel shifters, as well as identifying the likely critical path dependencies in the implementation of the algorithm. Additional evidence of hardware implementation complexity is also of interest.

B.4.3. Encoder Modalities

The submission may describe how the proposed algorithm supports various encoder modalities, such as:

- constrained codestream size
- constrained codestream distortion

B.5. Technical documentation

A complete technical description of the proposal, sufficient to enable standardization, shall be provided, including:

- overview of operations;
- codestream syntax;
- lossless transcoding process to and from JPEG 2000 Part 1; and
- decoding process

The description shall include all necessary information to create a bit-exact codestream.

B.6. Verification model source code

The submission shall include source code to serve as a verification model, written in a high-level language, such as C or C++.

The source code shall implement the Block Coder Library inteface specified in Annex C.

Assembly language or GPU code shall not be included.

Source code shall be documented and understandable.

The source code does not need to correspond to the executable programs of Annex B.2.

All libraries used by the source code shall be available in source code form with ISO/IEC and ITU-T compliant terms.

The codestream output of the source code shall match exactly the ones obtained using the submitted executable files.

ANNEX C– EVALUATION PROCEDURE

C.1. Overview

C.1.1. Overview

This Annex defines a procedure for evaluating submitted HTJ2K block coder algorithms relative to the JPEG 2000 Part 1 block coder.

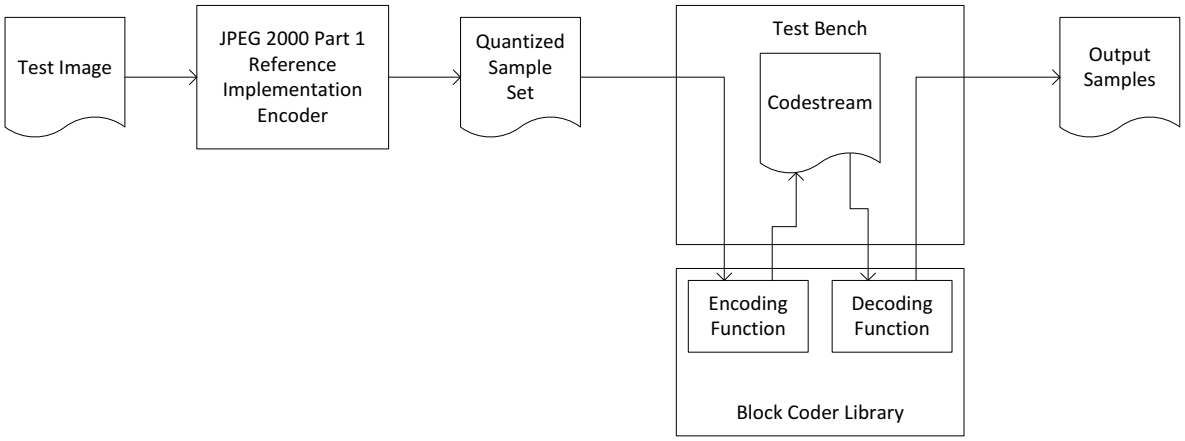

Figure 2. Evaluation Procedure.

As illustrated in Figure 2, the procedure is built around a software executable (the Test Bench described in Annex C.4), whose source code is available to all proponents. The Test Bench is responsible for measuring the performance of a Block Coder Library, which is a shared library that implements the encoding and decoding function of the algorithm under evaluation. The Test Bench communicates with the Block Coder Library using a well-defined API.

The Test Bench operates (see Annex C.6) by first providing the encoding function of the Block Coder Library with a Quantized Sample Set, which consists of quantized samples resulting from the encoding of a Test Image (Annex C.2) by the JPEG 2000 Reference (Annex D). The encoding function generates a codestream, which the Test Bench validates for compatibility with JPEG 2000 codestream syntax. The Test Bench then provides the codestream back to the decoding function of the Block Coder Library, which outputs quantized samples. The Test Bench records the time taken by the encoding and decoding steps as well as the size of the codestream. These measurements are then compared to those obtained using the JPEG 2000 Part 1 block coder, as described in Annex C.7.

NOTE: This approach is possible since HTJ2K is limited to replacing JPEG 2000 Part 1 block coder, and submitted HTJ2K block coder algorithms therefore accept the same quantized samples that would have been provided to the JPEG 2000 Part 1 block coder.

Each Block Coder Library is evaluated in two contexts:

- The standalone context covers end-to-end use of HTJ2K block coder. In this context, the Quantized Sample Set is generated by decoding JPEG 2000 Part 1 codestreams such that no coding pass

truncation is performed.
- The transcoding context covers the lossless transcoding of any valid JPEG 2000 Part codestreams. In this context, the Quantized Sample Set is generated by decoding JPEG 2000 Part 1 codestreams without restrictions on coding pass truncation. The Test Bench makes available the information about where each code-block has been truncated to the encoding function of the Block Coder Library.

In both contexts the quantized samples output from the Block Coder Library are expected to be identical to the Quantized Sample Set. Although these two contexts describe quite different use cases, the same combination of software Test Bench and proponent library can be used in both contexts.

C.2. Test Images

C.2.1. General

Proponents are invited to use the points of contact at Section 2.8.2 to retrieve the Test Images.

The set of Test Images include the original suite of test images used in the development of JPEG 2000 Part 1, supplemented with:
- the"Waltham1" and "Waltham2" images contributed[3] by Andy Walter from IBM during the development of JPIP;
- images from the High Density Camera Array (HDCA) data set;
- frames from the JPEG XS test content.

The set of Test Images may be revised as an outcome of Meeting 77.

C.2.2. Images

AERIAL2_2048x2048_8b_Y.pgm
BIKE_2048x2560_8b_RGB.ppm
CAFE_2048x2560_8b_RGB.ppm
CATS_3072x2048_8b_Y.pgm
COMPOUND2_5120x6624_8b_Y.pgm
FINGER_512x512_8b_Y.pgm
GOLD_720x576_8b_Y.pgm
HOTEL_720x576_8b_Y.pgm
MAT_1528x1146_8b_Y.pgm
SEISMIC_512x512_8b_Y.pgm
TEXTURE1_1024x1024_8b_Y.pgm
TEXTURE2_1024x1024_8b_Y.pgm
TOOLS_1520x1200_8b_RGB.ppm
ULTRASOUND_512x448_8b_Y.pgm
WALTHAM1_3600x2600_8b_RGB.tif
WALTHAM2_3800x2600_8b_RGB.tif

WATER_1465x1999_8b_Y.pgm
WOMAN_2048x2560_8b_RGB.ppm
XRAY_2048x1680_12b_Y.pgm
HDCA_set2_0000_0000.ppm
HDCA_set6_0000_0000.ppm
HDCA_set9_0000_0000.ppm
HDCA_set10_0000_0000.ppm
FemaleStripedHorseFly_1920x1080_8b.ppm
HintergrundMusik_1920x1080_8b.ppm
EBU_PendulusWide_3840x2160p_50_10b_bt709_444_0001.ppm
HUAWEI_ScMap_1280x720p_60_8b_sRGB_444_000.ppm
APPLE_BasketBallScreen_2560x1440p_60_8b_sRGB_444_000.ppm
ARRI_AlexaDrums_3840x2160p_24_12b_P3_444_00000.ppm
ARRI_AlexaDrums_3840x2160p_24_12b_logC_444_00000.ppm
ARRI_AlexaHelicopterView_3840x2160p_24_12b_P3_444_00000.ppm
ARRI_AlexaHelicopterView_3840x2160p_24_12b_logC_444_00000.ppm
ARRI_Lake2_2880x1620p_24_8b_bt709_444_0000.ppm
ARRI_PublicUniversity_2880x2160p_24_8b_bt709_444_0000.ppm
BLENDER_Sintel1_4096x1744p_24_8b_sRGB_444_00003096.ppm
BLENDER_Sintel2_4096x1744p_24_8b_sRGB_444_00004606.ppm
BLENDER_TearsOfSteel_4096x1714p_24_12b_sRGB_444_01290.ppm
RICTHER_ScreenContent_4096x2160p_15_8b_sRGB_444_0001.ppm
VQEG_CrowdRun_3840x2160p_50_8b_bt709_444_07111.ppm
VQEG_ParkJoy_3840x2160p_50_8b_bt709_444_15523.ppm
noise_3840x2160_12b.ppm

C.2.3. Sub-sampled Images

ARRI_AlexaDrums_3840x2160p_24_12b_bt709_422_00000.yuv
ARRI_AlexaHelicopterView_3840x2160p_24_12b_P3_422_00000.yuv
ARRI_Lake2_2880x1620p_24_10b_bt709_422_0000.yuv
ARRI_PublicUniversity_2880x2160p_24_10b_bt709_422_0000.yuv
EBU_PendulusWide_3840x2160p_50_10b_bt709_422_0001.yuv
VQEG_CrowdRun_3840x2160p_50_10b_bt709_422_7111.yuv
VQEG_ParkJoy_3840x2160p_50_10b_bt709_422_15523.yuv

C.3. Quantized Sample Sets

A Quantized Sample Set is generated by the following sequence of steps for each combination of (i) coding parameters specified in Table 3, (ii) Test Image specified in Annex C.2 and (iii) standalone and transcoding contexts:

- a codestream is generated by encoding the Test Image using the Reference Implementation (Annex

D.3) according to the combination of coding parameters, with no code-block truncation occurring in the standalone context; and
- the codestream is decoded to yield quantized samples, which form the Quantized Sample Set.

NOTE: To limit the implementation burden of the algorithm submitted for evaluation, the Quantized Sample Sets only exercise a subset of the block coder configurations supported by JPEG 2000 Part 1 described in Annex D.2. Each submitted algorithm is however expected to support lossless transcoding for all such configurations, including vertically causal context. This is demonstrated through the analysis provided in Annex B.4.

C.4. Test Bench

C.4.1. Overview

The Test Bench is an executable that runs single-threaded and configures the CPU affinity to run on exactly one logical CPU.

The Test Bench uses the following two interface functions to call the Block Coder Library:

```
htj2k_encode(int num_blocks, htj2k_blk block_inf[],
             int samples32[], unsigned char bytes[],
             size_t max_bytes_in, size_t *bytes_out)

htj2k_decode(int num_blocks, htj2k_blk block_inf[],
             int samples32[], unsigned char bytes[])
```

NOTE: Quantization and de-quantization lie outside the block coder. In fact, since de-quantization is not normatively defined by JPEG 2000 Part 1, it is important to define the interpretation of the quantized samples returned by `htj2k_decode()`. The Test Bench adopts that used by the original JPEG 2000 Verification Model software.

Each `htj2k_blk` structure in the `block_inf` array supplied to the interface functions is defined as follows:

```
struct htj2k_blk {
  // Read-only fields first
  int width, height; // block dimensions
  int stride; // Separation between successive block rows
  size_t sample_offset; // location in `samples32' array
  int Kmax; // num magnitude bit-planes -- sub-band specific

  // Fields that might be written by proponent binaries
  int missing_msbs; // read-write
  int passes; // read-write
  int ht_pass_lengths[88];
};
```

The following Sections define the interface functions and the fields of the `htj2k_blk` structure. Some of these fields remain constant across interface function calls, while other are set by the Block Coder Library. The Test Bench returns an error if the `htj2k_blk` structure returned by the Block Coder Library does not conform to the constraints specified herein.

C.4.2. htj2k_encode()

The `htj2k_encode()` function encodes quantized samples into a bitstream.

By default, `htj2k_inf::stride == htj2k_inf::width`, i.e. the sub-band samples appear in packed raster order within each block.

NOTE: The Test Bench source code is made available to all proponents. Proponents can modify the Test Bench, as provided at Annex B.2, to add extra in-memory representation of the quantized samples, so that these alternate representations can be provided to the proponent's Block Coder Library.

`htj2k_inf::sample_offset` provides an offset from the start of the `sample32` array that is passed to `htj2k_encode()` or `htj2k_decode()` to the top-left sample in the code-block. This value is not modified by the Block Coder Library.

A 32-bit representation is used for quantized sub-band samples. The latter are stored in sign-magnitude form, with the sign in bit 31, followed by the most significant available magnitude bit-plane in bit 30, and so forth, so that the least significant bit of the quantized magnitude for a sample is found in bit `(31 - Kmax)`. The `Kmax` value here is the maximum number of magnitude bit-planes that can possibly be available for any code-block of a given sub-band; it is recorded in the `htj2k_inf::Kmax` field. This quantity depends upon information found in the QCD/QCC marker segments of a JPEG 2000 codestream, including the number of guard bits that have been assigned.

It usually happens that some of the most significant magnitude bit-planes (from bit 30 down) are entirely 0. In calls to `htj2k_encode()`, the value of the `htj2k_inf::missing_msbs` field enters with the pre-computed number of most significant bit-planes that are known to be entirely 0. The `htj2k_encode()` call may change these values if it likes, relying upon the fact that the modified values will be (at least notionally) recorded in JPEG 2000 packet headers and passed to the `htj2k_decode()` function.

Any change to `htj2k_inf::missing_msbs` leaves values that can legally be represented via the existing JPEG 2000 packet header syntax.

A key consideration in transcoding tests is the amount of effective quantization that is introduced to the sample values by coding only a limited set of the available magnitude bit-planes. During encoding, the `htj2k_inf::passes` field identifies the effective number of JPEG 2000 coding passes whose information is communicated by the encoding process.

In the standalone context, `htj2k_inf::passes` is just set to its maximum value of `3*(Kmax-missing_msbs)-2`. In transcoding tests, it may be significantly smaller. The Block Coder Library may modify the value of the `htj2k_inf::passes` value to reflect the number of coding passes that it is producing – this value effectively gets recorded in the JPEG 2000 packet headers, along with the pass lengths, and is provided later to the decoding function.

The Block Coder Library may modify the `htj2k_inf::passes` field, so long as it is modified to a legal value. Moreover, the Block Coder Library shall provide length information (number of bytes) for each of the identified passes, via the `htj2k_inf::ht_pass_lengths` array. The first `htj2k_inf::passes` entries

of this array shall contain valid lengths.

The Block Coder Library shall write the bytes for each of its coding passes directly to the `bytes` array that is passed to the `htj2k_encode()` function, concatenating coding passes from each block, without any intervening space. The total number of written bytes is returned via the `bytes_out` argument to `htj2k_encode()`, while the maximum number of bytes that can be written to the `bytes` array is supplied via the `max_bytes_in` argument. If `max_bytes_in` turns out not to be large enough, the call to `htj2k_encode()` shall return prematurely, with `bytes_out` set to a value larger than `max_bytes_in`. If this happens, the entire Test Bench will generate a warning but then increase the size of the `bytes` array and try the whole process again.

The Block Coder Library does not need to implement coding passes. The only requirement is that the value stored in `htj2k_inf::passes` remains a legal number of coding passes from the perspective of the JPEG 2000 packet header syntax, and that the associated pass lengths contain all coded data generated by the encoding function. The Test Bench effectively records the number of passes within the relevant packet headers, storing the coded data in the corresponding packet bodies.

The coded bytes returned by the Block Coder Library conform to the following constraints required by the JPEG 2000 codestream syntax:

- no pair of bytes should form a code in the range `0xFF90` to `0xFFFF`
- no identified coding pass should end with `0xFF`.

For simplicity, bit-1 of the SPcod/SPcoc field within COD/COC markers is assumed to be set, meaning that the block coder is assumed to terminate each coding pass. This means that the pass lengths recorded in the `ht_pass_lengths` array will be preserved by JPEG 2000 packet headers and passed to the decoding function. A Block Coder Library is not required to use this functionality.

C.4.3. `htj2k_decode()`

During decoding, quantized sub-band samples should be returned by the Block Coder Library. The decoding function is supplied with the same `htj2k_inf::passes` and `htj2k_inf::missing_msbs` values that were returned by `htj2k_encode()`. `htj2k_decode()` may leave these fields unchanged, since they are not used by the Test Bench after decoding.

The `htj2k_decode()` function uses the `htj2k_inf::ht_pass_lengths` array to discover the number of coded bytes it needs to retrieve from the `bytes` array that it is passed. The code bytes for each block are tightly packed within the `bytes` array. The `bytes` array passed to `htj2k_encode()` might not be the same as that passed to `htj2k_decode()` but the referenced byte values (as determined by the pass lengths) will be identical.

The decoding function is, however, required to signal the number of magnitude bit-planes that were actually decoded for each sample by setting the bit position immediately below the least significant decoded bit for a sample. Specifically, writing X[**n**] or the magnitude part of a quantized sample returned by the decoding function, either X[**n**] is 0, meaning that the decoded sample value is 0, or else the decoded magnitude bits are to be found in bit positions h[**n**]+1 through 30, where h[**n**] is the least significant non-zero bit position in X[**n**]. That is, X[**n**] is equal to either 0 or (2M[**n**]+1)<<h[**n**], where M[**n**] holds the

decoded bit-planes, with the least significant decoded magnitude bit for the sample at location **n** found in the LSB of M[**n**].

NOTE: This strategy for encoding the number of decoded magnitude bits for each sample is the same one that was adopted in the JPEG 2000 Verification Model software. It has the desirable property that the returned values X[**n**] already incorporate the half-LSB offset of the mid-point reconstruction policy that is employed by many dequantizers. Other dequantizer strategies can be employed, since the dequantization procedure is not normative in JPEG 2000 Part-1, but the signalling convention is required to ensure integrity of the decoding or transcoding process in both the standalone and transcoding contexts.

The Test Bench verifies that the quantized samples produced by `htj2k_decode()` are identical to those originally passed to the `htj2k_encode()`.

C.5. Measurement Platform

The Measurement Platform is specified in Table 2.

Table 2. Measurement Platform.

Mainboard	HP EliteDesk 800 G2 Base Model Small Form Factor PC (L1G76AV)
CPU	Intel i7-6700 @ 3.4GHz, 8MB L3 Cache
RAM	2 x 8GB Dual Channel DDR4 2133 MHz (Slot 1 and 3)
Operating System	64bit Ubuntu 16.04-LTS
Hyperthreading	Disabled (at the BIOS level)

CPU clock frequency	Fixed (at the OS level) using the following commands:
	`sudo cpupower frequency-set -g performance`
	`sudo cpupower frequency-set -u 3400MHz -d 3400MHz`

C.6. Measurements

Given a Block Coder Library, the Test Bench performs on the Measurement Platform specified in Annex C.5 the following sequence of calls twice without interruption for each Quantized Sample Set:

- `htj2k_encode()`
- `htj2k_decode()`

For each cycle, the following data is recorded:

- the time taken for each of the API calls, as reported by the `clock()` POSIX function.
- the number of coded bytes produced by the encoding function, i.e. the value `bytes_out` returned by `htj2k_encode()`

NOTE: The encode-decode cycle is performed twice to confirm that the Test Bench does not introduce significant overhead in the execution of the library. This also allows “cache warmup” effects to be observed.

C.7. Evaluation

Each submitted Block Coder Library as well as the Anchor Block Coder Library is evaluated using the following metrics:

- bitrate change relative to the Reference Block Coder Library (vertical axis) against corresponding Reference Block Coder Library bits/pixel (horizontal axis), allowing the coding efficiency to be compared with the existing JPEG 2000 Part 1 block coder; and
- throughput increase (vertical axis) against corresponding Reference Block Coder Library bits/pixel (horizontal axis), allowing throughput to be compared with the existing JPEG 2000 Part 1 block coder, as a function of bitrate.

C.8. Reference Block Coder Library

The Reference Block Coder Library consists of a Block Coder Library that uses the block coder of the Reference Implementation of Annex D using the default code block style.

NOTE: The Reference Block Coder Library corresponds to the following configuration parameters for the Reference Implementation: `Cmodes=0`.

C.9. Anchor Block Coder Library

The Anchor Block Coder Library uses the block coder of the Reference Implementation of Annex D configured with the following modes selective arithmetic coding bypass mode used in such a way as to bypass arithmetic coding for all non-empty significance propagation and magnitude refinement passes

NOTE 1: The reset mode and the termination mode of the JPEG 2000 Part 1 block coder are not included in the anchor because these modes benefit from coarse-grained parallelization, i.e. multi-threaded or multi-core, which is not included in the evaluation – the evaluation uses a single thread on a single core. If these modes were included in the anchor, the coding efficiency would be degraded with no benefit to throughput on the evaluation platform.

NOTE 2: Item (a) above corresponds to fastest mode in ISO/IEC 15444-2 AMD4, but using JPEG 2000 Part 1 syntax.

NOTE 3: The Anchor Block Coder Library corresponds to the following configuration parameters for the Reference Implementation: `Cmodes=\{BYPASS\|BYPASS_E1\|BYPASS_E2\|CAUSAL\}`.

NOTE 4: Vertically causal context is not included in the anchor because it would not satisfy transcodability requirements in the context of the Test Bench.

ANNEX D– JPEG 2000 REFERENCE

D.1. Implementation

The Reference Implementation consists of the Kakadu SDK Version 7.10 (http://www.kakadusoftware.com).

D.2. Common Configurations

Table 3 specifies common coding parameters used with JPEG 2000 Part 1.

Table 3. JPEG 2000 Common Coding Parameters.

Code-block size (width x height)	32x32, 64x64, 1024x4	
Wavelet Filters	9-7 (lossy)	5-3 (lossless)
Decomposition levels	5	
Decorrelating multi-component transform	enabled	
Compressed bit-rate	0.5bpp, 1.0bpp, 2.0bpp, 4.0bpp and 6.0bpp	lossless

Visual weights	For images:
	`Cband_weights:C0=0.090100,0.275800,0.275800,0.701800,0.837800,0.837800,1.000000`
	`Cband_weights:C1=0.026300,0.086300,0.086300,0.136200,0.256400,0.256400,0.334600,0.469100,0.469100,0.544400,0.652300,0.652300,0.707800,0.779700,0.779700,1.000000`
	`Cband_weights:C2=0.077300,0.183500,0.183500,0.259800,0.413000,0.413000,0.504000,0.646400,0.646400,0.722000,0.825400,0.825400,0.876900,0.942400,0.942400,1.000000`
	For sub-sampled images:
	`Cband_weights:C0=0.090078,0.275783,0.701837,0.837755,0.999988,0.999994`
	`Cband_weights:C1=0.027441,0.089950,0.141965,0.267216,0.348719,0.488887,0.567414,0.679829,0.737656,0.812612`
	`Cband_weights:C2=0.070185,0.166647,0.236030,0.375136,0.457826,0.587213,0.655884,0.749805,0.796593,0.856065`	

D.3. Test Image Encoding

D.3.1. General

The command lines of this section illustrate the use of the Reference Implementation.

In the case of sub-sampled images, `kdu_compress` is replaced by `kdu_v_compress`.

D.3.2. Transcoding Context

D.3.2.1. Non-reversible transforms

```
kdu_compress -i <image> -o out.j2c Qstep=0.001 Cblk=\{64,64\} Corder=RPCL -
precise -full -tolerance 0 -no_info -rate <R>

kdu_compress -i <image> -o out.j2c Qstep=0.001 Cblk=\{32,32\} Corder=RPCL -
precise -full -tolerance 0 -no_info -rate <R>

kdu_compress -i <image> -o out.j2c Qstep=0.001 Cblk=\{4,1024\} Corder=RPCL -
precise -full -tolerance 0 -no_info -rate <R>
```

D.3.2.2. Reversible transforms

```
kdu_compress -i <image> -o out.j2c Cblk=\{64,64\} Corder=RPCL Creversible=yes -
full -no_info -rate <R>

kdu_compress -i <image> -o out.j2c Cblk=\{32,32\} Corder=RPCL Creversible=yes -
full -no_info -rate <R>

kdu_compress -i <image> -o out.j2c Cblk=\{4,1024\} Corder=RPCL Creversible=yes -
full -no_info -rate <R>
```

D.3.3. Standalone Context

D.3.3.1. Lossy

```
kdu_compress -i <image> -o out.j2c Qstep=<Q> Cblk=\{64,64\} Corder=RPCL -no_info

kdu_compress -i <image> -o out.j2c Qstep=<Q> Cblk=\{32,32\} Corder=RPCL -no_info

kdu_compress -i <image> -o out.j2c Qstep=<Q> Cblk=\{4,1024\} Corder=RPCL -
no_info
```

where <Q> is modulated by the script to hit the desired target bit-rate.

D.3.3.2. Lossless

```
kdu_compress -i <image> -o out.j2c  Cblk=\{64,64\} Corder=RPCL Creversible=yes -
no_info

kdu_compress -i <image> -o out.j2c  Cblk=\{32,32\} Corder=RPCL Creversible=yes -
no_info

kdu_compress -i <image> -o out.j2c  Cblk=\{4,1024\} Corder=RPCL Creversible=yes
-no_info
```