THE OPEN-SOURCE TURING CODEC: TOWARDS FAST, FLEXIBLE AND PARALLEL HEVC ENCODING

S. G. Blasi[1], M. Naccari[1], R. Weerakkody[1], J. Funnell[2] and M. Mrak[1]

[1]BBC, Research and Development Department, UK
[2]Parabola Research, UK

ABSTRACT

The Turing codec is an open-source software codec compliant with the HEVC standard and specifically designed for speed, flexibility, parallelisation and high coding efficiency. The Turing codec was designed starting from a completely novel backbone to comply with the Main and Main10 profiles of HEVC, and has many desirable features for practical codecs such as very low memory consumption, advanced parallelisation schemes and fast encoding algorithms. This paper presents a technical description of the Turing codec as well as a comparison of its performance with other similar encoders. The codec is capable of cutting the encoding complexity by an average 87% with respect to the HEVC reference implementation for an average coding penalty of 11% higher rates in compression efficiency at the same peak-signal-noise-ratio level.

INTRODUCTION

The ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) combined their expertise to form the Joint Collaborative Team on Video Coding (JCT-VC), and finalised the first version of the H.265/High Efficiency Video Coding (HEVC) standard (1) in January 2013. HEVC provides the same perceived video quality as its predecessor H.264/Advanced Video Coding (AVC) at considerably lower bitrates (the MPEG final verification tests report average bitrate reductions of up to 60% (2) when coding Ultra High Definition, UHD, content). HEVC specifies larger block sizes for prediction and transform, an additional in-loop filter, and new coding modes for intra- and inter-prediction. This variety of options makes HEVC encoders potentially extremely complex. Therefore, in order to achieve low complexity requirements and improved coding efficiency, practical HEVC encoders should be carefully designed with tools to speed up the encoding as well as architectures that allow parallel processing, reduced memory consumption, and scale according to the available computational resources.

This paper introduces the Turing codec[1], an open-source software HEVC encoder. The design of the codec has been mainly driven by the distribution of UHD content in 8 and 10 bits per component, 4:2:0 format and up to 60 frames per second (fps). As such, particular attention was devoted to design the codec architecture so that the memory footprint is reduced, and many tools were developed specifically for the codec to reduce the complexity of highly demanding encoding stages. This paper continues with an overview of

[1] http://turingcodec.org

the related background associated with open source HEVC encoders. The focus will then move to describe the main features, tools and architecture of the codec. After this description an evaluation of the compression performance and complexity will be performed and compared with other software HEVC open source codecs.

RELATED BACKGROUND

Many efforts are being spent towards the development of efficient HEVC software encoders. During the definition of HEVC, JCT-VC developed an open-source reference software encoder and decoder under the name of HEVC test Model (HM) (3). The HM encoder includes almost all the tools considered in HEVC, with the objective of providing a reference to be used during the development, implementation and testing of new coding algorithms and methods. An HEVC encoder needs to select the optimal partitioning of a Coding Tree Unit (CTU) in Coding Units (CUs), the best prediction mode for each CU, and the optimal inter- or intra-prediction for each Prediction Unit (PU) (1). The HM reference software performs most of these decisions by means of brute-force searches where many options are tested, and the optimal configuration is selected in a Rate-Distortion (RD) sense. As such, HM is optimised for efficiency, and requires too high computational complexity to be considered in practical scenarios. Nonetheless, HM is mentioned in this paper due to its completeness as an upper-bound benchmark in terms of efficiency.

In 2013, the x265 encoder project was launched with the goal of developing a fast and efficient HEVC software encoder, reproducing the successful development model of the x264 AVC encoder. The codec was developed using the HM software as a starting point, where the code functionalities and structure were heavily improved to increase performance and allow for various tools and enhancements. x265 includes many options desirable for a practical encoder implementation, such as parallel encoding, speed profiles, spatiotemporal prediction optimisations, etc. A complete description of the encoder functionalities is out of the scope of this paper, but many details can be found on the project official website (5).

A few other similar projects for HEVC open source software codecs are available, at various stages of maturity. Kvazaar (6) is an academic project with the main goal of providing a flexible modular structure to simplify the data flow modelling and processing, while at the same time supporting some parallelisation and optimisation tools. At the time of writing, Kvazaar supports Main and Main10 profiles and provides support for the majority of HEVC tools. Finally, the libde265 project (7) is currently being developed mainly as an HEVC decoder, distributed under the LGPL3 licence.

OVERVIEW OF THE TURING CODEC

The backbone of the Turing codec were developed following a parallel approach: on one side, the code foundations have been developed making full use of state-of-the-art C++11 constructs and assembly optimisations; on the other side, advanced algorithms were designed to improve demanding encoding processes. The final goal was that of providing a fast HEVC software encoder suitable for a variety of applications. The codec is at an advanced maturity stage, while it is also under active development to implement additional functionalities and further improve the codec performance. Recently it has been made available as open source. The codec was named after Alan Turing, one of the most influential scientists in the development of the foundations of theoretical computer science.

Main Features

The Turing codec is compliant with the Main and Main10 profiles of HEVC (mainly defined for content distribution). A partial list of the functionalities currently supported is as follows:

- Encoding of slice types I, P and B;
- Fixed Group of Picture (GOP) structures, with GOP length of 8 or 1;
- Configurable intra-refresh period;
- All CU sizes and PU types specified in HEVC;
- All 35 directions for intra-coding, with or without strong intra smoothing filtering;
- Inter-coding with uni- or bi-prediction from List 0 (L0) or List 1 (L1);
- Rate Distortion Optimised Quantisation (RDOQ);
- Deblocking filter;
- Rate control;
- Support for field (interlaced) coding;
- Shot change detection.

Alongside these functionalities the codec offers specific features designed to provide the encoder with high coding efficiency and low computational complexity.

Encoding Process Optimisations and Speed Presets

The flexibility provided by HEVC with its large number of coding modes is responsible for the high computational complexity associated with the standard. Encoders must select the optimal configuration for each image area. Performing all decisions with an exhaustive RD search is clearly not ideal. For this reason, during the development of the Turing codec, great importance was given towards evaluating the impact of HEVC tools on compression efficiency and complexity, with the goal of defining a set of requirements (in terms of parameters and tools) to guide the development of the encoder. Many experiments were performed for this purpose, as detailed in (8). In particular the HM reference software was used (Version 12.0) to encode 16 UHD sequences (spatial resolution of 3840 × 2160 luma samples), frame rate of 50 or 60 Hz, 4:2:0 format, 8 bits per component, according to the Common Test Conditions (CTCs) (9) used by the JCT-VC under the Random Access Main (RA-Main) configuration. Four Quantisation Parameters (QP) per sequence were selected to uniformly span quality across the test-set. Compression efficiency and encoder complexity were then measured. For compression efficiency, the Bjøntegaard Delta-rate (BD-rate) was used (10), where negative BD-rate values correspond to compression gains. For complexity, the encoding running time was used, where average time reduction across all QPs for a sequence is considered as the Encoder Speedup (ES).

Table 1 presents a summary of the results of some of these experiments. Avoiding the use of large CTU sizes leads to high BD-rate penalties, while limiting the Residual Quad-Tree

Table 1 - List of HEVC tools: coding efficiency and complexity

Tool	BD-rate [%]	ES [%]
CTU size 32 × 32	7.0	15.0
RQT depth 1	1.0	15.0
One reference frame	2.5	31.0
Sub-pel ME off	8.5	39.0
AMP off	0.5	5.0

(RQT) depth to 1 provides good ES for limited compression efficiency losses. Similarly, limiting the encoder to considering one inter-prediction reference frame from each list provides acceptable losses for consistent speedups. Limiting the Motion Vector (MV) precision to integer precision resulted in high efficiency losses, whereas disabling AMPs resulted in negligible effects. These experiments resulted in a set of requirements used throughout the development of the Turing codec. For example a maximum RQT depth of 1 is considered, only one reference frame is used, and AMPs are never tested.

The experiments were also used as a basis in the development of the various fast algorithms implemented in the codec. The Adaptive Partition Selection (APS) algorithm (8) analyses the motion activity in a portion of the sequence to determine whether to test the symmetric 2N × N and N × 2N modes (corresponding to splitting the CU into two rectangular PUs in the horizontal and vertical direction, respectively). Motion activity is computed based on the homogeneity of the residuals resulting from testing the 2N × 2N mode: if residuals are highly homogeneous, testing of symmetric modes is avoided, hence reducing complexity. The Multiple Early Termination (MET) algorithm stops integer-precision ME by analysing the residuals in the surrounding of the motion search starting points. In case an initial ME pattern search returns one of the multiple starting points as optimal MV, no other MVs are tested, else the starting point is updated and conventional ME is performed. The Reverse CU (RCU) algorithm reduces the set of depths to test on a CTU by either avoiding testing CUs at minimum or maximum depths. The depth selected in neighbouring blocks is used at this purpose to predict the maximum depth likely to be used in the current CTU and consequently to limit the range of depths to test on the CTU. Finally, the Fast Decision for All Modes (FDAM) algorithm analyses the residuals found after quantisation when testing each inter-prediction mode. If all residuals are zero, then transform and quantisation are avoided while testing all subsequent inter-prediction modes, which means the prediction block is used as final reconstruction.

Currently, the Turing codec supports three speed presets: *slow*, *medium* and *fast*. The presets control the pre-defined values of options and tools and enable/disable the above algorithms, to provide flexibility between computational complexity and encoder efficiency. A detailed list of the effects of the three speed presets is presented in Table 2.

Memory Usage, Parallel Processing Optimisations and Programming Optimisations

The Turing codec benefits from several performance optimisations that leverage parallel CPU features, reduce memory usage bandwidth and complexity, and minimise branch instructions in critical parts. The codec uses a thread pool with a controlled maximum number of active threads. By ensuring there are never more active threads than logical CPU cores, the likelihood of pre-emption and associated cost of context switch is greatly reduced. The thread pool is task-based with every encoding activity: RD searches, reconstruction, in-loop filtering, bitstream manipulation, etc. Tasks are nodes in an implicit directed dependency graph, and an efficient paradigm was designed to manage this in a generic fashion, extensible beyond a single CPU and proven to operate well when distributed between asynchronous nodes collaborating to encode a single sequence.

Current practical software codecs make use of SIMD instructions via assembly code or intrinsics in C/C++ functions. The Turing codec is no exception but uses a Just-In-Time (JIT) assembler to assemble optimised functions at runtime. This approach has two main

Table 2 - Turing codec speed presets

Tool (options)	Speed preset		
	Slow	Medium	Fast
AMPs	disabled	disabled	disabled
2N × N, N × 2N	enabled	enabled	disabled
RDOQ	enabled	enabled	disabled
RQT	enabled	disabled	disabled
APS	disabled	enabled	enabled
MET	disable	enable	enable
RCU	disabled	enabled	enabled
Sub-pel ME	all	all	only half-pel
Search range for ME	64	64	32
Search range for bi-prediction	5	5	1
Number of merge candidates	5	5	2
Number of intra modes to test with RD search	8	4	4
Sign Data Hiding	enable	enable	disable
Deblocking filter	enable	enable	enable
Minimum CU size	8	8	8
CTU size	64	64	64

benefits. First, it allows developers to use C++ templates, loops, conditions and function calls instead of ad-hoc assembler macros. Second, JIT assembly allows run-time parameters such as bit depth and raster stride to be embedded into the machine code, reducing the need for additional parameters and stack manipulation.

Moreover, the Turing codec introduces a novel approach to the management of temporary data in the form of the *snake* data structure. Video codecs have to make a large number of RD decisions, and in doing so intermediate variables are generated, discarded, recreated or copied for each alternative. The *snake* memory is a one-dimensional memory capable of representing two-dimensional data. It is employed in several modules where neighbouring sample data, modes or coefficients are required. While conventional codecs require metadata memory size of order *width* × *height*, the *snake* has a characteristic size of only *width* + *height*. This results in faster fill operations, better cache utilisation and lower memory bandwidth. Finally, every decision selected by the codec (modes, flags, MVs and coefficients) is stored in a serial format which can be easily cut, copied and pasted in memory, and is very convenient for fast output of the bitstream.

EVALUATION OF THE TURING CODEC

Different experiments were performed to assess the performance of the Turing codec under a variety of conditions. The results reported in this section refer to the codec run in single-thread mode, as this allows the algorithmic complexity associated with the codec workflow to be fully appreciated. First, the codec was evaluated with respect to its coding efficiency and computational complexity. Tests were performed using Main and Main10 profiles. In the former case, the same test set and conditions in Table 1 were used: sixteen UHD sequences were tested with four QP values. In these experiments, BD-rates and ES are reported as performance indicators, computed with respect to an anchor consisting of an HM encoder used to encode the same content under the JCT-VC CTCs (9).

Results of the experiments associated with the Main profile are shown in Table 3. The codec was tested with the three available speed presets and compared with an x265 encoder (Version 1.9). Three presets were used with x265, namely *placebo*, *medium* and *fast.* As can be seen, the Turing and x265 codecs are differently optimised.

Table 3 – Results of the Turing codec and x265 against HM under the Main profile

	Turing						*x265*					
	Slow		**Medium**		**Fast**		*Placebo*		*Medium*		*Fast*	
	BD-R [%]	**ES [%]**	**BD-R [%]**	**ES [%]**	**BD-R [%]**	**ES [%]**	*BD-R [%]*	*ES [%]*	*BD-R [%]*	*ES [%]*	*BD-R [%]*	*ES [%]*
ParkAndBuildings	13.2	36.9	19.8	88.5	35.3	95.9	16.2	71.9	20.6	86.0	44.8	98.7
NingyoPompoms	0.2	42.3	3.8	84.4	17.8	94.0	8.0	52.1	9.5	82.5	40.3	98.8
ShowDrummer1	10.2	35.1	14.4	86.0	36.0	95.3	11.8	63.6	15.2	85.5	60.3	98.8
Sedof	6.3	32.3	21.5	86.1	46.3	95.5	14.8	65.0	18.0	82.5	53.9	98.7
Petitbato	3.0	33.1	11.8	86.2	25.1	95.6	1.1	68.1	3.0	84.9	31.7	98.7
Manege	8.6	30.3	24.5	84.5	39.5	94.9	16.2	62.7	20.1	82.4	53.3	98.7
ParkDancers	3.9	32.7	5.2	88.0	17.7	95.8	11.5	77.0	12.8	89.2	34.4	98.9
CandleSmoke	5.3	37.4	9.2	89.0	24.6	96.0	16.9	74.4	19.6	88.2	46.2	98.9
TableCar	4.7	27.5	8.2	87.8	21.6	96.2	8.7	81.0	10.3	89.9	30.7	99.1
TapeBlackRed	4.8	37.1	9.6	89.2	18.6	95.6	9.0	73.9	12.9	88.7	54.0	98.9
Hurdles	6.5	34.7	16.8	88.3	40.2	95.9	12.0	72.9	16.6	86.7	59.8	98.8
LongJump	-0.7	37.0	8.4	86.6	28.6	95.5	2.9	64.1	16.2	83.8	61.8	98.7
Discus	-8.1	42.7	-3.6	87.8	14.5	95.7	-3.3	67.0	18.6	83.9	48.3	98.8
Somersault	5.2	37.0	11.6	89.6	25.6	96.3	11.8	78.3	18.3	89.3	72.5	99.0
Boxing	3.7	41.4	9.9	87.0	24.1	95.5	7.5	65.2	12.1	86.1	45.4	98.8
Netball	5.1	37.7	11.5	86.4	25.2	95.4	12.2	67.5	15.2	85.9	48.5	98.8
Average	4.5	35.96	11.4	87.22	27.5	95.57	9.8	69.05	14.9	85.97	49.1	98.83

The performance of the Turing codec is generally weighted towards achieving higher compression efficiency, whereas x265 tends to be more optimised towards lower computational complexity. When using the *slow* preset, the Turing codec is very close to the performance of HM (4.5% BD-rate losses), with an ES of 35.96%. Conversely, x265 with the *placebo* preset (i.e. when the codec can achieve the highest compression efficiency) results in higher 9.8% BD-rate loss, for 69.05% ES. The *medium* preset of both codecs can be taken as a fair term of comparison, as both codecs seem to target a similar trade-off between complexity and efficiency when using this preset. In this case, the Turing codec outperforms x265 both in terms of speed and coding efficiency, resulting in 11.4% BD-rate loss for 87.22% ES. Conversely, x265 results in a higher loss (14.9% BD-rate) for lower ES (85.97%). Finally, when using the *fast* preset, the performance of x265 degrades to 49.1% BD-rate loss, for very high speedups of 98.83%. The Turing codec instead favours quality, returning lower efficiency loss (27.5% BD-rates), for lower ES of 95.57%.

When testing the codecs under the Main10 profile, similar results are obtained as shown in Table 4. For this experiment, six high dynamic range sequences from the test material described in (12) were used. The sequences are in HD resolution (1920 × 1080 luma samples) with frame rates from 24 to 50 Hz, 4:2:0 format, 10 bits per component and compressed with the Hybrid Lo-Gamma (HLG) opto-electronic transfer function (13). The current version of the Turing codec only supports the *medium* and *fast* presets for the Main10 profile, which were both used for these tests. Again, results are weighted towards maintaining high compression efficiency. When using the *medium* preset 17.5% BD-rate loss is obtained for 73.67% ES compared to the HM reference encoder. Limited losses of 33.4% are obtained when using the *fast* preset, for 88.01% ES. Conversely, x265 already results in higher losses with the *placebo* preset (20.8% on average) for 76.73% ES, and up to 50.5% BD-rate loss with the *fast* preset for 98.61% ES.

The three encoders (HM, x265 and the Turing codec) were also compared in terms of the memory consumption utilised during the encoding in single thread mode. Accordingly, the first 25 frames of the *TapeBlackRed* UHD sequence were encoded with QP equal to 26.

Table 4 - Results of the Turing codec and x265 against HM under the Main10 profile

	Turing				*x265*					
	Medium		**Fast**		*Placebo*		*Medium*		*Fast*	
	BD-R [%]	**ES [%]**	**BD-R [%]**	**ES [%]**	*BD-R [%]*	*ES [%]*	*BD-R [%]*	*ES [%]*	*BD-R [%]*	*ES [%]*
BalloonFestival	**23.9**	**72.6**	**44.3**	**88.7**	*13.0*	*77.7*	*39.0*	*98.2*	*47.0*	*98.6*
FireEater2	**16.0**	**70.0**	**24.7**	**86.4**	*31.0*	*75.1*	*65.1*	*98.3*	*73.5*	*98.8*
GarageExit	**16.8**	**69.5**	**34.9**	**85.9**	*19.0*	*65.2*	*35.8*	*98.0*	*50.0*	*98.5*
MagicHourCut1	**12.7**	**74.1**	**20.5**	**87.6**	*22.9*	*79.6*	*39.2*	*98.1*	*45.1*	*98.6*
Market3	**20.3**	**77.2**	**40.6**	**89.9**	*16.2*	*77.1*	*34.8*	*98.2*	*39.0*	*98.7*
Sunrise	**15.3**	**78.7**	**35.4**	**89.7**	*22.9*	*85.7*	*42.2*	*98.1*	*48.6*	*98.5*
Average	**17.5**	**73.67**	**33.4**	**88.01**	*20.8*	*76.73*	*42.7*	*98.16*	*50.5*	*98.61*

A Linux machine using an Intel Xeon CPU E5-2680 at 2.50GHz was used for this experiment. The memory usage was sampled during the encoding at fixed intervals of one second, where the *slow* preset was used for the Turing codec, the *placebo* preset was used for x265, and the JCT-VC CTCs were used for HM. Results are shown in Figure 1. As can be seen, the Turing codec requires a fraction of the memory needed by the other two encoders. The maximum amounts of memory recorded were 247 MBytes for the Turing codec, 1260 for x265 and 2557 for HM. The jump shown for the HM encoder after the first GOP of 8 frames is due to the codec buffering at that stage all 8 frames in the second GOP, while also maintaining frames in the first GOP as reference frames. This does not happen in subsequent GOPs because the buffer is freed of unnecessary frames at time instants distant in the past. x265 and the Turing codec are more efficient in reducing memory requirements by only keeping necessary information in the buffers.

Finally, the codec was also used to encode five sequences in HD format representative of broadcasting applications. The compressed content was visually inspected by expert viewers and compared with the output of a practical AVC encoder at bitrates in line with the statistical multiplex averages. Side-by-side comparisons were performed at the viewing distance of three times the display height in standard viewing conditions. At the same perceived quality, the rate savings of the Turing codec were in the range of 15-40%.

CONCLUSIONS

This paper presented an overview of the Turing codec, focusing on a technical description of the main features. Algorithmic optimisations as well as programming performance

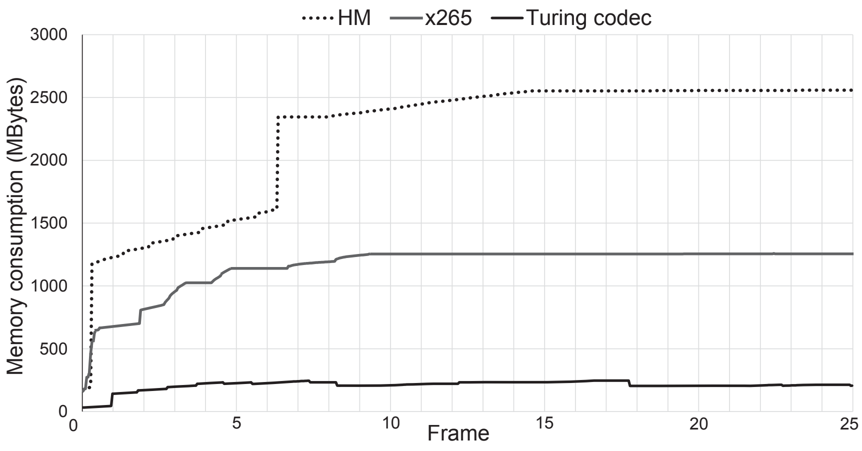

Figure 1 - Comparison of memory consumption of Turing codec, x265 and HM

optimisations are included in the codec, as detailed in the paper. A comprehensive performance assessment shows that the Turing codec achieves consistent results across a variety of content, providing high coding efficiency for relatively low complexity requirements. The codec is also capable of drastically lowering memory requirements with respect to similar projects, and it results in comparable subjective qualities for lower bitrates than those typically required by practical codecs used in the broadcasting chain.

ACKNOWLEDGEMENTS

The authors would like to thank InnovateUK for co-funding this work as part of the THIRA project. Part of this work has been conducted within the project COGNITUS, which has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 687605.

REFERENCES

1. Sullivan G. J., Ohm J.-R., Han W.-J. and Wiegand T., 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology. Vol. 22, pp. 1649 to 1668.
2. Tan T. K., Weerakkody R., Mrak M., Ramzan N., Baroncini V., Ohm J.-R. and Sullivan G. J., 2016. Video quality evaluation methodology and verification testing of HEVC compression performance. IEEE Transactions on Circuits and Systems for Video Technology. Vol. 26, pp. 76 to 90.
3. JCT-VC. HM Reference Software. Available: http://hevc.hhi.fraunhofer.de/HM-doc/.
4. Weerakkody R. and Mrak M., 2013. High Efficiency Video Coding for Ultra High Definition Television. Proceedings of NEM Summit, October 2013.
5. x265 Project Homepage. Available: http://x265.org/.
6. Kvazaar Project Homepage. Available: http://ultravideo.cs.tut.fi/#encoder.
7. Libde265 Project Homepage. Available: http://www.libde265.org/.
8. Naccari M., Gabriellini A., Mrak M., Blasi S. G., Zupancic I. and Izquierdo E., 2015. HEVC Coding Optimisation for Ultra High Definition Television Services. In Picture Coding Symposium, May 2015, pp. 1 to 5.
9. Bossen F., 2013. Common HM Test Conditions and Software Reference Configurations. JCTVC-L1100, 12th meeting, Geneva, January 2013.
10. Bjøntegaard G., 2001. Calculation of Average PSNR Differences Between RD-curves. VCEG-M33, 13th meeting, Austin, TX, April 2001.
11. Rosewarne C., Bross B., Naccari M., Sharman K. and Sullivan G. J., 2016. High Efficiency Video Coding (HEVC) Test Model 16 (HM 16) Update 5 of Encoder Description. JCTVC-W1002, 23rd meeting, San Diego, February 2016.
12. François E., Yin P. and Sole J., 2015. Common Test Conditions for HDR/WCG Video Coding Experiments. MPEG- N15793, 113th meeting, Geneva, October 2015.
13. Borer T. and Cotton A., 2015. A “Display Independent” High Dynamic Range Television System. Proceedings of the International Broadcasting Convention, September 2015.