Image and Video Coding Techniques for Ultra-low Latency

JAKUB ŽÁDNÍK, MARKKU MÄKITALO, JARNO VANNE, and PEKKA JÄÄSKELÄINEN,
Tampere University, Finland

The next generation of wireless networks fosters the adoption of latency-critical applications such as XR, connected industry, or autonomous driving. This survey gathers implementation aspects of different image and video coding schemes and discusses their tradeoffs. Standardized video coding technologies such as HEVC or VVC provide a high compression ratio, but their enormous complexity sets the scene for alternative approaches like still image, mezzanine, or texture compression in scenarios with tight resource or latency constraints. Regardless of the coding scheme, we found inter-device memory transfers and the lack of sub-frame coding as limitations of current full-system and software-programmable implementations.

CCS Concepts: • **Computing methodologies** → **Image compression**; • **Computer systems organization** → *Real-time system architecture*; • **General and reference** → **Surveys and overviews**; **Evaluation**; **Performance;**

Additional Key Words and Phrases: Video coding, texture compression, low latency, real-time video system

ACM Reference format:
Jakub Žádník, Markku Mäkitalo, Jarno Vanne, and Pekka Jääskeläinen. 2022. Image and Video Coding Techniques for Ultra-low Latency. *ACM Comput. Surv.* 54, 11s, Article 231 (September 2022), 35 pages.
https://doi.org/10.1145/3512342

1 INTRODUCTION

Since the advent of digital video, the need for saving network bandwidth and storage space has driven the effort for efficient compression. Over the years, the network speed has been increasing together with the processing power, but so did the video parameters—frame rate, resolution, or bit depth—putting more requirements on the video compression systems. To this end, **Motion Picture Experts Group (MPEG)** and **International Telecommunication Union Telecommunication Standardization Sector (ITU-T)** have introduced a steady line of video coding standards starting with H.261 [116] in 1988 through the nowadays ubiquitous **Advanced Video Coding (AVC/H.264)** [151] and **High-efficiency Video Coding (HEVC/H.265)** [132] up to the newest **Versatile Video Coding (VVC/H/266)** [20]. Each standard generation aims to approximately double the compression ratio for the same subjective and objective visual quality. However, the computation complexity also increases along with these improvements.

This work was financially supported by the Tampere University Graduate School and is part of the FitOptiVis project [4] funded by the ECSEL Joint Undertaking under Grant No. 783162. It also received funding from the European Union's Horizon 2020 research and innovation programme under grant number 871738 (CPSoSAware).
Authors' address: J. Žádník, M. Mäkitalo, J. Vanne, and P. Jääskeläinen, Tampere University, Korkeakoulunkatu 7, 33720 Tampere, Finland; emails: {jakub.zadnik, markku.makitalo, jarno.vanne, pekka.jaaskelainen}@tuni.fi.

0360-0300/2022/09-ART231
https://doi.org/10.1145/3512342

Solely improving the coding efficiency is not the only motivation for compression. Because often the energy cost of computation is much lower than the energy required for transmission, a lightweight compression scheme can achieve energy savings [14]. Battery life improvements are especially important in the context of multi-access edge computing [135]. In addition, even with the compression computation overhead, the decreased time spent on transferring the compressed data might result in a lower total time than transporting raw data without compression. In these types of scenarios, the tradeoff between computational complexity and achieved compression ratio must be carefully considered.

For non-interactive video systems, such as video playback, the coding[1] latency is not as important as the quality or file size—as long as the coding speed is fast enough to process the video at a required frame rate. Indeed, modern video codecs can exploit redundancies between frames, which requires frame buffering and an additional delay when buffering the future frames. However, interactive video applications require low latency to be usable. As shown later in the survey, with the current generation of wireless networks, the network transfer time is usually the main factor contributing to the total latency in real-world applications. Even with multi-gigabit Ethernet connections, it can be challenging to achieve sufficiently low latencies. However, recent advances in network technologies, such as WiFi-6 or 5G, promise to lower the transmission latency down to units of milliseconds. With such low latencies, the video coding pipeline becomes the bottleneck.

Faster networks open up space for new applications, such as cloud gaming, extended reality (XR), or internet of skills [37]. However, high latency can either hinder the user experience (cloud gaming) or contribute to motion sickness (virtual reality (VR)). While human-centered applications are tolerant to a degree of latency limited by perception, some automation systems such as robotized factories or autonomous vehicles rely on machine vision to control their actions. In the case of fast-moving and safety-critical applications, the latency requirements can be much stricter. Therefore, it is necessary to consider the coding latency also from the perspective of applications where humans with their imperfect brains are not involved, but the video stream is used to make ultra-low latency control decisions with robots.

Another trend motivating low latency video coding is "edge computing," or more specifically, the slight reversal of this trend: Instead of integrating more computing power to all sensors ("smart cameras") in cyber-physical systems, there is now a move back towards more centralized computing architectures, with processing done in a powerful computer connected with a fast network link to "dumber cameras" [125]. In response to the trend, AUTOSAR, the global leading automotive standardization partnership, has released an Adaptive Platform [49] with the focus on high-performance computing needed by intelligent cars, as an alternative to their previous classic decentralized architecture. Many car features, especially automated and autonomous driving, rely on fast video processing, where compression would play an important role in reducing the bandwidth and the latency between the sensors and the central server.

In general, video coding latency can be reduced by faster algorithms and their intelligent mapping (implementation) to the underlying compute platforms. Since the maximum clock frequencies do not scale anymore according to Moore's law, more processing power is achieved by parallelization, and so "faster" nowadays usually translates to "more parallel." The computational perspective shift requires reformulating traditional approaches to suit parallel execution models, which optimally should be visible at algorithm-level design decisions.

[1]In this survey, the word "coding" is used as a general term for both "encoding" and "decoding," since both parts are equally important in a full video system.

1.1 Contributions and Research Questions

To our best knowledge, this is the first comprehensive overview of video[2] coding techniques that (a) focuses primarily on the low latency aspect and (b) considers a wide range of alternative techniques, apart from the most common ones. The goal of this survey is also to settle the state-of-the-art video coding implementations into the context of full video systems with the upcoming networking trends in mind.

For categorization, we first define a precise limit for *low* and *ultra-low* latencies based on human perception limits and requirements of the upcoming networking standards. Then, by analyzing the state-of-the-art implementations of video coding schemes, we answer the following research questions:

(1) What are the main approaches to decreasing latency of video coding schemes and their implementations?
(2) What are the limitations of the current state-of-the-art video coding schemes in terms of latency?
(3) What are the computational requirements of the state-of-the-art video coding schemes?

1.2 Survey Outline

The rest of the survey is organized as follows. Section 2 formulates video system requirements from the perspective of low latency—we define how low should the latency be in different applications, and how it projects to a required coding speed and compression gain. Section 3 provides a general overview of image and video compression and categorizes existing approaches into four categories. Sections 4–6 contain the results of the survey, providing details about the findings in each of the categories. Section 7 summarizes our findings and proposes possible future research directions.

The main source of literature was the Scopus database, where we searched for each category introduced in Section 3 separately with a focus on real-time and low-latency keywords. The search results were sorted in reverse chronological order to prioritize the most current results. Papers reporting latency (i.e., not just the coding time) and a coding throughput of more than 50 MP per second were included with a priority. Furthermore, papers found outside this procedure were also included, when appropriate. A snowballing approach [153] was used in Section 4 with the help of the `pybliometrics` library [119] to discover possible related works within citations and references of included papers.

2 REQUIREMENTS OF A LOW-LATENCY VIDEO SYSTEM

Let us define a video system as the end-to-end chain that source video frames have to pass through until they are consumed by a target device. For this survey, the system's latency is the most important requirement. Hand-in-hand with low latency goes high throughput, since the system must be able to process a certain amount of data in a fixed time frame. The compression ratio is limited by the transport channel and directly affects the transmission speed. Therefore, whenever possible, we try to assess also the compression ratio. Visual quality, while certainly one of the key parameters of video compression, is hard to compare objectively. For this reason, we do not concentrate much on visual quality, unless Bjøntegaard metric [17] evaluation is provided, which estimates either the peak signal-to-noise ratio (PSNR) change for the same bitrate or the bitrate change for the same PSNR.

Figure 1 shows a generic diagram of a video system. Every arrow in the diagram represents a data transfer and therefore a possible source of latency. Furthermore, every device performs some

[2]We consider "video coding" in a broader sense as "image and video" coding.

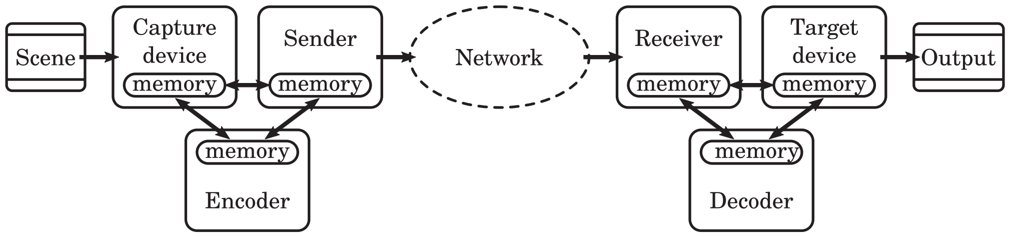

Fig. 1. Top-level view of a networked video system.

computation and internal memory transfers, again increasing the latency. In practice, devices can be merged. An example video coding system can look as follows: graphics processing unit (GPU) captures and encodes rendered frames from a running game. The encoded video stream has to be transferred to the central processing unit (CPU) to be sent via an operating system network stack. Receiving CPU then sends the encoded data into a hardware decoder integrated within a GPU from where the decoded video is transferred to an head-mounted device (HMD). In this example, the capture and encoding device is the same GPU while on the decoding side, the data has to pass three different devices.

2.1 Setting Latency Constraints

In interactive video systems, the "top-level" latency, also called *motion-to-photon latency*, is the time elapsed between a user's input on the source device and the observation on the display device. In machine control systems where the consumer is an algorithm and not a human, this term refers to the latency from the source video acquisition until the point when the transcoded signal is delivered to the input of the vision system. Even though the network is often a major latency contributor, the scope of the survey is limited to the encoder and decoder only.

How low the latency needs to be depends on the application. Kadowaki et al. [66] studied the users' performance solving a simple pointing task with their hand being projected at different *motion-to-photon latencies*. The authors concluded that if the latency is greater than 24.3 ms, the user's performance starts to decrease. Two articles by Lago and Kon [80] and Mäki-Patola and Hämäläinen [92] suggested that an audio latency below 20–30 ms is either imperceptible or acceptable for humans. Even though the articles refer to sound latencies, video is often accompanied by audio, and therefore, audio latency requirements have to be met as well.

Mochizuki et al. [98] face a restriction on the latency of a car surround view system to be under 100 ms, otherwise, the safety of the driver during parking could be violated. As the authors state, 100 ms corresponds to the migration distance of approximately 30 cm at the speed of 10 km/h. While 100 ms is sufficient for a parking system, to maintain the same travel distance of 30 cm at a speed of 100 km/h, the required latency would have to be 10 ms. From the example, it is apparent that low latency is critical for systems like autonomous driving where the vehicle moves fast. For example, a white paper by Ixia [63] defined end-to-end automotive Ethernet latency requirements as 0.25–1 ms for driver assistance and driver safety applications and <10 ms for human-machine interface systems.

A specification by the 3rd generation partnership project (3GPP) [1] defined use cases and requirements for 5G communication in automation. The specification refers to "low latency" as a latency lower than 10 ms and an "end-to-end" latency as a network end-to-end latency—different from the *motion-to-photon latency* defined for a complete video system. Among other things, latency requirements are mentioned for multiple use cases such as augmented reality (AR) (50 ms *motion-to-photon* to prevent motion sickness) or "mobile robots" (cycle time <10 ms for machine control, 10–100 ms for video-operated remote control).

Based on the above observations, we define *low latency* as a *motion-to-photon latency* between 20–50 ms and *ultra-low latency* as a *motion-to-photon latency* lower than 20 ms. For non-critical systems with a human in the loop, *low latency* ensures a high-quality user experience. *Ultra-low latency* sets a requirement for latency-critical systems where a precision higher than the human perception threshold is required to prevent damage or injuries.

The typical latency of current wireless technologies (e.g., 4G and 802.11ac, a.k.a. Wi-Fi 5) is in the order of tens of milliseconds, which by itself is too high for achieving *ultra-low latency*. The newest and upcoming wireless network standards (5G and 802.11ax, a.k.a. Wi-Fi 6) promise to reduce the latency towards units of milliseconds, making them possible to be used in latency-critical applications. For example, the ultra-reliable and low-latency communication (URLLC) paradigm used for the upcoming 5G standard requires 1 ms end-to-end network latency [16].

2.2 Throughput Versus Latency

High coding throughput (i.e., number of pixels encoder/decoder can process per unit of time) is a necessary condition to achieving low latency, although by itself it may not be sufficient. A processing latency of one frame might be too high to achieve *ultra-low latency* even at high frame rates—for example, a coding speed of 120 frames per second (FPS) corresponds to 1,000/120 = 8.3 ms coding time. Therefore, the ability of sub-frame coding would allow for achieving lower latencies without the necessity of significantly increasing the coding throughput, which is a very challenging task.

Theoretically, it should be possible to reduce the latency indefinitely, assuming the algorithm can support it. For example, mezzanine compression algorithms are capable of reaching latencies down to below one line of pixels [35]. In practice, however, sub-frame coding has its limits. Even if the algorithm and its implementation allow for sub-frame coding, the integration into a complete system might require buffering between the system pipeline stages, thus lowering the low-latency potential. Furthermore, the control overhead of managing multiple sub-frame streams might be too large if the sub-frame size is too small. Also, for highly parallel devices, most notably GPUs, the workload of encoding a small sub-frame can be too low to fully utilize its computational resources. Therefore, in some scenarios, optimizing for the lowest possible latency can paradoxically damage the coding throughput.

As the last point, data transfers between devices such as CPUs and GPUs are a known source of latency as well as a throughput bottleneck. Thus, spreading the computation across devices must be considered carefully.

2.3 Compression Ratio

The channel bandwidth plays an important role in determining the necessary compression ratio. Table 1 shows the minimum necessary theoretical compression ratios required to transport video (assuming 60 FPS without subsampling) over different digital video interfaces and network channels. The examples are given for 1080p, 4K (3,840 × 2,160) and 8K (7,680 × 4,320) resolutions with 10 bits per color component (assuming three components per pixel). The channel throughput values listed are theoretical maximum values not reflecting the channel overhead, and therefore, the required compression ratios should be perceived as slightly underestimated.

3 GENERAL COMPRESSION TECHNIQUES

We divided the possible video compression algorithms into four categories based on their design goals and coding principles: still image compression, video compression, mezzanine compression, and texture compression. Furthermore, we briefly discuss the impact of machine learning on image and video compression. The following subsections give an overview of these categories.

Table 1. Required Compression Ratios of Video (60 FPS, No Subsampling, 10 Bits/channel) at Different Resolutions Transferred Over Various Channels

Raw (60 FPS, 10-bit, 4:4:4)		Video Interfaces			Networks					
		3G-SDI	DP 1.4	HDMI 2.1	4G	5G mm	WiFi-5	WiFi-6	25 GbE	5G fu
Resolution	Gb/s	2.9	25.9	42.6	1.0	2.0	3.5	11	25	50
1080p	3.7	1.3	—	—	3.7	1.9	1.1	—	—	—
4K	14.9	5.1	—	—	14.9	7.5	4.3	1.4	—	—
8K	59.7	20.6	2.3	1.4	59.7	29.9	17.1	5.4	2.4	1.2

The channel bandwidths are listed as theoretical maximum values. "DP": DisplayPort standard; "5G mm": 5G mmWave standard; "5G fu": future 5G at short wavelengths.

3.1 Still Image Compression

Because video is essentially a stream of images, image compression algorithms can be used to compress video. This is the broadest and most generic category we define. Typical image compression is performed in three stages. First, the input samples are decorrelated. Decorrelation can be achieved with a variety of methods such as transform (e.g., discrete cosine transform, DCT), prediction (e.g., median edge detection, MED) or sub-band coding (e.g., discrete wavelet transform, DWT), often prepended by a color transform that splits the color channels into luminance and chrominance components. Decorrelated samples are then subjected to statistical modeling such as quantization discarding the least important samples, reordering that ensures the most important samples are transported first, or context modeling, which can be used to exploit local image features by predicting the error residuals. The final step is entropy coding, which reduces the statistical redundancies between samples, for example, Huffman, arithmetic, or Golomb-Rice coding. Decompression is performed by repeating the inverse operations of the encoding steps in reverse order: entropy decoding, sample restoration (de-quantization), and inverse transform, possibly followed by a color transform into the original domain.

The most widely used algorithms in this category are variants of the joint photographic experts group (JPEG) algorithm: The original JPEG [143], its more complex improvement JPEG 2000 [130], or JPEG-LS [149] meant primarily for lossless or near-lossless image coding. New JPEG XL [118] standard is intended to supersede the original JPEG for web applications.

3.2 Hybrid Video Compression

Nearly all modern video codecs share the same hybrid coding scheme: First, they look for redundancies both spatially (intra-frame) and temporally (inter-frame) and represent the differences between similar blocks instead of encoding all the redundant pixel data. This enables a reasonably accurate prediction of the original data. The prediction residuals are then encoded via a procedure similar to transform-based still image coding. Typically, a variation of discrete cosine transform (DCT) is used as a decorrelation transform, followed by a quantization driven by a psychovisual model. Finally, the quantized coefficients are entropy-encoded, typically through context-adaptive variable-length coding (CAVLC) or context-adaptive binary arithmetic coding (CABAC). In addition, the modern video encoding flow also includes in-loop filtering, especially for deblocking. Because of the added coding tools, hybrid video codecs can achieve rate-distortion performance superior to typical still image codecs.

Figure 2 shows generalized block diagrams of a video encoder and decoder. The blocks in the gray zone denote the core of the video coding algorithms, which is not fundamentally different from transform-based still image coding. However, the block diagram illustrates that apart from the additional computation of the prediction residuals, the intra and inter prediction complicate the data flow by introducing a feedback loop that replicates a part of the decoding process. Apart

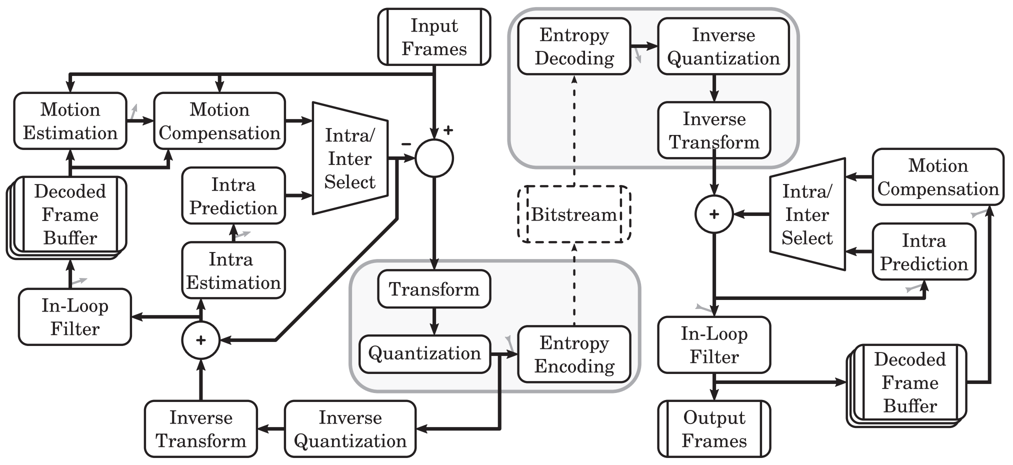

Fig. 2. Block diagram of a hybrid video encoder (left) and decoder (right). Gray zones denote the core of the algorithms that is similar to transform-based still image coding. Grey arrow pins denote intra-/inter-prediction and filter control data transferred to (from) the entropy encoder (decoder).

from the residuals, additional control data from the prediction estimation modules and the in-loop filters, required to restore the frames on the decoding side, are also encoded by the entropy coder (denoted by gray arrow pins in the figure). Additional memory storage is required for storing reference frames required by inter prediction.

Apart from regular video streams, special video formats were encountered during the survey, which often require additional processing steps during encoding. Notably, 360° video used in VR applications require stitching or a reprojection of the multi-camera view into a rectangular planar format [112].

First group of video coding standards are codecs developed by MPEG: *AVC (H.264)* [151], *HEVC (H.265)* [132], and the newest *VVC (H.266)* [20]. Each generation aims to approximately double the compression ratio for the same quality. Compared to Advanced Video Coding (AVC), HEVC simplified the in-loop filtering and added explicit support for parallelism via wavefront parallel processing (WPP), slices, and tiles. VVC further improves the coding efficiency of HEVC and extends its scope to be more versatile, i.e., usable in a broader range of applications such as wide-gamut, high dynamic range (HDR) or 360° video, still image coding or lossless coding.

Another line of codecs are open, royalty free formats by Google (the latest by Aliance for Open Media): *VP8* [13], *VP9* [100], and *AOMedia video 1 (AV1)* [28].

In addition, audio and video coding standard Workgroup of China developed their variants of video codecs: AVS [46], AVS2 [50], and AVS3 [160].

3.3 Mezzanine Compression

Mezzanine compression algorithms are intended to reduce the bitrate of high-resolution video transferred over high-bandwidth links, such as HDMI, DisplayPort, or a 10 Gb Ethernet. These algorithms are intended to be visually lossless with low latency and low complexity. Common use cases of mezzanine compression are professional video production chains or compression of a high-resolution video stream between a video card and a monitor. Willème et al. [152] define requirements for mezzanine compression as follows:

- Support for 1080p–8K, 8–12 bits per component, 4:4:4 and 4:2:2 chroma subsampling.
- Compression ratio in the range of 1.5–6:1.

- Visually lossless quality.
- Quality robustness in case of multiple encode-decode passes.
- Low computational complexity and low latency (<32 lines of a frame).

While mezzanine compression could be categorized as a part of still image compression, the low complexity, and low latency requirements justify a separate category. However, since the number of publications found was very low, the results are merged into the still image coding category in Section 4.

Some of the most well-known mezzanine compression algorithms are VC-2 [18], VESA DSC [144], and the newest standard JPEG XS [35]. Multiple proprietary formats exist such as LLVC by Sony or tiny codec (TICO) by intoPIX.

3.4 Texture Compression

Texture compression algorithms are primarily designed for offline compression of textures, which are stored into a GPU memory in the compressed state. During rendering, the texture components (texels) are repeatedly read from and decompressed at runtime during the display process. The fast decoding is therefore of utmost importance, and modern GPUs support hardware texture decompression to make the decoding process as fast as possible. Beers et al. [15] define the following design goals of a texture compression algorithm:

- Very fast decoding speed.
- Fast random access to the pixels in the texture, which implies a fixed compression ratio.
- Visual quality can be compromised for higher compression.
- Fast encoding is useful but not necessary.

Typically, the encoding process evolves around designing a codebook that quantizes the original color space and representing pixels as indices pointing into the quantized codebook. Decompression is thus achieved simply with a table lookup. Compared to previous categories, texture compression lacks the entropy coding step, which typically results in worse rate-distortion performance.

There are many variants of texture compression formats, mostly depending on the platform. Desktop GPUs support hardware decompression of the S3 texture compression (S3TC) family (also called block compression (BC) or DirectX texture compression (DXT)): BC1–5 [21, 62, 123] or the more complex BC6H (for HDR textures) and BC7 (for ordinary textures) [88, 124]. The newest evolution of this family is adaptive scalable texture compression (ASTC) [101], which supports variable block sizes and therefore multiple possible compression ratios. It is also the most complex texture compression format up to this date. Contrary to BCx, ASTC is only supported on mobile devices and Intel GPUs. For mobile Android-based devices, Ericsson texture compression (ETC)-based compression is used: ETC1 or ETC2 and EAC. Apple devices support their PowerVR texture compression (PVRTC) compression standard.

3.5 The Role of Machine Learning

In the past decade, the use of machine learning, and especially deep learning, has risen dramatically. We dedicate this last category to two distinct concepts connecting machine learning and image/video compression: the use of machine learning for image/video compression and the compression of visual content for machine vision applications (which typically utilize machine learning for the vision algorithm).

Machine Learning for Compression. As in many other fields, machine learning has also been used for solving hard optimization problems in image and video compression. Following a distinction made by Lie et al. [89] and Hoang and Zhou [59], we can divide applications of machine learning in video compression into two categories:

(1) Novel compression schemes built primarily with deep learning techniques.
(2) Use of deep learning tools in traditional coding pipelines.

Due to the high computational complexity of deep learning algorithms, most end-to-end implementations struggle to achieve high coding speed, which was confirmed during our literature search. Therefore, we decided to omit the end-to-end implementations from the survey. The mentioned studies [59, 89] can provide a further overview of this topic.

In some cases, we found deep learning used for accelerating complex steps in existing codecs. These publications are further discussed in their respective sections.

Compression for Machine Vision. Hand in hand with advances in machine learning, the range of machine vision applications has grown steadily. As pointed out by Le et al. [83], Cisco Annual Internet Report [30] predicts the share of machine-to-machine connections will grow from 33% in 2018 to 50% by 2023. Furthermore, the upcoming 5G latency improvements are expected to further accelerate the adoption of machine vision in low-latency systems. Thus, there is a growing need for compression optimized specifically for machine vision algorithms. However, only a handful of implementations were yet published, which we shortly list in this subsection without introducing a new category.

From the machine vision perspective, it can be beneficial to compress and transmit features extracted from the source pixels, instead of the pixels themselves. This is known as “feature compression” and two MPEG standards accompany this effort in the context of visual search applications: compact descriptors for visual search (CDVS) [41] and compact descriptors for visual analysis (CDVA) [42]. CDVS defines a set of handcrafted descriptors and a processing pipeline to ensure better reproducibility and interoperability between applications. CDVA further extends the concept CDVS by including deep features alongside the handcrafted ones and exploiting temporal redundancies between video frames. We found only three real-time implementations of CDVS [43, 51, 133] reaching 98, 144, and 39 MP per second on desktop-grade GPUs, respectively. Compared to other implementations reported later in the survey, the results are relatively slow, and more research would be necessary to decrease the computational complexity of CDVS and CDVA.

“Feature compression” might lead to optimal compression for machine vision, but the results are not directly interpretable by humans. The recent proposal of video coding for machines (VCM) [40] tries to find the balance between both the human and machine vision perception by finding a collaborative coding model.

Recently, end-to-end frameworks based on the idea of an autoencoder were proposed, specifically targeting the compression for machine vision [82, 83, 111]. In Reference [82], the authors achieved a throughput of only 14 MP per second on a high-end desktop GPU. However, the proposed method shows a great potential by outperforming Versatile Video Coding (VVC) in terms of machine vision accuracy.

Another meaning to “feature compression” was given by works that propose compressing intermediate deep features (e.g., Reference [45]). That is, instead of compressing the visual signal itself, part of the target vision neural network is executed on the source device, followed by the deep feature compression, transmission, feature decompression, and computing the rest of the neural network. However, we consider the approach a separate topic out of the scope of this survey.

4 STILL IMAGE COMPRESSION

This section presents the implementation results of still image and mezzanine compression algorithms. First, in Section 4.1, we mention implementations evaluating *motion-to-photon latency*. Then, the rest of the section is organized as follows:

JPEG Variants. (Section 4.2) This category includes both software (CPU and GPU) and hardware (field-programmable gate array (FPGA)) implementations of different JPEG standards. The results vary greatly given the wide range of design goals and target platforms.

Frame Memory Compression. (Section 4.3) A large group of the results consists of dedicated hardware accelerators (mostly application-specific integrated circuit (ASIC) but also FPGA) designed to reduce the memory footprint within another application (such as reference frame buffering in a video codec or a computer vision application). Despite being used as a part of a larger system, they can still be viewed as standalone image compression algorithms. Common characteristics of these systems are (a) pipelined architecture with a latency of units–tens of clock cycles, (b) lossless or near-lossless quality, (c) low compression ratio usually between 2–3, and (d) custom adaptations of basic statistical algorithms focusing on simplicity.

Mezzanine Compression. (Section 4.4) The mezzanine compression results are merged into this section, because only three implementations were found in the search process and the structure of the algorithms closely matches general still image coding. More specifically, VESA display stream compression (DSC) and JPEG XS algorithms are covered.

Other. (Section 4.5) Implementations of algorithms that do not fit to any of the above categories.

Table 2 summarizes basic properties of the reviewed implementations. Because of the large variety of different algorithms, the “Algorithm” column denotes the algorithm’s decorrelation and coding steps. The “Format” column denotes the color format and subsampling as specified by the authors. The “Len.” (length) column specifies whether the implementation uses variable-, semi-fixed-, or fixed-length coding. The “~” character anywhere in the table denotes an approximate value—for example read from a plot. The entries are sorted first by the coding type (encoding, then decoding), then by a throughput.

Figure 3 shows a visual comparison of throughputs and compression ratios of those reviewed implementations that report both values. Because the encoding and decoding throughputs usually vary, the figure shows them in two separate plots. The compression ratio is equal for both of the plots.

4.1 Implementations Reporting Latency

Holub et al. [60] proposed a GPU-accelerated coding of the JPEG standard. All encoding and decoding steps were performed on a GPU except the code stream parsing on the decoding side. The results in Table 2 correspond to encoding/decoding throughput of an 8K image at a quality level (Q) of 10 (highest TP) and 95 (lowest TP) for both the encoder and the decoder with and without memory transfers. With JPEG coding, the whole system exhibits a latency of 4 frames (133 ms), the same as an uncompressed video (with 1080i video input). Apart from JPEG, the publication also described GPU texture compression encoding, which is covered in more detail in Section 6.1.

Ubik et al. [138] proposed an FPGA implementation of TICO integrated into a 1080p video streaming system with the latency of 0.58 ms. The choice of a lightweight mezzanine compression

Table 2. Comparison of Reviewed Still Image Coding Implementations

Co-Processing

1st author	Algorithm	Len.	TP (Mpx/s)	Format	Platf.	Host	Device
Holub [60]	DCT$^{\#\#}$ + Huff./RLE$^{\#\#}$	var	1625$^{\circ\circ}$ E	YCbCr444 Q=10	CPU+GPU	Intel i7 3GHz	GTX 580
			824 E				
			1322$^{\circ\circ}$ E	YCbCr444 Q=95			
			634 E				
			2057$^{\circ\circ}$ D	YCbCr444 Q=10			
			899 D				
			1060$^{\circ\circ}$ D	YCbCr444 Q=95			
			586 D				
Bruns [24]	DWT^{++} + MSB^{++}	var	1584° D	YCbCr422	CPU+GPU	—	GTX 1080
			1269° D	YCbCr444			

Standalone HW

1st author	Algorithm	Len.	TP (Mpx/s)	Format	Platf.	Tech (nm)	Freq (Mhz)	Pow. (mW)	Ga/LUT ($\cdot 10^3$)	Mem (B)
Kim [72]	DWT + SPIHT	fix	4448 E	YUV420	ASIC	65	556	—	76.5	512
			4000 D				500		107.8	0
Guo [56]	MDA + SFL	semi	2133 E	YUV420	ASIC	90	300	—	45.1	—
			4267 D						34.5	
Guo [57]	DPCM + GOQ	var	800 E	YUV420	ASIC	40	300	2.05	4.95	—
			1600 D					2.60	9.75	
Lian [87]	custom + ExpGR	var	1540 E	YUV420	ASIC	65	578	5.3	36.5	192
			780 D				599	5.0	34.7	256
Kim [73]	custom + GR-like	fix	1113 E	RGBW	ASIC	130	167	—	56.1	1K
			1113 D						39.6	0
Kim [74]	DWT + SPIHT	fix	440 E	YCgCr422	ASIC	130	143	—	50.0	—
			477 D				167		68.8	
			354 E		FPGA	x	115	—	—	0
			397 D				139			
Kefalas [69]	custom† + DSU†	var	124 E	YCoCg422	FPGA	x	62	—	—	—
			62 E	YCoCg444						
Carneiro [25]	MED$^{\#}$ + GR$^{\#}$	var	121 E	—	FPGA	x	30	—	—	—
Kim [70]	MED$^{\#}$ + GR$^{\#}$	var	120 E	—	FPGA	x	—	—	—	—
Kau [68]	MED$^{\#}$ + GR$^{\#}$	var	113 E	—	FPGA	x	113	—	1.55	13K‡
Alonso [6]	LHE + Huff.	var	95 E	YUV/gray	FPGA	x	95	—	8.1	—
Chen [27]	PEP + GR$^{\#}$	var	52 E	12-bit gray	FPGA	x	—	—	—	264K
Kamalavasan [67]	DCT** + Huff.**	var	276 D	—	FPGA	x	200	900*	7.0	248K
Kim [75]	custom† + DSU†	var	246 D	—	ASIC	65	82	62	—	—

Standalone SW

1st author	Algorithm	Len.	TP (Mpx/s)	Format	Platf.	Model	#Cores
de Cea-Dominiquez [34]	DWT^{+} + BPC-PaCo	var	~810$^{\circ\circ}$ E	grayscale	GPU	GTX 1080 Ti	3584
			~760$^{\circ\circ}$ D				
Bruns [23]	DWT^{+} + EBCOT^{+}	var	186° E	—	GPU	—	—
			159° E				
Rubino [121]	DWT + ordering	var	69 E	—	CPU	Cortex-A53	4

*Dynamic power only, **JPEG XT, $^{+}$JPEG 2000, $^{++}$JPEG XS, $^{\#}$JPEG-LS or LOCO-I, $^{\#\#}$JPEG, †DSC, ‡ plus one row, $^{\circ}$only one-way memory transfers, $^{\circ\circ}$without memory transfers.
TP denotes throughput where E, D, and S denote encoding, decoding, and full system throughput, respectively. Ga/LUT denotes either the number of gates (ASIC) or LUTs (FPGA). "Cores" denotes the number of CPU or GPU cores.

algorithm was motivated by its latency of only several pixel lines and visually lossless quality. The system was used for distributed streaming of a musical performance over 300 km with an end-to-end latency of approximately 5 ms. However, the authors do not provide throughput measurements, therefore, we omit this publication from Table 3.

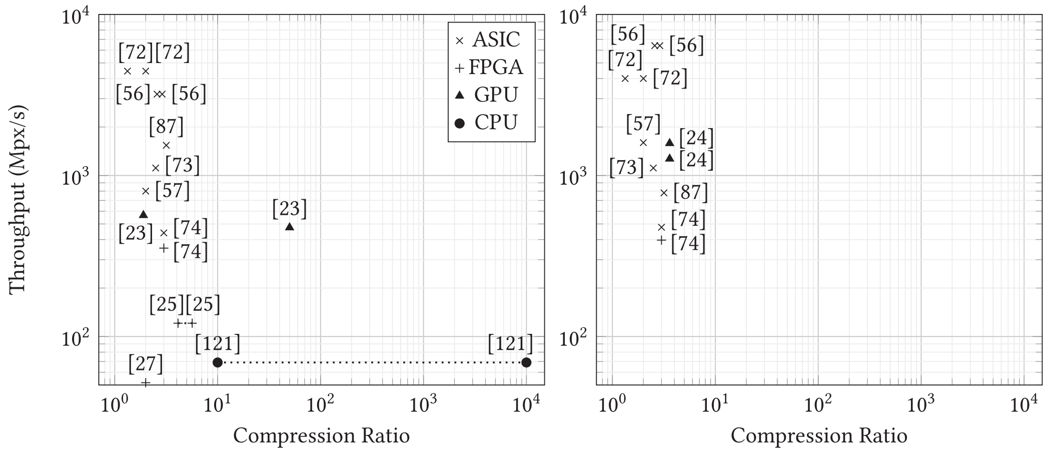

Fig. 3. Throughput of still image encoders (left) and decoders (right) compared to a compression ratio. Dotted lines represent multiple compression ratios reported over a range of values.

4.2 JPEG Variants

Three FPGA implementations of a JPEG-LS encoder were presented by Kim et al. [70], Kau and Lin [68] and Carneiro et al. [25]. Both References [68, 70] used a pipelined architecture while Reference [25] presented a parallel architecture for better scalability. LOCO-I, the core of the JPEG LS algorithm, was also implemented by Chen et al. [27] with a parallel error prediction (PEP).

Bruns et al. [23] evaluated different CPU/GPU co-processing models for post-compression rate-distortion optimization (PCRD-Opt) and packetization routines of JPEG 2000. The authors concluded that all coding steps should be executed on the GPU for the highest throughput. The throughput values in Table 2 refer to the low-bitrate (100 Mb/s, upper) and high-bitrate (2500 Mb/s, lower) scenarios reported in the publication. The authors reported that performing the whole encoding on a GPU reduces the device-to-host data transfers significantly.

A GPU implementation of JPEG 2000, proposed by de Cea-Dominiquez [34], replaced the embedded block coding with optimized truncation (EBCOT) entropy coding by a custom bit plane parallel coder called BPC-PaCo [44] and a bit stream tightening algorithm. The authors reported only a grayscale throughput; The corresponding throughput of an 8-bit RGB image would be $800/3 = 267$ Mpx/s. Also, the reported throughput does not take memory transfers to/from a GPU into account.

Kamalavasan et al. [67] proposed an FPGA JPEG XT decoder targeting decompression of 4K HDR images at 30 FPS in VR headsets. JPEG XT is a backwards-compatible extension to the original JPEG standard. The authors chose this standard for its lower computational complexity compared to JPEG 2000.

4.3 Frame Memory Compression

Guo et al. [55, 56] proposed a lossless embedded compression method based on multi-mode differential pulse code modulation (DPCM) and averaging (MDA) prediction followed by a semi-fixed length (SFL) coding. The length of a SFL-coded partition can be read from a fixed-length part of the partition, thus decreasing the number of data dependencies within the partition compared to a conventional variable-length coding. The basic processing unit is an 8×8 pixel block.

Kim et al. [74] proposed FPGA and ASIC implementations of a lossy frame memory compression (FMC) based on a discrete wavelet transform (DWT) and a 1D set partitioning in hierarchical trees (SPIHT). Contrary to the usual dynamic order of SPIHT, the encoder produces a fixed-length output and therefore has a fixed compression ratio and higher processing speed. By minimizing dependencies between different algorithm passes, the authors achieved a throughput of about one bit-plane per cycle. The results are obtained with a block size 1×64 and DWT decomposition level 3, evaluated on a Kodak dataset [48] transformed and subsampled into a YCbCr 4:2:2 format.

The previous design was improved by Kim et al. [72] by dividing a larger block into sub-blocks and processing them in parallel by small SPIHT processing elements. The results were obtained with a block size 8×8 and DWT decomposition level 2.

Another work by Kim et al. [73] introduced a lossy compression scheme for RGBW images. The algorithm consists of two custom RGBW prediction steps followed by a fixed-length Golomb-Rice coding. A coding block size 32×1 was used to obtain the results. Apart from a lossy mode, the algorithm can support lossless encoding as well.

A lossless FMC was proposed by Lian et al. [87]. Compared to previous works (e.g., Reference [56]), the implementation supports higher bit depths (such as 10 or 12 bits/sample) without a quality degradation and requires less memory. The authors developed a directional prediction method followed by a dynamic unary/Exp-Golomb-Rice coding and implemented the algorithm as two-core to maximize the throughput. The implementation uses encoding partition size 16×16 for luma and 8×8 for chroma channels.

Guo et al. [57] proposed a DPCM-based compression method where DPCM-predicted values are coded with a lossy gradient-oriented quantization (GOQ) coding. The choice of optimizing for a minimal gradient error was motivated by the fact that in computer vision applications the gradient information is more important. The encoder reads the input in $n \times 1$ sub-blocks and reorders them into $n \times k$ blocks suitable for vision processing. Because of that, the decoder reads its input in a block-based order.

4.4 Mezzanine Compression

Bruns et al. [24] proposed a GPU-based JPEG XS decoder. All decoding steps were performed on the GPU except for the initial depacketization, which was performed on a CPU due to its serial nature. To fully utilize the GPU and maximize the throughput, the computing kernels operate on the whole frame, thus sacrificing the low latency potential of the algorithm. The GPU–CPU memory transfer was a bottleneck for an unsubsampled 4K video with a bit depth of more than 8 bits. Compression ratios in Figure 3 were calculated from reported bits per pixel (bpp) values and 12 bits/channel video.

Kim et al. [75] proposed an ASIC implementation of a DSC decoder aiming for high power-efficiency. The work concentrates mostly on optimizing the line buffer, which was shown to be a major bottleneck in both area and power consumption.

Kefalas and Theodoridis [69] proposed an FPGA implementation of a DSC encoder without any external memory requirements. The encoder achieves a latency of less than one line. The achieved throughput is twice as high for 4:2:2- and 4:2:0-subsampled video than without subsampling.

4.5 Other

Alonso et al. [6] proposed an FPGA implementation of a logarithmical hopping encoding (LHE) prediction and quantization algorithm [9]. For the entropy coding step, the authors chose Huffman coding. Because the prediction is calculated as an average of the top and left pixels, the predicted value depends on the previous line of the image (as well as on the previous pixel of the current line). The authors demonstrated the algorithm's low complexity and memory requirements by implementing it on a small-factor FPGA-based system on chip (SoC) with a low resource consumption but still achieving enough throughput to stream 1080p video with a latency of 23 clock cycles.

Rubino et al. [120, 121] proposed a novel image compression algorithm depth embedded block tree (DEBT) in the context of underwater robotics applications. DEBT is a DWT-based progressive compression scheme that does not have an explicit entropy coding step. Instead, it scans the list of the transform coefficients ordered by their significance, sending the most significant elements first. According to the authors, the algorithm is highly parallelizable. An implementation of DEBT on a Raspberry Pi 3 embedded computer can achieve 30 FPS at 1080p resolution.

4.6 Summary

None of the results except Reference [60] (discussed in Section 6.1) and Reference [138] explicitly state full-system latency measurements. In the case of hardware implementations, the authors often report the length of the underlying pipelined architecture in clock cycles, which can be used to determine the resulting latency of the encoder/decoder. Software implementations only report encoding times for the whole image, therefore a latency of one frame has to be assumed.

As shown in Table 2, fixed-function ASIC or FPGA implementations of FMC are the fastest [55–57, 73, 74, 87]. Combined with low latency and good visual quality, they can be seen as a viable target for *ultra-low latency* video coding applications. However, the low compression ratio limits their usability to high-bandwidth transfer channels.

Only References [73, 74] implement fixed-length coding. This property is important in case any of these algorithms were implemented as software encoders, as fixed-length coding allows predictive memory allocations and simplifies the decoding process.

The results of Reference [24] showed that optimizing for maximum throughput might come at the expense of not achieving a sub-frame latency, which is one of the defining features of mezzanine codecs.

Some slower results come from implementations on small-scale embedded devices with a limited computational capacity [6, 121].

An apparent overall trend is to move away from typical entropy coding algorithms and focus on simpler solutions with fewer data dependencies. For applications following a co-processing execution model (most notably CPU-GPU), memory transfers between the device and the host can be a bottleneck.

5 HYBRID VIDEO COMPRESSION

Before reviewing the existing implementations, in Section 5.1, we shortly analyze the underlying complexity of the algorithms used in hybrid video codecs. This is possible, because algorithms presented in this subsection share similar coding principles, unlike, e.g., still image coding (Section 4). Furthermore, the complexity of dedicated video codecs is generally higher compared to other categories, thus giving a non-exhaustive overview of the underlying algorithms might provide a better insight into the possible limitations.

Publications reporting *motion-to-photon latency* of a full video system are presented in Section 5.2. The rest of the results is split into *Intra-frame Compression* (Section 5.3) and *Inter-frame Compression* (Section 5.4) depending on whether the implementation uses only intra-frame prediction or also inter-frame prediction. The division to intra-only and inter-frame coding is based on the fact that inter-frame prediction significantly increases the algorithm's complexity, data dependencies, and memory requirements to improve the coding efficiency. Results that do not fit any of the categories (e.g., because it is not clear whether they include inter-frame prediction or not) are reported in Section 5.5.

Table 3 shows a summary of the collected results. The pixel format column indicating the color space and bit-depth was omitted as most of the reviewed publications did not mention it specifically. However, the most common video configuration corresponds to the High Efficiency Video Coding (HEVC) main profile, which assumes YUV420 format with a bit depth of 8.

Table 4 shows Bjøntegaard metric [17] results from those publications that report them. Bjøntegaard metric calculates average PSNR (BD-PSNR) and bitrate (BD-Rate) difference between two rate-distortion curves. Other error metrics can be considered for the BD-Rate calculation, but we consider only PSNR unless noted otherwise. In the table, the reference column denotes the source of the reference rate-distortion curve—an HEVC or a AVC reference software (HM or JM, respectively) or an x265 open-source encoder [155].

5.1 Computational Complexity Analysis

Bossen et al. [19] provided an overview of HEVC complexity compared to AVC and described the reference HEVC software encoder and decoder. While the decoding complexity is designed to be similar to the previous-generation AVC format, a full-featured HEVC encoder is significantly more complex to meet the design goal of doubling the compression efficiency. Vanne et al. [139] conducted a rate-distortion-complexity analysis of HEVC against AVC. In the analysis, HEVC showed a 23–40% BD-Rate improvement at the cost of 3.2–1.2× encoding and 2.0–1.4× decoding overhead, depending on the test condition. Corrêa et al. [31] studied HEVC encoder complexity and concluded that just by selecting the right parameters, the encoding complexity can be decreased significantly without a significant efficiency cost.

Similar to HEVC, VVC increased the encoding complexity compared to its predecessor. According to Mercat et al. [96], the encoding efficiency (BD-Rate) of the reference VVC test model (VTM) compared to the reference HEVC model (HM) improved by 23–33.1% at the cost of 34–7.5× encoding complexity depending on the test condition. The authors reported a constant 1.8× increase of the decoding complexity. Similar results were reported by Siqueira et al. [128] and Pakdaman et al. [106]. Cerveira et al. [26] conducted a similar analysis focusing on memory usage profiling.

Multiple studies also included the AV1 codec in their comparisons: [52, 81, 94]. According to García-Lucas et al. [52], AV1 reference encoder does not outperform the HEVC reference encoder in terms of coding efficiency, despite the higher encoding complexity. However, both Mansri et al. [94] and Laude et al. [81] report AV1 as more efficient than HEVC. Saldanha et al. [122] conducted a review on hardware implementations of AV1 and VVC coding tools. The authors concluded that in the case of AV1, there are no hardware-based solutions for 2D hybrid transforms supporting any of the 16 allowed combinations. In the case of VVC, the authors claim that hardware implementations of inter prediction are still missing.

The next paragraphs provide a short overview of complexity reduction and dedicated acceleration of hybrid coding modules. Because considering all modules is out of the scope of this survey, we focus only on those commonly reported as the most expensive in the literature (e.g., References [19, 96, 106, 139]).

Inter prediction. Usually, inter prediction is reported as the most computationally expensive step, strongly affected by the quantization parameter (QP) value. Unlike its predecessor, HEVC adopted recursive splitting of macroblocks into coding units (CUs) and then into intra/inter prediction units. To reduce the added complexity, Shen et al. [126] proposed a CU split decision and early termination method for HEVC achieving 42/41% savings of encoding time at the cost of 1.49/1.15% of BD-Rate and 0.049/0.037% BD-PSNR increase. Shen et al. [127] proposed an inter-mode decision algorithm based on a statistical analysis. The authors saved 49–52% of the encoding time at a 0.60–0.88% BD-Rate increase. Another acceleration strategy is to offload the full inter prediction step, or a part of it, to a GPU [53, 58, 134] or an FPGA [10, 12]. A comprehensive review by Zhang et al. [164] covers literature on inter prediction of HEVC up to the year 2019.

Table 3. Comparison of Surveyed Hybrid Video Coding Implementations

Full System

1st author	Standard	Pred.	TP (Mpx/s)	Encoding Device	Decoding Device
Viitamäki [140]	HEVC	intra	829 S 747 S	Xeon E5-2699v4 + 2x Arria 10	3x laptop
Tu [137]	AVC*	—	442 S	i7-6800k + NVIDIA Titan Xp hwenc	HMD
Kim [71]	HEVC/SHVC*	—	126 S	i7-7700HQ	Galaxy S7 + i7-7700HQ
Pitkänen [112]	HEVC*	—	62 S	i7-7820HK	Galaxy S8 SoC hwdec
Fouladi [47]	VP8	—	55 S	Xeon E3-1240v5	Xeon E3-1240v5

Co-Processing

1st author	Standard	Pred.	TP (Mpx/s)	Type	Host	Device
Sjövall [129]	HEVC	intra	995 E	CPU+FPGA	Xeon E5-2680v4	2x Arria 10
Takano [134]	HEVC	inter	498 E	CPU+GPU	2x Xeon E5-2667v3	2x Titan X
Grossi [53]	VP8	inter	266 E	CPU+GPU	Xeon E5-2620v3	GeForce GTX 980
Adeyemi-Ejeye [3]	AVC	—	166 E	CPU+GPU	—	NVIDIA
Wang [145]	HEVC	inter	1,990+ D	CPU+GPU	Xeon E5-2699v3	Titan X
Ayadi [11, 12]	HEVC	inter	498 D	CPU+FPGA	Cortex-A9	Kintex-7
Liu [91]	MV-HEVC	inter	292 D	CPU+FPGA	Zynq UltraScale+ ZCU102	

Standalone HW

1st author	Standard	Pred.	TP (Mpx/s)	Type	Tech (nm)	Freq (MHz)	Power (mW)	Gates/LUTs ($\cdot 10^3$)	Mem (kB)
Omori [103]	HEVC	inter	995 E	ASIC	28	600	15,000	—	—
Yang [158]	AVC	—	553 E	ASIC	Titan X hwenc				
		—	202 E	ASIC	Iris 540 hwenc				
		—	157 E	ASIC	Intel HD 5000 hwenc				
Jiang [65]	AVC	—	415 E	ASIC	GTX 1080 hwenc				
		—	586 D	ASIC	GTX 1080 hwdec				
Xu [156]	HEVC	—	265 E	ASIC	28	350	103	2,880	117
Liu [90]	HEVC	inter	250 E	ASIC	10	504	57	3,550	172
Pastuszak [110]	HEVC	intra	249 E	ASIC	90	200	273	1,086	52
				FPGA	x	100	—	93.2	52
Zhang [162]	HEVC	intra	249 E	ASIC	90	320	236	2,288	120
Zhang [163]	HEVC	intra	249 E	ASIC	90	320	290	2,186	127
Ding [36]	HEVC	intra	124 E	FPGA	x	175	2,337	63.5	62.3
Zhang [162]	HEVC	intra	93 E	FPGA	x	120	—	201	120
Zhang [163]	HEVC	intra	93 E	FPGA	x	120	—	196	127
Mochizuki [98]	AVC	—	750 D	ASIC	16	400	197	—	—

Standalone SW

1st author	Standard	Pred.	TP (Mpx/s)	Type	Model	#Cores (used)
Tang [136]	HEVC*	—	252 S	CPU	i7-6700k	4 (1)
Huang [61]	AVC	inter	72 E	CPU	Intel Quad Core 3.3GHz	4
Jiang [64]	HEVC	inter	52 E	DSP	TMS320C6678	8
Wieckowski [150]	VVC	inter	~684 D	CPU	i9-9980HK	8 (8)
Zhu [166]	VVC	inter	537 D	CPU	i7-9700	8 (8)
Gudumasu [54]	VVC	inter	193 D	CPU	i9-9900X	10 (10)
Zhang [161]	AVS2	inter	131 D	CPU	Cortex-A57 + Cortex-A53	4+4 (4)
Wang [146]	AVS3	inter	70 D	CPU	Cortex-A73 + Cortex-A53	4+4 (1)

*360° video; +only one-way memory transfers.
TP denotes throughput where E, D, and S denote encoding, decoding and full system throughput. "cores" denotes the number of CPU cores. "hwenc" and "hwdec" denote the usage of a GPU-integrated hardware encoder or decoder.

Table 4. Quality and Bitrate Comparison of Publications That Report BD-Rate or BD-PSNR

1st author	Standard	TP (Mpx/s)	BD-Rate (%)	BD-PSNR (dB)	Reference
Takano [134]	HEVC inter	498 E	7.5	–	x265
Tang [136]	HEVC	252 S	−10.7	0.33*	–
Zhang [162]	HEVC intra	249 E	4.39	−0.21	HM 15.0
Zhang [163]	HEVC intra	249 E	2.79	−0.13	HM 15.0
Pastuszak [110]	HEVC intra	249 E	5.46	−0.13	HM 16
Ding [36]	HEVC intra	124 E	10.5	–	HM
Zhang [162]	HEVC intra	93.3 E	4.39	−0.21	HM 15.0
Zhang [163]	HEVC intra	93.3 E	2.79	−0.13	HM 15.0
Huang [61]	AVC inter	72.4 E	−18.1 −22.9	0.93 1.24	JM 19.0
Jiang [64]	HEVC inter	52.1 E	–	−0.93	HM 10.0

*spherical PSNR.

Han et al. [58] implemented a motion compensation algorithm for VVC decoding on a high-end desktop GPU, reaching 5 ms prediction time for 4K resolution (16× improvement over the reference inter prediction on a CPU and 46% acceleration of the whole decoding). Pan et al. [107] used a convolutional neural network to accelerate the VVC inter prediction, achieving 31% encoding time savings at the expense of 3% BD-Rate increase.

Intra prediction. The overall complexity of an encoder can be decreased by disabling the inter prediction and using the "all-intra" mode. In such a case, inter prediction overhead is zero and most of the computational burden lies on transform and quantization, intra prediction, entropy coding, and in-loop filtering.

Cho and Kim [29] proposed a decision strategy for early CU split and pruning decisions of HEVC intra coding based on a rate-distortion cost estimation. The combined method yielded an encoding time savings of approximately 50% with only 0.6% BD-Rate increase. Zhang and Ma [159] proposed a two-stage intra mode decision strategy that achieved 60% encoding time savings at the cost of 1.0% BD-Rate increase.

Compared to HEVC, the complexity of VVC intra prediction increased significantly by expanding the number of block partitionings and directional prediction modes and introducing new tools. Yang et al. [157] reduced the intra precision complexity of VVC by proposing a low-complexity coding tree unit partitioning and intra prediction mode decision scheme, achieving 63% encoding time savings at the cost of 1.93% BD-Rate increase over the reference VTM encoder. Lei et al. [84] addressed the increased coding tree unit partitioning complexity by pruning redundant partitions and deciding on the binary/ternary splitting direction in advance. This greatly reduces the number of modes to be searched, resulting in a 41% reduction of the encoding time and only a 0.84% increase of bitrate, compared to VTM. Dong et al. [39] reduced the VVC intra prediction complexity by pruning and early termination of non-important prediction modes. The authors achieved up to 53% encoding time savings at the cost of 1.08% BD-Rate increase. Li et al. [86] developed a tunable decision model for estimating the VVC CU partition allowing for early termination. The achieved encoding time savings range between 23–68% at the cost of 0.56–2.6% BD-Rate increase. A deep learning method for predicting the CU partitioning of VVC was presented by Li et al. [85]. The authors managed to reduce the overall encoding time by 45–67% with only a 1.3–3.2% BD-Rate increase.

Entropy coding. Entropy encoding and decoding present a challenge due to its serial nature and high branch divergence [145]. For this reason, Sjövall et al. [129] leave the CABAC encoding on the CPU side, together with other control logic, in their FPGA-based HEVC encoder. Similarly, a GPU-based HEVC decoder by Wang et al. [145] also leaves the CABAC decoding on the CPU side while the rest of the algorithm is implemented on a GPU. Despite the serial nature of CABAC, there have been efforts to reduce its data dependencies (Pastuszak [108, 109]) and increase its parallelism (Menasri et al. [95]). Notably, [108] achieves a throughput of 13.4 bins/cycle—high enough to encode high tier level 6.2 video (i.e., equivalent of 8K at 120 FPS). To the best of our knowledge, no literature has been published targeting the computational complexity of entropy coding specifically for VVC. However, as both HEVC and VVC use CABAC, improvements made for HEVC entropy coding are likely to be applicable also in the VVC context.

5.2 Implementations Reporting Latency

Tu et al. [137] proposed a 360° VR video viewing system with AVC coding and a 3,840 × 1,920 resolution operating at 60 FPS. Video is generated with six GoPro cameras, encoded on a GPU-equipped desktop computer, and sent over a wireless network to be displayed on an HMD. The authors measured the latency as approximately 500 and 160 ms with and without a wireless network and display overhead, respectively. For encoding, the authors used hardware codec integrated within the GPU and accelerated some preprocessing steps with compute unified device architecture (CUDA).

Yang et al. [158] proposed a desktop screen sharing system with AVC compression. To save the computation time, the authors exploited high temporal inter-frame redundancy and low-noise characteristics typical for screen content. The encoding was performed by up to four GPU-integrated hardware encoders. The results presented in Table 3 correspond to a high screen activity over the whole 4K screen as this scenario better corresponds to a general video streaming. The authors achieved 17–25 ms *motion-to-photon latency* on a high-end workstation with a 4K screen capture over a wired network for low–high screen activity. Over the wireless network, the authors achieved 33–65 ms latency for a 2,736 × 1,824 resolution.

Fouladi et al. [47] proposed Salsify—a real-time video conferencing architecture using a VP8 codec and a custom network transport protocol. The authors achieved 449 ms *motion-to-photon latency* over an emulated long-term evolution (LTE) network channel. The Salsify system combines a packet congestion control and a video codec rate control into a single algorithm to prevent them from disrupting each other.

Viitanen et al. [142] proposed a VR gaming cloud-edge system. The source 360° 1080p video is generated on a server running a game. Each frame is split into slices, which are encoded independently, each by its own instance of an open-source Kvazaar HEVC encoder [141], making use of WPP. The encoded video is streamed over a Gigabit Ethernet and decoded on the client with Open-HEVC software decoder [105]. The client laptop accepts control from and sends display frames to an HMD device. The authors were able to achieve close to 30 ms *motion-to-photon latency* while the system can maintain 30 FPS.

A direct continuation of Reference [142] is presented by Pitkänen et al. [112]. Instead of a laptop client, the authors used a smartphone mounted as an HMD and connected via a standard WiFi network. The authors achieved the same video throughput with a *motion-to-photon latency* close to 50 ms.

Tang et al. [136] proposed an omnidirectional stereoscopic video (ODSV) system achieving 2.2 s *motion-to-photon latency* and 0.5 s encoding latency. The relatively high system latency is caused by the complicated nature of the ODSV optical flow calculations. Based on the application and the presence of wireless network modules, we assume a wireless network was used even though this information was not clearly stated in the publication. Nevertheless, the system can maintain 30

FPS at 4,096 × 2,048 resolution with encoding performed by a modified x265 HEVC encoder [155] running on a single thread.

5.3 Intra-frame Compression

Pastuszak and Abramowski [110] proposed ASIC and FPGA implementations of an HEVC intra encoder achieving 30 FPS ecoding speed for a 4K video. The encoder is computationally scalable allowing to trade off the lower compression efficiency for higher throughput by choosing simpler methods for rate-distortion estimation and mode preselection.

Sjövall et al. [129] proposed an FPGA-accelerated Kvazaar HEVC encoder. The encoder runs on a Nokia AirFrame server—a 14-core Intel Xeon processor with two Arria 10 FPGA boards connected over peripheral component interconnect express (PCIe), each board accommodating three accelerator instances. The CABAC and the control logic are processed by the CPU due to the serial nature of the algorithm. The authors used WPP and picture-level parallel processing for increased parallelism. The setup is capable of encoding 4K video at 120 FPS.

Viitamäki et al. [140] demonstrated the previous approach on a more compact setup with 3 × 4K at 30 FPS interactive and 4K at 100 FPS non-interactive video streaming.

Zhang and Lu [162] proposed an HEVC intra encoder implemented as both ASIC and FPGA. The computational complexity is reduced at the expense of a slightly higher bitrate and quality loss. The design simplifies the luma and chroma prediction mode preselection, CU mode decision, and CABAC rate estimation algorithms to achieve higher throughput and fewer data dependencies. The design uses a pipelined architecture.

A follow-up work by Zhang and Lu [163] aims for a similar tradeoff between computational complexity and higher bitrate and quality loss. The main difference is a change in the mode prediction, which helps to reduce the worst-case BD-Rate but causes more data and timing dependencies. The potentially lower throughput was countered by improving the timing diagrams and optimizing the hardware architecture.

Ding et al. proposed a flexible HEVC intra encoder [36] implemented on an FPGA. The encoder is designed as a configurable framework of basic processing elements, which implement the encoder functionality. Overall, the encoder is implemented as a four-stage pipelined architecture. The authors parallelize the design by minimizing data dependencies between neighboring blocks in the intra prediction.

5.4 Inter-frame Compression

Onishi et al. [104] developed the first single-chip HEVC encoder (labeled as NARA) capable of encoding 4K resolution at 60 FPS with 4:2:2 subsampling that can also achieve 8K encoding in a multichip configuration. This encoder was then used in future works by Kobayashi et al. [76], Omori et al. [103] and Omori et al. [102]. In Reference [76], the authors improved the perceptual quality of HDR video encoding. In Reference [103], the authors achieved 4K, 120 FPS encoding by using four parallel NARA encoders, each configured for 2K, 120FPS encoding. Temporal scalability (i.e., encoding in a 60 or 120 FPS mode) was achieved by a base 60 FPS encoding stream optionally interleaved with another 60 FPS stream encoding frames in between the base layer frames. Similar principle is used in Reference [102] to achieve 4K, 120 FPS encoding. However, it uses two 4K instead of four 2K encoders. The encoder focuses specifically on a low latency achieving 21.8 ms encoding latency. Only the results from Reference [103] are included in Table 3 as it provides also the implementation details.

Takano et al. [134] proposed an HEVC encoder with an inter prediction (including transform and quantization of inter symbols) implemented on a GPU while the rest (intra prediction and entropy coding) is executed on a CPU. Instead of using neighboring blocks during the motion search, the authors separate the process into multiple layers with each layer reusing results from the previous one. This method relaxes the data dependencies between blocks and allows for better parallelization. The BD-Rate in Table 4 corresponds to a value reported at 60 FPS.

Liu et al. [91] proposed an FPGA-accelerated multi-view HEVC (MV-HEVC) decoder. The decoder operates on several inter-frames in parallel without considering data dependencies between them. Based on profiling of the reference HTM software, the authors decided to accelerate the slice decompression function. The authors made use of WPP for increased parallelism. The throughput presented in Table 3 is calculated from the highest achieved decoding speed for a three-view video at a total resolution of 1,920 × 1,088, QP of 25 and PIP inter-frame ordering.

Xu et al. [156] proposed a low-power ASIC HEVC encoder. The implementation makes use of frame memory compression to reduce the necessary memory bandwidth and power consumption. Also, the encoder can operate at three different performance levels offering a performance versus power savings tradeoff. The results in Table 3 are extracted from the high-performance profile.

Wang et al. [145] proposed a parallel GPU-accelerated HEVC decoder running on a server-grade CPU. The authors exploited both intra and inter levels of parallelism with WPP overlapping multiple inter-frames. All decoding steps except entropy decoding were performed on the GPU. The throughput value listed in Table 3 corresponds to the reported 8K encoding speed of 60 FPS. The authors specifically mentioned encoding the whole frame at once to achieve maximum available GPU throughput—as opposed to a block-based execution in their previous multi-core CPU-based design.

Grossi et al. [53] studied a CPU-GPU cooperative design of a VP8 encoder. The authors focused especially on accelerating the motion estimation stage as it was determined to be the main contributor to the execution time on a reference encoder. To achieve short encoding times, maximizing the usage of shared memory was shown to be more efficient than maximizing the GPU utilization. The throughput value in Table 3 was derived from a reported average total encoding time of 500 frames of a 1080p video sequence.

Zhang et al. [161] proposed an audio-video coding standard (AVS2) decoder intended for multi-view 3D applications on mobile devices. Based on an execution time analysis of a reference decoder, the authors focused on accelerating mostly the motion compensation module with single instruction multiple data (SIMD) extensions of the ARM CPU.

Liu et al. [90] proposed a ASIC implementation of a deep learning–assisted HEVC encoder capable of 4K encoding at 30 FPS. A neural network is trained to recognize the human visual contact field to estimate the visual attention distribution, which is then used to guide the selection of encoding parameters.

Huang et al. [61] proposed a modified AVC encoder with shaky video stabilization. The video stabilization is embedded directly into an existing encoder instead of encoding frames after the stabilization. To reduce the overall computational complexity, the stabilization vectors are used also during the motion vector search stage. The encoder was implemented both on a desktop and ARM-based mobile CPU.

Jiang et al. [64] proposed a SIMD-accelerated HEVC encoder implemented on a digital signal processor (DSP). The authors focused on improving data parallelism and minimizing the memory traffic overhead. The throughput value in Table 3 corresponds to a maximum reported FPS value for a 1080p video.

In References [11, 12], Ayadi et al. proposed an FPGA-accelerated HEVC decoder on a Zynq SoC platform. Based on the decoding time distribution of the reference decoder running on an ARM processor, the authors determined the calculation of interpolation filters during motion compensation to be the most computationally expensive part and implemented this function on the FPGA part of the Zynq board.

Wang et al. [146] developed a software implementation of a audio-video coding standard (AVS3) decoder on a mobile platform (Huawei P10). Even with a single thread, the authors achieved real-time decoding of a 1080p video.

A full software VVC decoder was proposed by Wieckowski et al. [150]. Their implementation achieved efficient decoding by using SIMD instructions. The authors also explored thread-level parallelism with both frame-level and coding tree unit (CTU)–level granularity. The throughput reported in Table 3 corresponds to the maximum achieved frame rate for 1080p video over the whole output bitrate range.

Gudumasu et al. [54] proposed a parallelized VVC software decoder. The authors achieved an average decoding speed of 23.3 FPS on Class A video sequences (4K resolution with a bit depth of 10 and YUV420 format).

Another real-time VVC software decoder was proposed by Zhu et al. [166]. The authors achieved a decoding speed over 60 FPS of a 4K video by the means of SIMD optimizations, multi-level parallelism, and efficient selection between 8- and 16-bit code paths. The result in Table 3 corresponds to the maximum achieved FPS with a QP of 37.

5.5 Other

Jiang et al. [65] proposed an accelerated AVC library that operates on top of existing encoders (either NVIDIA GPU hardware codec or a libx264-based software encoder [97, 154]). The authors were able to reduce the video streaming delay to only one frame and achieved 4K frame compression/decompression times 21.3/15.1 ms.

Mochizuki et al. [98] proposed an ASIC implementation of a 12-channel 1080p AVC video processing SoC for a car surround view system. The implementation achieved 70 ms decoding and skew compensation latency by concurrent execution of decoding processing units without waiting for several frames to buffer.

Kim et al. [71] proposed a 360° video streaming system based on the scalability extension of HEVC (SHVC) extension of HEVC. The video is encoded by a powerful server and sent over a 60 GHz wireless network (802.11ad) where it is displayed on a VR HMD (smartphone). To conserve the limited computational resources on the mobile device, parts of the decoding and postprocessing tasks were offloaded over the network back to the streaming server.

Adeyemi-Ejeye et al. [3] conducted a review of current possibilities and limitations of video streaming systems. The authors also implemented a GPU-accelerated AVC encoder with a 4K encoding speed of 15–30 FPS, depending on the test video sequence. As a part of the study, the authors measured the transmission latency of a 4K video, compressed with max. 160:1 ratio, via an 802.11n network operating at 5 GHz over a total distance of 40 m. Depending on the source video frame rate (24 or 30 FPS) and subsampling (4:2:0 or 4:2:2), the network latencies were in the range of 40–71 ms.

5.6 Summary

Viitanen et al. [142] and Pitkänen et al. [112] achieved *low latency* over a wireless network with their VR game streaming systems. Desktop streaming system by Yang et al. [158] also achieved *low latency* over a wireless network and even reached *ultra-low latency* when using a wired network at a low screen activity. The mentioned systems are relatively simple—the number of devices is limited and the network transfer is carried over a short distance. The collected results of *motion-to-photon latencies* suggest that the current generation of wireless networks is not suitable for *low latency* applications over a greater distance [47]. Likewise, achieving *low latency* is problematic for complicated setups involving external cameras or additional video processing steps [136].

Furthermore, most of the 360° video processing systems reviewed in this survey report also *motion-to-photon latency*, which underlines the importance of latency to VR systems.

The general trend to improve the coding speed is to reduce data dependencies to promote parallelism. In the case of software-programmable solutions, heterogeneous co-processing execution models (such as CPU+GPU or CPU+FPGA) seem to be more common than in the still image coding category, likely due to the more complex nature of video coding algorithms. Usually, the entropy coding is left to be performed on the CPU side. Implementations using inter prediction focus mostly on optimizing the motion estimation/compensation stage, which was shown to be the most computationally expensive part of the algorithms. Intra-only implementations most commonly focus on optimizing the intra prediction stage.

In the context of HEVC, the WPP feature proved to be essential for achieving high throughput [91, 129, 140, 145].

The newest VVC standard shows a promising decoding performance; however, we did not encounter any real-time encoders, likely due to the high computational complexity of VVC encoding and its early adoption stage.

A deep learning approach to reduction of HEVC encoding complexity by Liu et al. [90] shows the lowest power consumption of all the reviewed HEVC/AVC ASIC encoders. Deep learning tools have been also used [85, 107] to reduce the inter and intra prediction complexity of VVC, respectively.

The complexity reduction techniques for inter and intra prediction mentioned in Section 5.1 achieved time savings in the range of 31–67% at the expense of BD-Rate degradation of 0.6–3.2%. Table 4 shows that fast encoder implementations typically sacrifice the rate-distortion performance. Notable exceptions are References [61, 136], which managed to improve it.

6 TEXTURE COMPRESSION

This section presents the implementation results of texture compression algorithms.

In Section 6.1, systems reporting *motion-to-photon latency* are reviewed. Sections 6.2 and 6.3 cover encoding into simple and advanced formats, respectively. The division to simple and advanced is motivated by the large complexity variety between texture compression formats and their implementations. For the purpose of this article, BC7 and ASTC formats are classified as advanced because of the large number of possible configurations compared to other formats (e.g., BC1, BC3, ETC1/2, etc.). Furthermore, custom texture compression methods are also classified as advanced. Section 6.4 summarizes publications reporting decoding results. This includes decoding of native GPU formats as well as so-called "supercompressed" textures: textures compressed first with a standard texture compression algorithm, then again with a lossless encoder (e.g., LZMA) for a higher compression ratio.

Table 5 summarizes the results of the surveyed texture compression implementations. Since the volume of literature covering texture compression is relatively scarce, older results, as well as results with less throughput than 50 MP per second, are included in the comparison. Figure 4 visualizes the achieved throughput and compression ratio of the implementations.

6.1 Implementations Reporting Latency

Holub et al. [60] proposed a GPU encoding of BC1 and YCoCg-BC3 formats using a method originally developed by van Waveren and Castaño [148]. The authors also describe a GPU implementation of JPEG as discussed in Section 4.2. Memory transfers between a CPU and a GPU were identified as the major bottleneck, degrading the performance of the otherwise faster GPU. The publication also describes a video system consisting of a capture card (video source), 10 Gb Ethernet link, decoding device, and another capture card (display). The *motion-to-photon latency* of the

Table 5. Comparison of Surveyed Texture Compression Implementations

Full System & Co-Processing & Standalone HW

1st author	Format	Year	Type	TP (Mpx/s)	Implementation Details
Pohl [113]	BC1	2017	full sys.	516–811 S	4x Xeon E5-2699 v3 enc + Intel HD 530 dec
Dolonius [38]	custom	2019	CPU+GPU	0.0025 E	i7-3930K
Pratapa [114]	MPTC-BC1	2017	CPU+GPU	320 D	HTC Nexus 9 with Tegra-K1 GPU
	MPTC-ASTC 4×4			149 D	
	MPTC-ASTC 8×8			546 D	
Zhou [165]	BC1	2016	FPGA	44 E	90 MHz, 1133 LUTs, 406 registers

Standalone SW

1st author	Format	Year	Type	TP (Mpx/s)	Model	#Cores (th. used)
Holub [60]	BC1	2013	GPU	3485°° E	GTX 580	512
				798 E		
	YCoCg-BC3			2942°° E		
				593 E		
	BC1			17898°° E	ATI 5990	2 × 1536
				349 E		
	YCoCg-BC3			12016°° E		
				305 E		
Pohl [113]	BC1	2017	CPU	7373 E	Xeon E5-2699v3	18
	ETC1			3686 E		
	ETC2			2836 E		
Waveren [148]	BC1	2007	GPU	1690°° E	GeForce 8800 GTX	128
	YCoCg-BC3			939°° E		
Renambot [117]	BC1	2007	CPU	207 E	—	—
Waveren [147]	BC1	2006	CPU	201 E	Core 2 2.9 Ghz	2
	BC3			128 E		
Krajcevski [78]	PVRTC	2014	CPU	2.70 E	i7-4770k	4 (1)
Krajcevski [77]	BC7	2013	CPU	~2.10 E	Xeon 2.4 GHz	40 (64)
Krajcevski [79]	BC7	2014	CPU	0.46 E	i7-4770k	4 (1)
Pratapa [115]	ASTC	2019	CPU	0.012 E	—	- (1)
Andries [8]	custom (Haar)	2017	GPU	20700°° D	GTX 980	2,048
	custom (bior(2,2))			12500°° D		
Dolonius [38]	custom	2019	GPU	2621°° D	GTX 1080	2,560
Cui [33]	BC1	2016	—	143 D	—	—
	custom BC1			127 D		
	ETC1			146 D		
	custom ETC1			130 D		
Cui [32]	ETC1	2016	—	131 D	—	—
	custom ETC1			146 D		
Strom [131]	ETC1 + custom	2016	CPU	0.87 D	laptop 1.3 GHz	—
	ETC1 + ZIP			4.37 D		

°°without memory transfers.
TP denotes throughput where E, D, and S denote encoding, decoding and full system throughput. "cores" denotes the number of CPU cores (with the number of threads used).

system was 4 frames (133 ms) for an uncompressed and JPEG-encoded video while both texture compression algorithms resulted in latency of 5 frames (166 ms).

Pohl et al. [113] proposed a foveated video streaming system consisting of a four-server encoding setup and a thin desktop client for decoding. The compression was performed with optimized BC1, ETC1, and ETC2 encoders using only one CPU. The authors achieved *motion-to-photon*

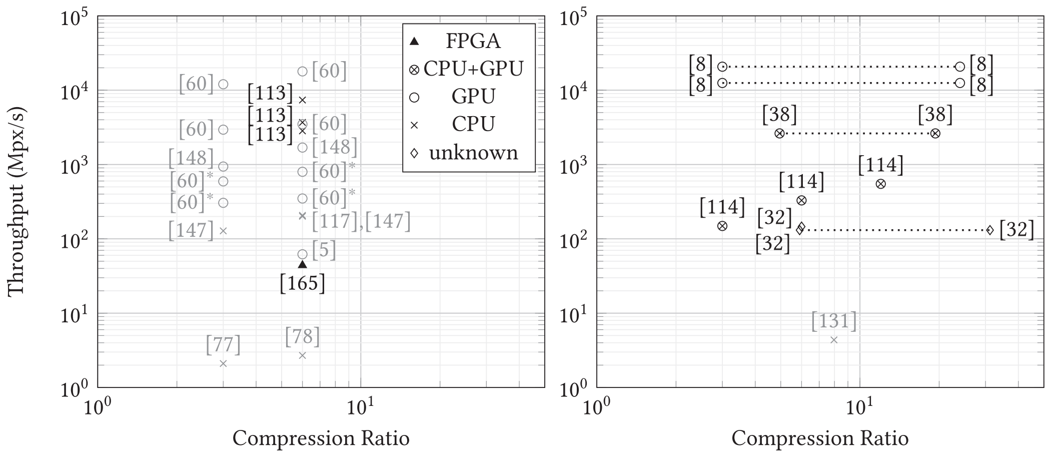

Fig. 4. Throughput of texture compression encoders (left) and decoders (right) compared to a compression ratio. Dotted lines represent multiple compression ratios reported over a range of values. Gray data points denote implementations older than the year 2015. *including memory transfers.

latency of 40–60 ms when streaming over a 10 Gb Ethernet link at both 5K and 1440p resolution. Compared to the AVC encoding, the authors achieved lower latency at the price of a higher data rate.

6.2 Encoders of Simple Formats

Van Waveren [147] proposed a real-time implementation of BC1 and BC3 encoders on a CPU using SIMD extensions. The codebook endpoints are selected using a bounding box method, i.e., selecting minimum and maximum color values within a block. Van Waveren and Castaño [148] proposed a GPU implementation of the same formats with BC3 modified to utilize the YCoCg color space [93] instead of the original RGBA. By storing the luminance (Y) value in the high precision alpha channel of the BC3 format, the authors achieve superior visual quality compared to BC1.

Renambot et al. [117] expanded upon van Waveren [147] by using C/C++ intrinsic functions instead of hard-coded SSE2 assembly instructions.

Krajcevski and Manocha [78] proposed a low-frequency signal modulated method for texture compression and applied it for a PVRTC encoding. By minimizing the number of computing passes through the entire texture, the authors managed to accelerate the encoding process and achieved 97 ms encoding time for a 512×512 texture.

Zhou et al. [165] proposed an FPGA implementation of a BC1 algorithm used for a frame memory compression. Even though the authors achieve low throughput, the resource utilization is minimal.

6.3 Encoders of Advanced Formats

Krajcevski et al. [77] proposed FasTC—an accelerated BC7 CPU encoder. For each block partitioning, the endpoints are estimated by a bounding box method [147] followed by a cluster-fit algorithm [22] to compute the indices. A texture of size $2{,}048 \times 2{,}048$ was encoded on a 40-core processor in about 2 s using 64 threads.

Krajcevski and Manocha [79] further accelerated FasTC by using image segmentation to select the block partitions (SegTC). Instead of choosing the partitioning for each block separately, the authors used a simple linear iterative clustering (SLIC) superpixels method [2] to assign the partition subset labels globally for the whole image. The method achieves about 4× speedup compared to FasTC while maintaining a similar quality.

Dolonius et al. [38] proposed their own block-based algorithm for compressing color data of voxelized surfaces. The algorithm follows an endpoint-based scheme similar to BC1. However, the encoding is variable-length and the decompression requires a binary search to locate a specific block. The most time-consuming part is the iterative search for the best endpoint color approximation. The results reported in Table 5 and Figure 4 correspond to the fastest results for an "Epic" scene in the publication.

Pratapa et al. [115] took a neural network approach to compress textures in the ASTC format. The most computationally intensive parts (partition selection and color endpoint approximation) are offloaded to the training phase. During the compression itself, the partition/endpoints search step is replaced by a neural network inference. The authors were able to achieve a speedup of almost 10× compared to a standard reference implementation.

6.4 Implementations Mentioning Decoding Speed

Ström and Wennersten [131] proposed a lossless compression of already compressed textures to further reduce the size of the texture—an approach also referred to as "supercompressed textures." With the proposed method, the authors were able to achieve the same bitrate as JPEG with similar quality. The reported decompession time in Table 5 corresponds to the CPU decompression time of a 512×512 texture with the ZIP algorithm, not the actual GPU texture decompression.

Pratapa et al. [114] proposed a supercompressed MPTC format and demonstrated it on real-time 360° video rendering. Unlike other texture compression formats, MPTC exploits both spatial coherence between multiple pixel blocks and temporal coherence between frames. The results in Table 5 correspond to a frame rate at which MPTC-compressed frames with 2K ($2{,}560 \times 1{,}280$) resolution can be read from a disk and rendered to a mobile device's screen.

Andries et al. [8], expanding previous work [7], proposed wavelet transform as an improvement to traditional texture compression formats. The wavelet coefficients are packed into a regular texture compression format and reconstructed on the decoding side in a shader. The authors list decoding times in the form of GPU fillrates for the different wavelet transform variants. The values reported in Table 5 correspond to the fillrate when restoring texture compressed with the Haar and biorthogonal(2,2) transforms with one level of wavelet decomposition. The method offers more fine-grained bitrate selection than conventional texture codecs and outperforms both ASTC and BC3 formats in terms of quality on the Kodak dataset [48].

Cui et al. [32, 33] proposed a novel texture compression method based on intra-frame block matching. When the encoder encounters an already encoded block, it encodes it as a reference to the encoded block, thus increasing the compression ratio. Since the number of the derived blocks is not known in advance, the resulting bitstream is of a variable length. The compression ratio in Figure 4 was calculated from the reported bpp values.

6.5 Summary

Considering *motion-to-photon latency*, Pohl et al. [113] managed to achieve *low latency* even at 5K resolution. The relatively high latency of 4–5 frames achieved by Holub et al. [60] seems to be caused by a more complicated pipeline where the additional capture cards can cause several frames of latency. Interestingly, even though the texture encoding itself was shown to be faster than JPEG encoding, the texture compression setup resulted in higher total latency. However, the authors do not discuss this difference further.

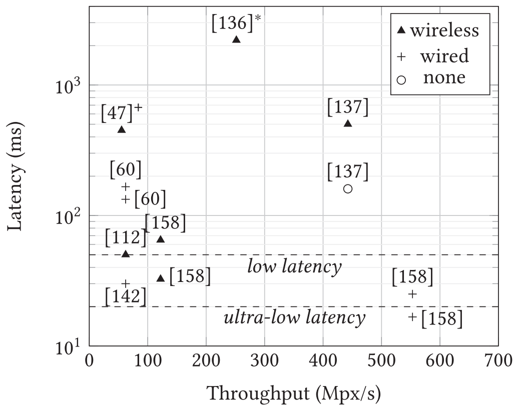

Fig. 5. Latency of video systems compared to the achieved throughput; *network type not 100% clear from the publication, +emulated.

Implementations of BC7 and ASTC encoders were shown to be too slow even for real-time encoding due to the complex nature of the algorithms. However, by reducing the number of configuration options, it is possible to reduce the complexity.

Fast encoding has been demonstrated for real-time implementations of simpler methods such as BC1, BC3, or ETC1/2. However, as illustrated by, e.g., Reference [148], real-time encoders of these formats compromise on visual quality compared to thorough offline encoders.

Fast decoding of compressed textures was demonstrated by Andries et al. [8], reporting GPU fillrates in the order of tens of gigapixels per second. The fillrate of NVIDIA GTX 980 GPU used by the authors is 144 gigatexels per second, which can be seen as a theoretical limit to the decoding performance. The real performance including the wavelet coefficients' restoration is thus only about an order of magnitude lower than the theoretical maximum.

The supercompressed approach yield promising rate-distortion performance [131] and decoding speed [114], however, the ecoding procedure is relatively complex and likely challenging to achieve in real time.

7 DISCUSSION AND CONCLUSIONS

7.1 Latency Discussion

In Figure 5, *motion-to-photon latencies* of video systems are visualized as a function of the system throughput. Reference [138] is not depicted, since the achievable throughput was not clear from the publication. However, the implementation achieves *ultra-low latency* over a long distance. Otherwise, it can be seen that reaching *low latency* is challenging for real-time video systems with high resolutions. The screen sharing system by Yang et al. [158] managed to achieve *low latency* (even *ultra-low latency* with a low screen activity) for a 4K resolution. Such a low latency was possible thanks to dedicated hardware encoders within a GPU and the rather simple nature of the screen sharing system consisting of only two interconnected computers. As seen in the previous sections, data transfers between multiple devices (such as external capture cards in Reference [60]) require buffering and can cause delays of several frames. Of course, the network is also a major contributor to the latency. Especially wireless networks are more unpredictable, with the link bandwidth being more dependent on the device proximity than in wired networks.

If we look at the achieved throughput only, then we can see that even though high-speed encoders and decoders exist, their employment in real systems is still rather limited. This suggests that the coding speed is not the main latency bottleneck of current video systems.

Table 6. Theoretical Expected End-to-end Latencies of Selected Implementations Taking Into Account Network Transport Times

			Latency								Total
	Video	Enc.	Encoder		Network		Decoder		CR	Enc. bitrate	latency
	input	unit		(ms)		(ms)		(ms)	—	(Gb/s)	(ms)
(a)	8K@120 FPS 12 bpp	8 lines	[72]	0.014	40 GbE	0.009	[72]	0.015	2	23.9	**0.038**
(b)	5K@60 FPS 24 bpp	1 frame	[113]	2.00	10 GbE	5.90	—	~0.59	6	3.54	**8.49**
					5G future	1.18					**3.77**
(c)	4K@120 FPS 12 bpp	1 frame	[129]	8.34	802.11n	1.17	[145]	4.17	142	0.084	**13.7**
					WiFi-6	0.06					**12.6**
(d)	4K@60 FPS 36 bpp	1 frame	—	8.34	WiFi-6	7.54	[24]	6.54	3.6	4.98	**22.4**

"CR": compression ratio.

7.2 Theoretical Achievable Latency Examples

To draw better conclusions, in Table 6, we estimate a theoretically expected latency of selected implementations from each category when transferred over a network channel. The total latency includes the encoding, decoding, and network data transfer latency, without taking into account the overhead of the network technology.

(a) Custom FMC ASIC by Kim et al. [72] operates on 8×8 blocks, therefore the minimal achievable latency is 8 lines assuming a row-major scanning order. The model input is 8K120FPS 8-bit video with 4:2:0 subsampling, which can fit into a 40 Gb Ethernet's bandwidth with the encoder's compression ratio of 2:1. Assuming a minimum achievable latency of 8 pixel lines, the final latency can be under 40 μs when transferred over a 40 Gb Ethernet. Such setup can be implemented as a part of a tightly integrated lightweight video system. This example assumes the whole system can operate at 8 lines granularity.

(b) This example uses BC1 compression in an in-home streaming system for 5K ($5{,}120 \times 2{,}880$) video developed by Pohl et al. [113]. Since the decompression speed for BC1 was not reported during the survey, we can use the fastest reported throughput 25 gigapixels per second of wavelet texture compression by Andries et al. [8] and consider it as a conservative estimate. By upgrading the network to a future 5G standard, it is possible to reduce the theoretical latency by more than twice.

(c) An example of a high-performance HEVC system uses an all-intra FPGA-accelerated encoder by Sjövall et al. [129] and a GPU-accelerated decoder by Wang et al. [145]. Since Reference [129] does not specify encoded bitstream size, it is extracted form a follow-up demonstrator by Viitamäki et al. [140] where the encoded bitstream was 21 Mb/s for a 4K@30FPS video. Both the encoder and the decoder require high-performance servers, which limits their usability in consumer applications. However, the high compression ratio makes it possible to efficiently transfer the bitstream even through an older wireless network (assuming a bitrate of 600 Mb/s for the 802.11n network). The new WiFi-6 could reduce the network latency to negligible values.

(d) JPEG XS GPU decoder by Bruns et al. [24] can decode one frame of 12-bit unsubsampled 4K video (i.e., 36 bpp) in 6.54 ms. Since no encoder was found during the survey, a hypothetical encoder with a throughput of 120 FPS is assumed. Decoded frames are assumed to stay in the GPU memory. This setup could be hypothetically used in a live HDR video streaming setup, still achieving *low latency*.

Only in case (a), we assumed sub-frame encoding. If buffering of the whole frame was required after the encoding and before the decoding stage, then the latency would increase $1,080\times$ to 42 ms. If we look at it from the other end, then case (b) uses a texture compression algorithm that is trivially parallelizable. By reducing the coding unit to 1/8 of the frame, it would be possible to reach a theoretical latency of 1.1 ms.

7.3 Summary of Reviewed Implementations

(1) Decreasing Latency. The main approach to decreasing video coding latency seems to be, not so surprisingly, increasing the coding throughput, since these goals usually go hand-in-hand.

We observed the following patterns used to improve the throughput or latency:

- Pipelined hardware architecture (e.g., Reference [57]).
- Reduction of computational complexity (e.g., Reference [34]).
- Increased parallelism (e.g., Reference [145]).
- More computing resources (e.g., Reference [113]).
- Utilization of hardware accelerators (e.g., References [65, 129]).

Very often, the above approaches are used in combination and are even prerequisites to each other. For example, it might be necessary to simplify an algorithm by removing data dependencies to achieve a higher degree of parallelism.

(2) Limitations of the State-of-the-Art. Most of the reviewed implementations (except for the pipelined ASIC and FPGA architectures) consider coding whole frames. An example of sub-frame AVC encoding with a programmable media processor was presented by Mody et al. [99]. The authors managed to reduce the latency from one frame (33 ms) to only 2 ms by using a sub-frame coding approach, which indicates a strong potential of sub-frame coding even for software-programmable implementations. As mentioned in References [24, 145], encoding whole frames might be necessary to preserve the high encoding throughput when utilizing massively parallel hardware such as GPUs.

When accelerating the coding process on parallel hardware, the most common step left to be performed on the CPU side is entropy coding. The serial nature of most of the entropy coding algorithms makes their parallel implementations challenging. Even though there has been a successful effort to parallelize and accelerate the process in hardware [108], parallel software implementations still seem to either execute entropy coding on a CPU [129] or implement a less complex parallel algorithm [34, 60].

To summarize the bottlenecks of each category, the fastest still image coding algorithms lack a high compression ratio. Therefore, their use is limited to only high-bandwidth networks. Hybrid video codecs achieve a high compression ratio at the expense of increased computational complexity. While some implementations achieve very high coding speeds, they usually require a significant amount of processing power. Although mezzanine codecs seem to be a promising solution for low-latency video coding, more research is necessary, as the number of discovered implementations is too low and their performance was found to be limited. Fast texture encoding can only be achieved with basic methods, which are limited in quality. High-quality formats require navigating through a large configuration space, which was shown to be too slow to achieve in low latency.

(3) Computational Requirements. For JPEG-based algorithms, desktop computers with high-end GPUs were used for coding at resolutions higher than 4K while both ASIC and FPGA, as well as embedded implementations, were found to be insufficient for processing high resolutions. ASIC implementations of FMC require about 20–80× fewer gates and a fraction of power consumption compared to HEVC, which makes it suitable for including into remote embedded systems.

High-throughput implementations of hybrid video codecs such as AVC or HEVC require a significant amount of processing resources and power [103, 129, 140, 145]. Decoding of the newest VVC standard was demonstrated at high throughput on consumer CPUs [150, 166]. A co-processing implementation by Ayadi et al. [12] was able to reach 4K@60FPS on a high-end Zyng-7045 embedded platform by utilizing both the CPU and FPGA parts, which makes it possibly the most lightweight solution for high-throughput HEVC compression covered in this survey.

Software implementations of texture encoding for resolutions higher than 4K use high-end desktop computers [60] and multi-server setups [113], although most of the processing power of the latter was used for rendering. Hardware GPU texture decompression was shown to be extremely fast, even with a custom restoration shader overhead [8]. The only hardware (FPGA) implementation of texture encoding [165] shows very low performance but also very low resource usage.

7.4 Future Prospects

Due to the increasing complexity of modern video codecs (such as VVC), latency-critical applications might require reformulating design goals of video compression with latency being the primary goal. After the latency target is met, improvements may be made towards a better quality or compression ratio. A promising standard in this regard is the recently introduced JPEG XS, which meets both high-quality and low-latency criteria. However, academic publications about this codec are scarce and to our best knowledge, no open implementation exists.

Deep learning has been used to accelerate complex steps of existing algorithms [85, 90, 107, 115]. While the computational cost of deep learning inference is very high, it might still be lower than some parts of modern complex formats such as ASTC or VVC. With hardware improvements and more research on lightweight neural network inference, this method can become a viable strategy for achieving low latency.

Machine vision–optimized compression can significantly outperform traditional human-centered video coding algorithms in terms of the rate-accuracy tradeoff, which in turn allows reducing the encoding configuration search space and thus accelerating the coding process. However, more research is necessary to explore this direction.

Even though it is possible to select a pipelined ASIC implementation of, e.g., an FMC encoder/decoder with a latency of less than one pixel line, system integration might prevent exploiting the low-latency potential due to buffering, since most of the encountered video systems assume a frame-level granularity. Therefore, research on sub-frame video system integration would help to leverage the full potential of existing *ultra-low latency* encoders and decoders.

Existing commercial GPUs feature hardware texture decompression, which is naturally simple and might as well be one of the overall fastest decompression solutions. From a video system integration perspective, a hardware texture compression would enable more optimal encoding performance as utilizing parallel hardware such as GPU is limited by the memory transfer overhead, despite the otherwise superior processing speed. In general, texture compression is limited by a poor quality versus complexity ratio and more research is necessary to achieve an acceptable encoding speed with advanced texture compression formats.

REFERENCES

[1] 3rd Generation Partnership Project (3GPP). 2018. *Study on Communication for Automation in Vertical Domains (CAV).* Technical Report 22.804.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. 2010. *SLIC Superpixels.* Technical Report.

[3] F. Adeyemi-Ejeye, M. Alreshoodi, L. Al-Jobouri, and M. Fleury. 2018. Prospects for live higher resolution video streaming to mobile devices: Achievable quality across wireless links. *J. Real-Time Image Process.* 16, 1 (2018).

[4] Z. Al-Ars et al. 2019. The FitOptiVis ECSEL project: Highly efficient distributed embedded image/video processing in cyber-physical systems. In *Proceedings of the International Conference on Computing Frontiers.*

[5] O. Alexandersson, C. Gurell, and T. Akenine-Möller. 2006. Compressing dynamically generated textures on the GPU. In *Proceedings of the ACM SIGGRAPH 2006 Sketches.*

[6] T. Alonso, M. Ruiz, Á. L. García-Arias, G. Sutter, and J. E. L. de Vergara. 2018. Submicrosecond latency video compression in a low-end FPGA-based system-on-chip. In *Proceedings of the International Conference on Field-Programmable Logic and Applications.*

[7] B. Andries, J. Lemeire, and A. Munteanu. 2014. Optimized quantization of wavelet subbands for high quality real-time texture compression. In *Proceedings of the International Conference on Image Processing.*

[8] B. Andries, J. Lemeire, and A. Munteanu. 2016. Scalable texture compression using the wavelet transform. *Visual Comput.* 33, 9 (2016).

[9] J. J. G. Aranda, M. G. Casquete, M. C. Cueto, J. N. Salmerón, and F. G. Vidal. 2015. Logarithmical hopping encoding: A low computational complexity algorithm for image compression. *IET Image Process.* 9, 8 (2015).

[10] L. A. Ayadi, T. Damak, H. Loukil, M. A. B. Ayed, and N. Masmoudi. 2018. HEVC decoder analysis on ARM processor. In *Proceedings of the International Multi-Conference on Systems, Signals Devices.*

[11] L. A. Ayadi, H. Loukil, M. A. B. Ayed, and N. Masmoudi. 2018. Efficient implementation of HEVC decoder on Zynq SoC platform. In *Proceedings of the International Conference on Advanced Technologies for Signal and Image Processing.*

[12] L. A. Ayadi, H. Loukil, M. A. B. Ayed, and N. Masmoudi. 2020. Hardware-software implementation of HEVC decoder on Zynq. *Multimedia Tools Appl.* 79, 11 (2020).

[13] J. Bankoski, P. Wilkins, and Y. Xu. 2011. Technical overview of VP8, an open source video codec for the web. In *Proceedings of the International Conference on Multimedia and Expo.*

[14] K. C. Barr and K. Asanović. 2006. Energy-aware lossless data compression. *ACM Trans. Comput. Syst.* 24, 3 (2006).

[15] A. C. Beers, M. Agrawala, and N. Chaddha. 1996. Rendering from compressed textures. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques.*

[16] M. Bennis, M. Debbah, and H. V. Poor. 2018. Ultrareliable and low-latency wireless communication: Tail, risk, and scale. *Proc. IEEE* 106, 10 (2018).

[17] G. Bjøntegaard. 2001. Calculation of average PSNR differences between RD-curves. VCEG-M33. https://www.itu.int/wftp3/av-arch/video-site/0104_Aus/VCEG-M33.doc.

[18] T. Borer. 2013. *The VC-2 Low Delay Video Codec.* White Paper WHP 238.

[19] F. Bossen, B. Bross, K. Suhring, and D. Flynn. 2012. HEVC complexity and implementation analysis. *IEEE Trans. Circ. Syst. Video Technol.* 22, 12 (2012).

[20] B. Bross et al. 2021. Overview of the versatile video coding (VVC) standard and its applications. *IEEE Trans. Circ. Syst. Video Technol.* 31, 10 (2021).

[21] P. Brown, S. Ian, N. Haemel, A. Pooley, A. Rasmus, and M. Shah. 2013. EXT_texture_compression_s3tc. Retrieved from https://www.khronos.org/registry/OpenGL/extensions/EXT/EXT_texture_compression_s3tc.txt.

[22] S. Brown. 2015. Libsquish. Retrieved from https://code.google.com/archive/p/libsquish.

[23] V. Bruns, M. Á. Martínez del Amor, and H. Sparenberg. 2017. Evaluation of GPU/CPU co-processing models for JPEG 2000 packetization. In *Proceedings of the International Workshop on Multimedia Signal Processing.*

[24] V. Bruns, T. Richter, B. Ahmed, J. Keinert, and S. Föel. 2018. Decoding JPEG XS on a GPU. In *Proceedings of the Picture Coding Symposium.*

[25] C. A. Carneiro, F. P. Garcia, H. C. Freitas, C. P. S. Martins, and F. M. F. Ferreira. 2017. Scalable spatio-temporal parallel parameterizable stream-based JPEG-LS encoder. *IEICE Electronics Express* 14, 2 (2017).

[26] A. Cerveira, L. Agostini, B. Zatt, and F. Sampaio. 2020. Memory assessment of versatile video coding. In *Proceedings of the International Conference on Image Processing.*

[27] L. Chen, L. Yan, H. Sang, and T. Zhang. 2018. High-throughput architecture for both lossless and near-lossless compression modes of LOCO-I algorithm. *IEEE Trans. Circ. Syst. Video Technol.* 29, 12 (2018).

[28] Y. Chen et al. 2018. An overview of core coding tools in the AV1 video codec. In *Proceedings of the Picture Coding Symposium.*

[29] S. Cho and M. Kim. 2013. Fast CU splitting and pruning for suboptimal CU partitioning in HEVC intra coding. *IEEE Trans. Circ. Syst. Video Technol.* 23, 9 (2013).

[30] Cisco. 2020. Cisco Annual Internet Report (2018–2023) White Paper. Retrieved from https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[31] G. Corrêa, P. Assunção, L. Agostini, and L. A. da Silva Cruz. 2012. Performance and computational complexity assessment of high-efficiency video encoders. *IEEE Trans. Circ. Syst. Video Technol.* 22, 12 (2012).

[32] L. Cui and E. S. Jang. 2016. Intra-picture block-matching method for codebook-based texture compression. *KSII Trans. Internet Info. Syst.* 10, 10 (2016).

[33] L. Cui, H. Kim, and E. S. Jang. 2016. A hybrid texture coding method for fast texture mapping. *J. Comput. Sci. Eng.* 10, 2 (2016).

[34] C. d. Cea-Dominguez, P. Enfedaque, J. C. Moure, J. Bartrina-Rapesta, and F. Auli-Llina. 2018. High throughput image codec for high-resolution satellite images. In *Proceedings of the International Geoscience and Remote Sensing Symposium.*

[35] A. Descampe, J. Keinert, T. Richter, S. Fößel, and G. Rouvroy. 2017. JPEG XS, a new standard for visually lossless low-latency lightweight image compression. In *Proceedings of the Applications of Digital Image Processing XL.*

[36] D. Ding, S. Wang, Z. Liu, and Q. Yuan. 2019. Real-time H.265/HEVC intra encoding with a configurable architecture on FPGA platform. *Chinese J. Electr.* 28, 5 (2019).

[37] M. Dohler et al. 2017. Internet of skills, where robotics meets AI, 5G and the tactile internet. In *Proceedings of the European Conference on Networks and Communications.*

[38] D. Dolonius, E. Sintorn, V. Kämpe, and U. Assarsson. 2017. Compressing color data for voxelized surface geometry. In *Proceedings of the Symposium on Interactive 3D Graphics and Games.*

[39] X. Dong, L. Shen, M. Yu, and H. Yang. 2021. Fast intra mode decision algorithm for versatile video coding. *IEEE Trans. Multimedia* 24 (2021), 400–414.

[40] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao. 2020. Video coding for machines: A paradigm of collaborative compression and intelligent analytics. *IEEE Trans. Image Process.* 29 (2020).

[41] L.-Y. Duan et al. 2015. Overview of the MPEG-CDVS standard. *IEEE Trans. Image Process.* 25, 1 (2015).

[42] L.-Y. Duan et al. 2018. Compact descriptors for video analysis: The emerging MPEG standard. *IEEE MultiMedia* 26, 2 (2018).

[43] L.-Y. Duan et al. 2018. Fast MPEG-CDVS encoder with GPU-CPU hybrid computing. *IEEE Trans. Image Process.* 27, 5 (2018).

[44] P. Enfedaque, F. Aulí-Llinàs, and J. C. Moure. 2017. GPU implementation of bitplane coding with parallel coefficient processing for high performance image compression. *IEEE Trans. Parallel Distrib. Syst.* 28, 8 (2017).

[45] A. E. Eshratifar, A. Esmaili, and M. Pedram. 2019. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *Proceedings of the International Symposium on Low Power Electronics and Design.*

[46] L. Fan, S. Ma, and F. Wu. 2004. Overview of AVS video standard. In *Proceedings of the International Conference on Multimedia and Expo.*

[47] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein. 2018. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *Proceedings of the Symposium on Networked Systems Design and Implementation.*

[48] R. Franzen. 2020. Kodak Lossless True Color Image Suite. Retrieved from http://r0k.us/graphics/kodak.

[49] S. Fürst and M. Bechter. 2016. AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform. In *Proceedings of the International Conference on Dependable Systems and Networks Workshop.*

[50] W. Gao and S. Ma. 2014. An overview of AVS2 standard. In *Advanced Video Coding Systems.* Springer, 35–49.

[51] A. Garbo and S. Quer. 2018. A fast MPEG's CDVS implementation for GPU featured in mobile devices. *IEEE Access* 6 (2018).

[52] D. García-Lucas, G. Cebrián-Márquez, and P. Cuenca. 2020. Rate-distortion/complexity analysis of HEVC, VVC, and AV1 video codecs. *Multimedia Tools Appl.* 79, 39 (2020).

[53] G. Grossi, P. Paglierani, F. Pedersini, and A. Petrini. 2018. Enhanced multicore–manycore interaction in high-performance video encoding. *J. Real-Time Image Process.* 17, 4 (2018), 887–902.

[54] S. Gudumasu, S. Bandyopadhyay, and Y. He. 2020. Software-based versatile video coding decoder parallelization. In *Proceedings of the Multimedia Systems Conference.*

[55] L. Guo, D. Zhou, and S. Goto. 2013. Lossless embedded compression using multi-mode DPCM averaging prediction for HEVC-like video codec. In *Proceedings of the European Signal Processing Conference.*

[56] L. Guo, D. Zhou, and S. Goto. 2014. A new reference frame recompression algorithm and its VLSI architecture for UHDTV video codec. *IEEE Trans. Multimedia* 16, 8 (2014).

[57] L. Guo, D. Zhou, J. Zhou, S. Kimura, and S. Goto. 2018. Lossy compression for embedded computer vision systems. *IEEE Access* 6 (2018).

[58] X. Han, S. Wang, S. Ma, and W. Gao. 2020. Optimization of motion compensation based on GPU and CPU for VVC decoding. In *Proceedings of the International Conference on Image Processing.*

[59] T. M. Hoang and J. Zhou. 2021. Recent trending on learning based video compression: A survey. *Cogn. Robot.* 1 (2021), 145–158.

[60] P. Holub, M. Šrom, M. Pulec, J. Matela, and M. Jirman. 2013. GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video. *Future Gen. Comput. Syst.* 29, 8 (2013).

[61] H. Huang, X.-X. Wei, and L. Zhang. 2019. Encoding shaky videos by integrating efficient video stabilization. *IEEE Trans. Circ. Syst. Video Technol.* 29, 5 (2019).

[62] K. I. Iourcha, K. S. Nayak, and Z. Hong. 1999. System and Method for Fixed-rate Block-based Image Compression with Inferred Pixel Values. U.S. Patent 5,956,431. (1999).

[63] Ixia. 2014. *Automotive Ethernet: An Overview.* White Paper Rev. A. Retrieved from https://support.ixiacom.com/sites/default/files/resources/whitepaper/ixia-automotive-ethernet-primer-whitepaper_1.pdf.

[64] H. Jiang, R. Fan, Y. Zhang, G. Wang, and Z. Li. 2019. Highly paralleled low-cost embedded HEVC video encoder on TI KeyStone multicore DSP. *IEEE Trans. Circ. Syst. Video Technol.* 29, 4 (2019).

[65] J. Jiang, T. Fogal, C. Woolley, and P. Messmer. 2016. A lightweight H.264-based hardware accelerated image compression library. In *Proceedings of the Symposium on Large Data Analysis and Visualization.*

[66] T. Kadowaki, M. Maruyama, T. Hayakawa, N. Matsuzawa, K. Iwasaki, and M. Ishikawa. 2018. Effects of low video latency between visual information and physical sensation in immersive environments. In *Proceedings of the Symposium on Virtual Reality Software and Technology.*

[67] K. Kamalavasan, R. Natheesan, K. Pradeep, S. Gowthaman, S. Aravinth, and A. Pasqual. 2019. FPGA IP for real-time 4K HDR image decoding in VR devices. In *Proceedings of the Latin American Symposium on Circ. Systems.*

[68] L.-J. Kau and S.-W. Lin. 2013. High performance architecture for the encoder of JPEG-LS on SOPC platform. In *Proceedings of the Workshop on Signal Processing Systems: Design and Implementation.*

[69] N. Kefalas and G. Theodoridis. 2019. Implementing VESA display stream compression encoder in FPGAs. In *Proceedings of the International Symposium on Power and Timing Modeling, Optimization and Simulation.*

[70] B.-S. Kim, S. Baek, D.-S. Kim, and D.-J. Chung. 2013. A high performance fully pipeline JPEG-LS encoder for lossless compression. *IEICE Electronics Express* 10, 12 (2013).

[71] H.-W. Kim, T. T. Le, and E.-S. Ryu. 2018. 360-degree video offloading using millimeter-wave communication for cyberphysical system. *Trans. Emerg. Telecommun. Technol.* 30, 4 (2018).

[72] S. Kim, J. H. Jang, H.-J. Lee, and C. E. Rhee. 2017. Fine-scalable SPIHT hardware design for frame memory compression in video codec. *J. Semiconduct. Technol. Sci.* 17, 3 (2017).

[73] S. Kim, M. Kim, J.-S. Kim, and H.-J. Lee. 2016. Fixed-ratio compression of an RGBW image and its hardware implementation. *IEEE Trans. Emerg. Sel. Topics Circ. Syst.* 6, 4 (2016).

[74] S. Kim, D. Lee, J.-S. Kim, and H.-J. Lee. 2016. A high-throughput hardware design of a one-dimensional SPIHT algorithm. *IEEE Trans. Multimedia* 18, 3 (2016).

[75] S. W. Kim, S. Park, J. Jun, and Y. Han. 2019. Design and implementation of display stream compression decoder with line buffer optimization. *IEEE Trans. Consum. Electron.* 65, 3 (2019).

[76] D. Kobayashi, K. Nakamura, T. Onishi, H. Iwasaki, and A. Shimizu. 2018. A 4K/60p HEVC real-time encoding system with high quality HDR color representations. *IEEE Trans. Consum. Electron.* 64, 4 (2018).

[77] P. Krajcevski, A. Lake, and D. Manocha. 2013. FasTC: Accelerated fixed-rate texture encoding. In *Proceedings of the Symposium on Interactive 3D Graphics.*

[78] P. Krajcevski and D. Manocha. 2014. Real-time low-frequency signal modulated texture compression using intensity dilation. In *Proceedings of the Meeting of the Symposium on Interactive 3D Graphics and Games.*

[79] P. Krajcevski and D. Manocha. 2014. SegTC: Fast texture compression using image segmentation. In *Proceedings of the Symposium on High Performance Graphics.*

[80] N. Lago and F. Kon. 2004. The quest for low latency. In *Proceedings of the International Computer Music Conf.*

[81] T. Laude, Y. G. Adhisantoso, J. Voges, M. Munderloh, and J. Ostermann. 2019. A comprehensive video codec comparison. *APSIPA Trans. Signal Info. Process.* 8 (2019).

[82] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, and E. Rahtu. 2021. Image coding for machines: An end-to-end learned approach. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing.*

[83] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, H. R. Tavakoli, and E. Rahtu. 2021. Learned image coding for machines: A content-adaptive approach. In *Proceedings of the International Conference on Multimedia and Expo.*

[84] M. Lei, F. Luo, X. Zhang, S. Wang, and S. Ma. 2019. Look-ahead prediction based coding unit size pruning for VVC intra coding. In *Proceedings of the International Conference on Image Processing.*

[85] T. Li, M. Xu, R. Tang, Y. Chen, and Q. Xing. 2021. DeepQTMT: A deep learning approach for fast QTMT-based CU partition of intra-mode VVC. *IEEE Trans. Image Process.* (2021).

[86] Y. Li, G. Yang, Y. Song, H. Zhang, X. Ding, and D. Zhang. 2021. Early intra CU size decision for versatile video coding based on a tunable decision model. *IEEE Trans. Broadcast.* 67, 3 (2021), 710–720.

[87] X. Lian, Z. Liu, W. Zhou, and Z. Duan. 2016. Lossless frame memory compression using pixel-grain prediction and dynamic order entropy coding. *IEEE Trans. Circ. Syst. Video Technol.* 26, 1 (2016).

[88] B. Lichtenbelt et al. 2019. ARB_texture_compression_bptc. Retrieved from https://www.khronos.org/registry/OpenGL/extensions/EXT/EXT_texture_compression_s3tc.txt.

[89] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu. 2020. Deep learning-based video coding: A review and a case study. *ACM Comput. Surv.* 53, 1 (2020).

[90] T.-M. Liu et al. 2018. A 0.76 mm^2 0.22 nJ/Pixel DL-assisted 4K video encoder LSI for quality-of-experience over smartphones. *IEEE Solid-State Circ. Lett.* 1, 12 (2018).

[91] W. Liu, W. Li, P. S. Un, and Y. B. Cho. 2018. High-throughput HW-SW implementation for MV-HEVC decoder. In *Proceedings of the International SoC Design Conference.*

[92] T. Mäki-Patola and P. Hämäläinen. 2004. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *Proceedings of the International Computer Music Conference.*

[93] H. S. Malvar and G. J. Sullivan. 2003. Transform, Scaling & Color Space Impact of Professional Extensions. ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031.

[94] I. Mansri, N. Doghmane, N. Kouadria, S. Harize, and A. Bekhouch. 2020. Comparative evaluation of VVC, HEVC, H.264, AV1, and VP9 encoders for low-delay video applications. In *Proceedings of the International Conference on Multimedia Computing, Networking and Applications.*

[95] W. Menasri, A. Skoudarli, A. Belhadj, and M. S. Azzaz. 2019. Field programmable gate array implementation of variable-bins high efficiency video coding CABAC decoder with path delay optimisation. *IET Image Process.* 13, 6 (2019).

[96] A. Mercat, A. Mäkinen, J. Sainio, A. Lemmetti, M. Viitanen, and J. Vanne. 2021. Comparative rate-distortion-complexity analysis of VVC and HEVC video codecs. *IEEE Access* 9 (2021).

[97] L. Merritt and R. Vanam. 2006. x264: A High Performance H.264/AVC Encoder. Retrieved from http://akuvian.org/src/x264/overview_x264_v8_5.pdf.

[98] S. Mochizuki et al. 2016. A 197mW 70ms-latency full-HD 12-channel video-processing SoC for car information systems. In *Proceedings of the International Solid-State Circuits Conference.*

[99] M. Mody, P. Swami, and P. Shastry. 2014. Ultra-low latency video codec for video conferencing. In *Proceedings of the International Conference on Electronics, Computing and Communication Technologies.*

[100] D. Mukherjee et al. 2013. The latest open-source video codec VP9—An overview and preliminary results. In *Proceedings of the Picture Coding Symposium.*

[101] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson. 2012. Adaptive scalable texture compression. In *Proceedings of the Conference on High-Performance Graphics.*

[102] Y. Omori, K. Nakamura, T. Onishi, D. Kobayashi, T. Osawa, and H. Iwasaki. 2019. 4K 120fps HEVC temporal scalable encoder with super low delay. In *Proceedings of the International Conference on Electronics, Circuits, and Systems.*

[103] Y. Omori, T. Onishi, H. Iwasaki, and A. Shimizu. 2018. A 120 fps high frame rate real-time HEVC video encoder with parallel configuration scalable to 4K. *IEEE Trans. Multi-Scale Comput. Syst.* 4, 4 (2018).

[104] T. Onishi et al. 2015. Single-chip 4K 60fps 4:2:2 HEVC video encoder LSI with 8K scalability. In *Proceedings of the Symposium on VLSI Circuits.*

[105] OpenHEVC 2017. Retrieved from https://github.com/OpenHEVC/openHEVC.

[106] F. Pakdaman, M. A. Adelimanesh, M. Gabbouj, and M. R. Hashemi. 2020. Complexity analysis of next-generation VVC encoding and decoding. In *Proceedings of the International Conference on Image Processing.*

[107] Z. Pan, P. Zhang, B. Peng, N. Ling, and J. Lei. 2021. A CNN-based fast inter coding method for VVC. *IEEE Signal Process. Lett.* 28 (2021), 1260–1264.

[108] G. Pastuszak. 2019. Generative multi-symbol architecture of the binary arithmetic coder for UHDTV video encoders. *IEEE Trans. Circ. Syst. I, Reg. Papers* 67, 3 (2019).

[109] G. Pastuszak. 2020. Multisymbol architecture of the entropy coder for H.265/HEVC video encoders. *IEEE Trans. Very Large Scale Integr. Syst.* 28, 12 (2020).

[110] G. Pastuszak and A. Abramowski. 2016. Algorithm and architecture design of the H.265/HEVC intra encoder. *IEEE Trans. Circ. Syst. Video Technol.* 26, 1 (2016).

[111] N. Patwa, N. Ahuja, S. Somayazulu, O. Tickoo, S. Varadarajan, and S. Koolagudi. 2020. Semantic-preserving image compression. In *Proceedings of the International Conference on Image Processing.*

[112] M. Pitkänen, M. Viitanen, A. Mercat, and J. Vanne. 2019. Remote VR gaming on mobile devices. In *Proceedings of the International Conference on Multimedia.*

[113] D. Pohl et al. 2017. The next generation of in-home streaming: Light fields, 5K, 10 GbE, and foveated compression. In *Proceedings of the Federated Conference on Computer Science and Information Systems.*

[114] S. Pratapa, P. Krajcevski, and D. Manocha. 2017. MPTC: Video rendering for virtual screens using compressed textures. In *Proceedings of the Symposium on Interactive 3D Graphics and Games.*

[115] S. Pratapa, T. Olson, A. Chalfin, and D. Manocha. 2019. TexNN: Fast texture encoding using neural networks. *Comput. Graph. Forum* 38, 1 (2019).

[116] Recommendation H.261 (11/88) 1988. *H.261: Video Codec for Audiovisual Services at p x 384 kbit/s.* Standard. ITU-T.

[117] L. Renambot, B. Jeong, and J. Leigh. 2007. Real-time compression for high-resolution content. In *Proceedings of the Access Grid Retreat.*

[118] A. Rhatushnyak et al. 2019. Committee Draft of JPEG XL Image Coding System. Retrieved from https://arXiv:1908.03565.

[119] M. E. Rose and J. R. Kitchin. 2019. pybliometrics: Scriptable bibliometrics using a Python interface to Scopus. *SoftwareX* 10 (2019).

[120] E. M. Rubino et al. 2017. Underwater radio frequency image sensor using progressive image compression and region of interest. *J. Brazil. Soc. Mech. Sci. Eng.* 39, 10 (2017).

[121] E. M. Rubino, A. J. Álvares, R. Marín, and P. J. Sanz. 2019. Real-time rate distortion-optimized image compression with region of interest on the ARM Architecture for Underwater Robotics Applications. *J. Real-Time Image Process.* 16, 1 (2019).

[122] M. Saldanha et al. 2020. An overview of dedicated hardware designs for state-of-the-art AV1 and H.266/VVC video codecs. In *Proceedings of the International Conference on Electronics, Circuits and Systems*.

[123] M. Satran and M. Jacobs. 2018. Block Compression (Direct3D 10). Retrieved from https://docs.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-resources-block-compression.

[124] M. Satran and M. Jacobs. 2018. Texture Block Compression in Direct3D 11. Retrieved from https://docs.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11.

[125] A. Shankar. 2019. Future automotive E/E architecture. *IEEE India Info. Newslett.* 14, 3 (2019).

[126] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang. 2012. An effective CU size decision method for HEVC encoders. *IEEE Trans. Multimedia* 15, 2 (2012).

[127] L. Shen, Z. Zhang, and Z. Liu. 2014. Adaptive inter-mode decision for HEVC jointly utilizing inter-level and spatiotemporal correlations. *IEEE Trans. Circ. Syst. Video Technol.* 24, 10 (2014).

[128] Í Siqueira, G. Correa, and M. Grellert. 2020. Rate-distortion and complexity comparison of HEVC and VVC video encoders. In *Proceedings of the Latin American Symposium on Circuits and Systems*.

[129] P. Sjövall, V. Viitamäki, J. Vanne, T. D. Hämäläinen, and A. Kulmala. 2018. FPGA-powered 4K120p HEVC intra encoder. In *Proceedings of the International Symposium on Circuits and Systems*.

[130] A. Skodras, C. Christopoulos, and T. Ebrahimi. 2001. The JPEG 2000 still image compression standard. *IEEE Signal Process. Mag.* 18, 5 (2001).

[131] J. Ström and P. Wennersten. 2011. Lossless compression of already compressed textures. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*.

[132] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circ. Syst. Video Technol.* 22, 12 (2012).

[133] W. Sun, X. Zhang, S. Wang, J. Chen, and L.-Y. Duan. 2017. GPU based fast MPEG-CDVS encoder. In *Proceedings of the International Conference on Image Processing*.

[134] F. Takano, H. Igarashi, and T. Moriyoshi. 2017. 4K-UHD real-time HEVC encoder with GPU accelerated motion estimation. In *Proceedings of the International Conference on Image Processing*.

[135] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. 2017. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Commun. Surveys Tuts.* 19, 3 (2017).

[136] M. Tang et al. 2019. A universal optical flow based real-time low-latency omnidirectional stereo video system. *IEEE Trans. Multimedia* 21, 4 (2019).

[137] J.-S. Tu, K.-S. Lin, C.-L. Lin, J.-Y. Kao, G.-R. Shih, and P.-H. Tsai. 2017. Low-latency implementation of 360 panoramic video viewing system. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems*.

[138] S. Ubik, J. Halák, J. Melnikov, and M. Kolbe. 2020. Ultra-low-latency video transmissions for delay sensitive collaboration. In *Proceedings of the Mediterranean Conference on Embedded Computing*.

[139] J. Vanne, M. Viitanen, T. D. Hämäläinen, and A. Hallapuro. 2012. Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs. *IEEE Trans. Circ. Syst. Video Technol.* 22, 12 (2012).

[140] V. Viitamäki, P. Sjövall, J. Vanne, T. D. Hämäläinen, and A. Kulmala. 2018. Live demonstration: 4K100p HEVC intra encoder. In *Proceedings of the International Symposium on Circuits and Systems*.

[141] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen. 2016. Kvazaar: Open-source HEVC/H.265 encoder. In *Proceedings of the International Conference on Multimedia*.

[142] M. Viitanen, J. Vanne, T. D. Hämäläinen, and A. Kulmala. 2018. Low latency edge rendering scheme for interactive 360 degree virtual reality gaming. In *Proceedings of the International Conference on Distributed Computing Systems*.

[143] G. K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Trans. Consum. Electron.* 38, 1 (1992).

[144] F. G. Walls and A. S. MacInnis. 2016. VESA display stream compression for television and cinema applications. *IEEE Trans. Emerg. Sel. Topics Circ. Syst.* 6, 4 (2016).

[145] B. Wang et al. 2018. Highly parallel HEVC decoding for heterogeneous systems with CPU and GPU. *Signal Process.: Image Commun.* 62 (2018).

[146] Z. Wang, B. Han, R. Wang, K. Fan, and W. Gao. 2020. uAVS3d: Fast decoder for the 3rd generation audio video coding standard (AVS3). In *Proceedings of the International Conference on Digital Signal Processing*.

[147] J. M. P. Van Waveren. 2006. *Real-time DXT Compression.* Technical Report. id Software, Inc.

[148] J. M. P. Van Waveren and I. Castaño. 2007. *Real-time YCoCg-DXT Compression.* Technical Report. id Software, Inc. and NVIDIA Corp.

[149] M. J. Weinberger, G. Seroussi, and G. Sapiro. 2000. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Trans. Image Process.* 9, 8 (2000).

[150] A. Wieckowski et al. 2020. Towards a live software decoder implementation for the upcoming versatile video coding (VVC) codec. In *Proceedings of the International Conference on Image Processing*.

[151] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circ. Syst. Video Technol.* 13, 7 (2003).
[152] A. Willème, A. Descampe, S. Lugan, and B. Macq. 2016. Quality and error robustness assessment of low-latency lightweight intra-frame codecs for screen content compression. *IEEE Trans. Emerg. Sel. Topics Circ. Syst.* 6, 4 (2016).
[153] C. Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering*.
[154] x264 .2020. Retrieved from https://www.videolan.org/developers/x264.html.
[155] x265 HEVC Encoder/H.265 Video Codec. 2020. Retrieved from http://x265.org.
[156] K. Xu et al. 2018. A low-power 4096x2160@30fps H.265/HEVC video encoder for smart video surveillance. In *Proceedings of the International Symposium on Low Power Electronics and Design.*
[157] H. Yang, L. Shen, X. Dong, Q. Ding, P. An, and G. Jiang. 2019. Low-complexity CTU partition structure decision and fast intra mode decision for versatile video coding. *IEEE Trans. Circ. Syst. Video Technol.* 30, 6 (2019).
[158] S. Yang, B. Li, Y. Song, J. Xu, and Y. Lu. 2018. A hardware-accelerated system for high resolution real-time screen sharing. *IEEE Trans. Circ. Syst. Video Technol.* 29, 3 (2018).
[159] H. Zhang and Z. Ma. 2013. Fast intra mode decision for high efficiency video coding (HEVC). *IEEE Trans. Circ. Syst. Video Technol.* 24, 4 (2013).
[160] J. Zhang, C. Jia, M. Lei, S. Wang, S. Ma, and W. Gao. 2019. Recent development of AVS video coding standard: AVS3. In *Proceedings of the Picture Coding Symposium.*
[161] Y. Zhang, Z. Lin, W. Feng, J. Sun, and Z. Guo. 2018. A real-time multi-view AVS2 decoder on mobile phone. In *Prococeedings of Advances in Multimedia Information Processing—Pacific Rim Conference on Multimedia.*
[162] Y. Zhang and C. Lu. 2019. Efficient algorithm adaptations and fully parallel hardware architecture of H.265/HEVC intra encoder. *IEEE Trans. Circ. Syst. Video Technol.* 29, 11 (2019).
[163] Y. Zhang and C. Lu. 2019. High-performance algorithm adaptations and hardware architecture for HEVC intra encoders. *IEEE Trans. Circ. Syst. Video Technol.* 29, 7 (2019).
[164] Y. Zhang, C. Zhang, R. Fan, S. Ma, Z. Chen, and C.-C. J. Kuo. 2019. Recent advances on HEVC inter-frame coding: From optimization to implementation and beyond. *IEEE Trans. Circ. Syst. Video Technol.* 30, 11 (2019).
[165] Y. Zhou, X. Jin, and T. Xiang. 2016. Fixed-ratio DXT format frame buffer compressor for mobile graphics systems. In *Proceedings of the International Conference on Field-Programmable Technology.*
[166] B. Zhu et al. 2021. A real-time H.266/VVC software decoder. In *Proceedings of the International Conference on Multimedia and Expo.*

Received July 2020; revised December 2021; accepted January 2022