Reprinted from
Journal of Imaging Science and Technology® 61(1): 010403-1–010403-13, 2017.

Architectures and Codecs for Real-Time Light Field Streaming

Péter Tamás Kovács
Tampere University of Technology, Tampere, Finland
Holografika, Budapest, Hungary
E-mail: peter.t.kovacs@tut.fi

Alireza Zare
Tampere University of Technology, Tampere, Finland
Nokia Technologies, Tampere, Finland

Tibor Balogh
Holografika, Budapest, Hungary

Robert Bregović and Atanas Gotchev
Tampere University of Technology, Tampere, Finland

Abstract. *Light field 3D displays represent a major step forward in visual realism, providing glasses-free spatial vision of real or virtual scenes. Applications that capture and process live imagery have to process data captured by potentially tens to hundreds of cameras and control tens to hundreds of projection engines making up the human perceivable 3D light field using a distributed processing system. The associated massive data processing is difficult to scale beyond a specific number and resolution of images, limited by the capabilities of the individual computing nodes. The authors therefore analyze the bottlenecks and data flow of the light field conversion process and identify possibilities to introduce better scalability. Based on this analysis they propose two different architectures for distributed light field processing. To avoid using uncompressed video data all along the processing chain, the authors also analyze how the operation of the proposed architectures can be supported by existing image/video codecs.* © *2017 Society for Imaging Science and Technology.*

INTRODUCTION

Three-dimensional (3D) displays[1,2] represent a new class of terminal devices, that make a major step forward in realism toward displays that can show imagery indistinguishable from reality. 3D display technologies use different approaches to make the human eyes see spatial information. While the most straightforward approaches use glasses or other headgear to achieve separation of left and right views, autostereoscopic displays achieve similar effects without the necessity of wearing special glasses. Typical autostereoscopic display technologies include parallax barrier,[3] lenticular lens,[4] volumetric,[5] light field[6,7] and pure holographic[8] displays, most of them available commercially, each with its unique set of associated technical challenges. Projection-based light field 3D displays[9–11] represent one of the most practical and scalable approaches to glasses-free 3D visualization, achieving, as of today, a 100 Mpixel total 3D resolution and supporting continuous parallax effect.

Various applications where 3D spatial visualization has added value have been proposed for 3D displays. Some of these applications involve displaying artificial/computer generated content, such as CAD design, service and maintenance training, animated movies and 3D games, driving and flight simulation. Other applications require the capturing, transmission/storage and rendering of 3D imagery, such as 3D TV and 3D video conferencing (3D telepresence). From the second group, applications that require live 3D image transmission are the technically most demanding, as 3D visual information has much more information content compared to its 2D counterpart. In state of the art 3D displays, total pixel count can be 1–2 orders of magnitude higher than in common 2D displays, making such applications extremely data intensive.

We focus on the problems associated with applications that require real-time streaming of live 3D video. We consider projection-based light field 3D displays as the underlying display technology (with tens or hundreds of projection engines) coupled with massive multi-camera arrays for light field capturing (with tens or hundreds of cameras). We explore possibilities that can potentially work in real time on today's hardware. Please note that the majority of the problems discussed here also apply to other 3D capture and display setups in which the necessary data throughput cannot be handled by a single data processing node, and as such distributing the workload becomes necessary. Our novel contribution presented in this article lies in: analysis of the typical data flow taking place during

Received July 17, 2016; accepted for publication Oct. 30, 2016; published online Dec. 20, 2016. Associate Editor: Hideki Kakeya.

converting multiview content to display-specific light field representation; analysis of bottlenecks in a direct (brute force) light field conversion process; presentation of two possible approaches for eliminating such bottlenecks; and a suitability analysis of existing image and video codecs for supporting the proposed approaches.

The article is organized as follows. First light fields and light field displays, as well as different content generation and rendering methods for light field displays are introduced. Then the proposed architectures for scalable light field decompression and light field conversion are described, followed by a comparative analysis of the discussed architectures and codecs.

LIGHT FIELDS, LIGHT FIELD DISPLAYS AND CONTENT CREATION

The propagation of visible light rays in space can be described by the plenoptic function, which is a 7D continuous function, parameterized by location in 3D space (3 parameters), angles of observation (2 parameters), wavelength and time.[12] In real-world implementations, a light field, which is a simplified 3D or 4D parameterization of the plenoptic function, is used, which allows to represent visual information in terms of rays with their positions in space and directionality.[13,14] In a 4D light field, ray positions are identified by either using two planes and the hit point of the light ray on both planes, or by using a single hit point on a plane and the direction to which the light ray propagates. This 4D light field describes a static light field in a half-space. For a more detailed description, the reader is referred to Ref. 15.

Having a light field properly reproduced will provide the user with 3D effects such a binocularity and continuous motion parallax. Today's light field displays can typically reproduce a horizontal-only-parallax light field, which allows the simplification of the light field representation to 3D.

Optical System and Image Properties

Projection-based light field displays are based on a massive array of projection modules that project light rays onto a special holographic screen to reproduce an approximation of the continuous light field. The light rays originating from one source (later referred to as light field slices) hit the screen at different positions all over the screen surface and pass through the screen while maintaining their directions. Light rays from other sources will also hit the screen all over the screen surface, but from slightly different directions (see Figure 1). The light rays that cross the screen at the same screen position but in different directions make up a pixel that has the direction selective light emission property, which is a necessary condition to create glasses-free 3D displays. By showing different but consistent information to slightly different directions, it is possible to show different images to the two eyes, with a specific displacement. Also, as the viewer is moving, eyes will see the other (previously unseen) light rays emitted—causing motion parallax effect. As the angular resolution between light rays is typically below 1°, the motion parallax effect is smooth; as the viewing angle can be up to

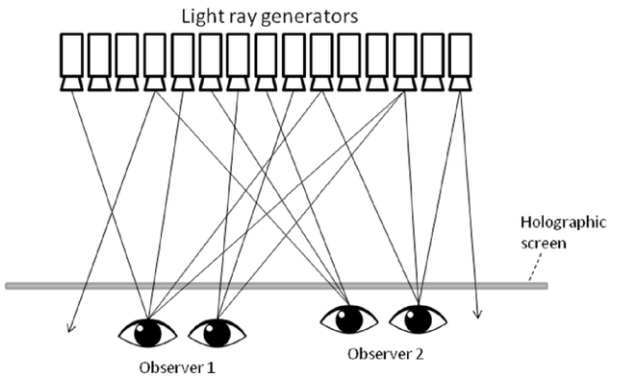

Figure 1. Basic principle of projection-based light field displays: projection modules project light rays from the back, toward the holographic screen. Object points are visible where two light rays appear to cross each other consistently. Human eyes can see different imagery at different locations due to the direction selective light emission property of the screen.

180°, the image can be walked around. The 3D image can be observed without glasses, and displayed objects can appear floating in front of the screen, or behind the screen surface, like with holograms.

This image formation is achieved by controlling the light rays so that they mimic the light field that would be present if the real object would be placed to the same physical space, up to the angular resolution provided by the projection modules. Such glasses-free display technology has dramatic advantages in application fields like 3D TV, and especially 3D video conferencing, where true eye contact between multiple participants at both sides can only be achieved by using a 3D display.[16] These same applications require real-time processing (capture, transmission, and visualization) of live images, which has special requirements on the rendering system.

Projection-based light field displays from Holografika have been used for the purposes of this work. The interested reader is referred to Ref. 7 for a complete description of the display system. Note that other projection-based light field displays based on very similar architectures also exist;[10,11,17] thus our findings are relevant for those displays too.

Rendering System and Data Flow

The previously described distributed optical system is served by a parallel rendering system, implemented as a cluster of multi-GPU computers. In the simple case, each GPU output drives a single optical module, multiple outputs of the same GPU correspond to adjacent optical modules, while one computer (having multiple GPUs) corresponds to a bigger group of adjacent modules. As in Ref. 18, we define the part of the whole light field that is reproduced by a single optical module a light field slice. One such slice is not visible from a single point of observation, nor does it represent a single 2D view of the scene. The set of light rays contained in such a slice is determined by the optical setup of the display and is calculated during the design process.

The cluster nodes are connected through a network (for example, Gigabit Ethernet, Infiniband, or 40 Gbit Ethernet),

which carries all the data to be rendered/visualized, unless it is precomputed and stored locally in the rendering nodes. The data received from an external source first needs to travel through the network, in the RAM of the rendering nodes, uploaded to and processed by the GPUs, output on a video interface, and finally displayed by the optical modules. The fastest light field conversion technique currently employed in light field rendering software is interpolation of light rays from the nearest samples. All data that contributes to the calculation of the color of a specific outgoing light ray thus needs to be available in the GPU responsible for that light ray.

Content Creation for Light Field Displays

For the sake of achieving the highest possible light field quality on the display side, we prefer to use a dense set of input views, so that no depth estimation and virtual view synthesis is necessary to perform the light field conversion; thus the rendering process involves light field interpolation only. This approach has been successfully used on many occasions for light field displays resulting in crisp and artifact-free images regardless of scene complexity, material properties, or visual effects involved. Therefore, this seemingly brute-force approach is preferred also for live imagery, so that any type of scene can be handled with no view synthesis artifacts.

While this requires a high number of capture cameras, installing such a rig of low cost cameras is feasible in fixed settings (such as a film studio or a video conferencing room). Installing one for capturing events, while more demanding, is also possible and is done commercially[19] as well as for experimental projects (although most of these target free viewpoint viewing and are not necessarily aware that the captured input can also be used as a light field).

If the captured light field is to be transmitted over longer distances, or stored, it is absolutely necessary to compress it due to the large volume of data. It might even be necessary to compress the video streams captured by the cameras (preferably inside the cameras) to make the data throughput manageable. While compressing tens or even hundreds of views may seem counterintuitive due to reasons of bandwidth requirements, recent results have shown that compressing an 80-view XGA resolution light field stream (while keeping good visual quality) is not much more demanding than compressing a 4K video signal in terms of consumed bandwidth.[20] It has to be noted that the complexity of compressing or even decompressing this many views using the technology discussed in the article does not allow for real-time operation today.

One of the most well-known camera arrays for capturing light fields is the Stanford Multi-Camera Array,[21] consisting of 128 video cameras that can be arranged in various layouts, such as a linear array of parallel cameras or a converging array of cameras having horizontal and/or vertical parallax. Numerous other multi-camera setups have been built since then for both research and commercial purposes, e.g., the 100-camera system at Nagoya University,[22] the 27-camera system at Holografika[23] (discussed later in this article), or the 30-camera system from USC Institute for Creative Technologies.[24] These camera systems capture a sufficiently dense (in terms of angular resolution) and wide (in terms of baseline) light field, so that the captured data can be visualized on a light field display without synthesizing additional views beforehand.

One can also use a single moving camera for capturing a static scene,[25] or a static camera with a rotating object[26] to obtain a light field. As the last two approaches capture a static light field, they are not in the scope of the discussion presented in this article, which is about processing light field video. Also please note that plenoptic, range, and other single-aperture capture methods are not discussed in this article, as those cannot capture a scene from a wide viewing angle, unless the captured scene is extremely close (small).

When imagery is derived from a geometrical representation (that is, rendered from a 3D model), the resulting 2D image is a projection of the 3D scene. Synthesizing 2D views from many viewing angles is relatively straightforward in this case, as creating an array of synthetic cameras and rendering additional images from those is a matter of scripting in most modeling tools such as 3ds Max, Maya, or Blender. As such, rendering a dense set of views to serve as light field content is only challenged by increased rendering time, and potentially 2D-only visual effects applied in the rendering pipeline which do not work consistently across multiple views. The practicality of this approach has been demonstrated with the Big Buck Bunny light field sequences,[27] or the San Miguel sequence.[28] Densely rendered sequences can be used for light field displays without synthesizing additional views. When rendering synthetic scenes, it is also relatively easy to extract ground truth depth information, as depth maps are commonly used during rendering. Thus, depth information is available as a byproduct of the process.

There are attempts to capture and represent LF content with sparser camera views while estimating the geometry (depth) of the scene and augmenting the available views with depth maps to be used for subsequent intermediate view interpolation (synthesis) in order to provide the missing rays. Many techniques have been proposed for rendering intermediate views from a sparse set of views,[29] as well as rendering extrapolated views from narrow angle content.[30] Most of these techniques are rooted from stereo matching for depth estimation, and depth-based view synthesis by pixel shifting[31] or warping,[32] and improving the resulting views by hole filling,[33] inpainting,[34] and other techniques. As soon as these techniques can provide sufficient quality and work in real time to generate the rays needed for the display based on live imagery, they will be used in LF displays.

Light field conversion is possible without explicit geometry/depth information about the scene. The simplest technique involves resampling the light field, which can be implemented by interpolating the 4D light field function from the nearest samples.[14] More involved is the lumigraph approach,[13] where a rough geometric model is obtained, which is then used to improve the quality of the rendered

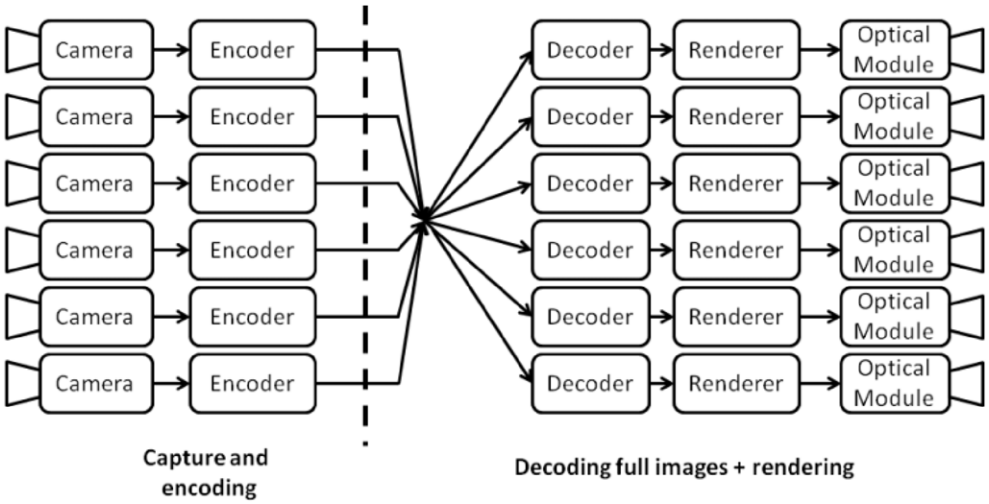

Figure 2. The brute-force decoder–renderer architecture. All encoded image data is received and fully decoded by each rendering node. The renderer then uses parts of the decoded image data to render the light field.

light field by defining the depth of the object being rendered, thus allowing for better ray space interpolation.

When it comes to live imagery, dense image-based techniques, while requiring a massive amount of images captured or rendered, result in the highest possible light field quality. This dense set of input views is easily converted to display-specific LF slices, however at the expense of getting the large number of input pixels to be available in the display's rendering nodes. The sheer amount of data necessitates novel streaming architectures and codecs that are able to handle such imagery.

Light Field Streaming

When performing light field streaming from an array of cameras, the simplest approach is to have all images captured by the cameras transferred to the GPUs of all rendering nodes to make sure all pixels that might be required during the light field conversion process are immediately available. This brute-force approach has previously been implemented and described in Ref. 23, and illustrated in Figure 2. In that system, a linear 27-camera array was used to feed a light field display. The cameras performed MJPEG compression of the images in hardware, which have been transferred to all rendering nodes simultaneously. The rendering nodes decompressed all 27 MJPEG images in parallel, and performed light field conversion on their respective light field slices on GPU. The performance of the whole system reached 15 FPS with 640 × 480 images, and 10 FPS with 920 × 720 images back in 2009.

The brute-force approach outlined above has several possible bottlenecks. First, the time necessary to decode all the JPEG images (either on CPU or GPU) increases linearly with the number of images (that is, number of cameras), and the resolution of the images. Second, the network bandwidth required to transfer all the compressed images to the rendering nodes increases linearly with the number and resolution of images. Third, the memory required to store the uncompressed images increases linearly with the number and resolution of captured images (both CPU and GPU memory).

If we consider an 80-view input (we used the BBB light field test sequences),[27] each view with 1280 × 768 resolution and 24 FPS, uncompressed, then the bandwidth required to transmit all pixels is 45.3 Gbit/s, which is difficult to reach with common network technologies.

Considering the same input, using 1:10 ratio JPEG compression, still gives a total bandwidth requirement of 4.5 Gbit/s and an averaged PSNR of 44.65 dB calculated over the luminance channel. While it is possible to transmit this amount of data through a 10- or 40-Gigabit Ethernet channel in real time, it is not feasible to decompress and process it. Using the fastest available libjpeg-turbo JPEG decoder library,[35] and a 6-core i7-5930K CPU@3.5 Ghz, one can decode 80 such views in ∼43 ms. If the system performs JPEG decoding only, this results in ∼23 FPS decoding speed, and this does not even include any further processing or rendering. Achieving higher resolution or processing a wider or denser set of views is clearly limited by hardware, and thus not scalable.

If we choose H.264 instead of JPEG for compressing the views using the x264 encoder[36] and maintaining a similar PSNR with $QP = 20$ (QP: Quantization Parameter, which determines image quality versus bandwidth in the video encoder), and use ffmpeg[37] to decode the 80 views on the same CPU in parallel, we can reach ∼16.94 FPS for decoding all views. In this measurement we used a RAM disk to avoid any significant I/O overhead.

Clearly, such a system having decoding time linear with the number of views and total pixel count cannot scale beyond a specific number of input cameras and resolution, given a specific hardware configuration. While the hardware can be upgraded to some extent, the processing power of a single node cannot be increased indefinitely (nor is it economical).

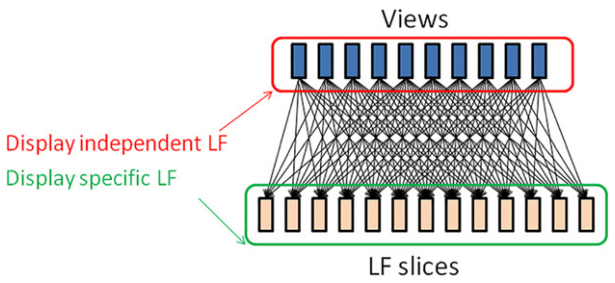

Figure 3. A set of perspective views depicting the same scene are considered as a display independent light field representation, as it can be used to visualize the light field on any suitable 3D display. The images required by a specific display's projection modules are referred to as light field slices, and the set of light field slices are therefore referred to as a display-specific light field.

While increasing the number of cameras and/or the resolution of these cameras increases the quality of the generated light field, the number of rendered light rays remains constant. It seems counterintuitive to use an increasing amount of input data to generate the same amount of outgoing light rays, and, as we will see in the next sections, this can be avoided.

Light Field Conversion and Data Access Patterns

Light field conversion transforms many input 2D views (which can be considered as a display independent light field representation) into light field slices, as required by the optical modules of the targeted light field display. One such light field slice is composed of many outgoing light rays (for example, 1024 × 768 pixels in case of an XGA optical module), which are in turn interpolated from pixels from many input views. Starting from views, we can also see that one input view typically contributes to many light field slices. Figure 3 shows that views typically contribute to many adjacent light field slices, and that light field slices are typically composed of pixels originating from many adjacent views.

An efficient implementation of the light field conversion process is using weighted look-up tables that map multiple pixels originating from views to an outgoing light ray, with the weighting appropriate for the given light ray. The pixel correspondences stored in the look-up table are calculated based on the geometry of the captured views (intrinsic and extrinsic parameters) and the geometry of the optical modules (display design, calibration data). The light field conversion process only needs to look up and blend the necessary view pixels to generate an outgoing light ray. As such, as long as the capture configuration, display configuration, and mapping between them does not change, the look-up tables remain constant, the same pixel positions are used while generating the same set of light rays.

In the light field conversion algorithm used with HoloVizio displays today, which uses a variant of the light field conversion algorithm described in Ref. 14, a maximum of two captured pixels contribute to one outgoing light ray. That is, the maximum number of light rays used by a rendering node cannot be more than twice the number

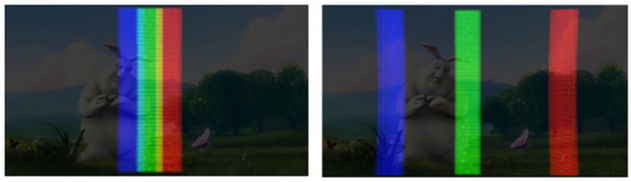

Figure 4. Left: adjacent rendering nodes consume adjacent, slightly overlapping parts of a source view. Red, green, and blue overlays (vertical stripes) represent the areas of the image used by three rendering nodes that drive adjacent optical modules. The areas are slightly overlapping. Right: Rendering nodes that drive optical modules positioned further away from each other use a disjoint set of pixels from the same source view, here the used areas are further away from each other.

of light rays generated by the same node, regardless of the number of total pixels captured. Take a HoloVizio 722RC display with 72 optical modules as an example, served by 6 rendering nodes, each rendering adjacent light field slices for 12 optical modules. Consider again an 80-view input, each view with 1280 × 768 resolution. Checking which pixels of a specific view are used by each rendering node, we can realize that rendering nodes that drive adjacent optical modules use slightly overlapping parts of the view (see Figure 4). However, rendering nodes that are driving optical modules positioned further from each other use a disjoint set of pixels from the input view. None of the other pixels of this view contribute to the final image. These pixels do not have to be transmitted to the renderer. It may even happen that a specific rendering node does not use any pixels of a specific view. In this case, dropping the view on the specific rendering node would not have any effect on the final image.

These properties of the typical data access patterns of light field conversion can be exploited to make the system more scalable, and eventually achieve real-time operation regardless of the number and resolution of the incoming views. Two proposed solutions are described next.

STREAMING ARCHITECTURES AND CODECS

Two-Layer Architecture

One possibility to scale the decompression workload without introducing any new bottlenecks is to introduce two layers in the system: one for decoding the incoming streams in parallel, and a second one to perform the light field conversion (see Figure 5). The nodes in the first layer decompress the video streams in a way that the decoding workload is distributed equally (so that given N views and M nodes, one node decodes N/M video streams). The number of nodes in the first layer should be chosen so that given the processing power of each node, we have enough nodes to decompress all the individual video streams in real time and transmit them to the second layer.

It is easy to see that in this case the processing power of the individual nodes can be chosen arbitrarily—as long as one can decode at least one video stream, the number and processing power of the nodes can be traded off. The second layer requests uncompressed pixels from the first layer through a high-speed network. As we have seen before,

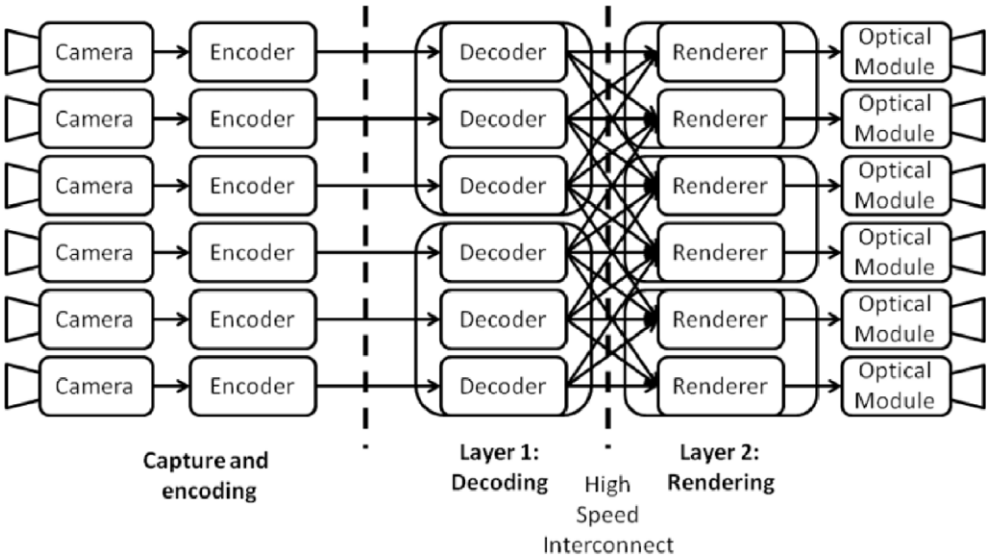

Figure 5. The two-layer decoder–renderer architecture. The first layer decodes video streams in parallel, while the second layer requests portions of uncompressed video data on demand.

the number of requested pixels can be maximum twice the number of pixels generated by a specific node. In case of driving 12x1280 × 800 resolution optical modules, this can be up to 1.41 Gbits/second, assuming 24 FPS, which is in the manageable range inside a rendering cluster. To avoid the collection and transfer of single pixels, we can transfer rectangular blocks which form the bounding box of the required pixels. Due to the compact shape of the required image regions, a bounding box does not add too much extra pixels to transmit, but has the benefit of being continuous in memory.

All rendering nodes require a different set of pixels; thus, each will request the transmission of different, slightly overlapping regions. As the regions are fixed if the camera settings, display parameters, and the mapping between them are fixed, the rendering nodes can subscribe to receive these fixed image regions on each new frame. The bandwidth required for transmitting and receiving the decompressed image regions can be scaled with appropriately choosing the number of processing nodes in both layers.

While the first layer can be connected to the data source using a network necessary to transmit the compressed video, the first and second layers are connected using a high-speed network to be able to carry the uncompressed image regions.

If the selected network architecture can take advantage of transferring large chunks of continuous in-memory blocks more efficiently than doing a large number of small transfers (for example InfiniBand RDMA),[38] then the images can be stored rotated with 90° in memory, so that full height vertical blocks become a continuous memory block, assuming the common line-by-line interleaved image storage format.

In order to reduce the network load between the first and second layer, it is possible to employ a slight, fixed ratio compression method that can be decompressed even when only partially transmitted, such as the S3TC/DXTn texture compression method that is available on virtually all GPUs.

One-Layer Architecture with Partial Decoding

The second proposed architecture does not require two processing layers, as decoding and light field conversion happens on the same processing nodes (see Figure 6).

In this case however, decoding needs to happen in real time regardless of the number of views to be decoded. To achieve that, we can take advantage of the fact that each renderer is using only portions of the views, as shown before. As we have seen before, the main bottleneck is decompression of all parts of all views, even those that are finally left unused in the specific rendering node. Therefore, assuming that we can save time by skipping the decoding of some views, and also skipping the decoding of image regions that are not needed, we can decode only those image regions which are necessary on the current rendering node. Using this approach, the decoding workload can be reduced with different codecs, if certain conditions are met.

This, however, poses unusual requirements on the image/video codec used for the transmission of the views. In the following subsections, we analyze common image/video codecs from this perspective: JPEG, JPEG2000, H.264, and HEVC. In the case of JPEG and JPEG2000, frames are encoded independently. In the case of H.264 and HEVC, we enable interframe compression.

In a horizontal-only-parallax light field display, the image regions are typically vertical. H.264, HEVC, and JPEG2000 support vertical slice shapes, while in the case of JPEG, which supports only horizontal subdivision, rotating the image before encoding is necessary.

The configuration used for all tests described below is a 6-core i7-5930K CPU@3.5 Ghz PC with a GeForce GTX 960 GPU with 2GB RAM, and all I/O performed on RAM disk.

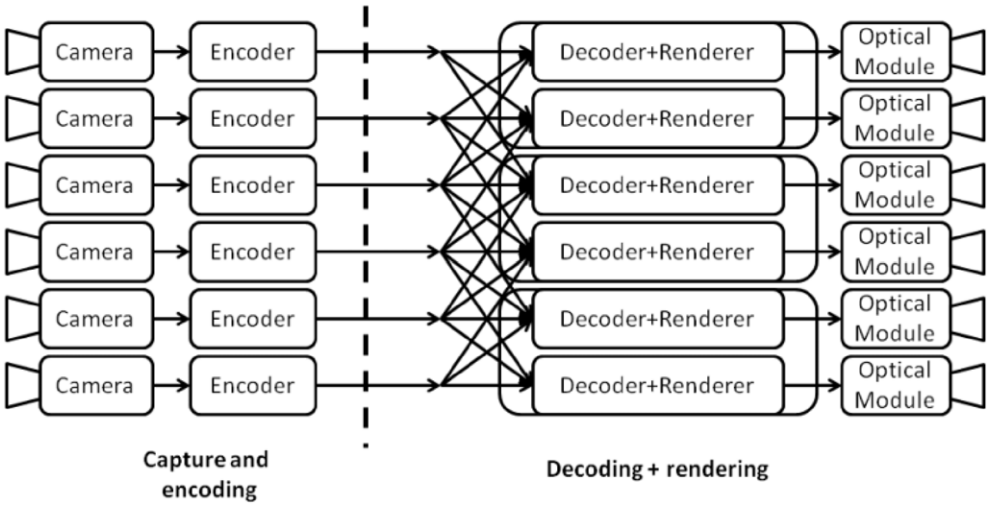

Figure 6. The one-layer decoder–renderer architecture. Decoders and renderers are running on the same nodes. Decoders decode those parts of the video that the renderer on the same node will need for rendering. No data exchange except frame synchronization takes place between the nodes.

Figure 7. A frame subdivided into horizontal H.264 slices. In an I-frame, these slices can be decoded independently.

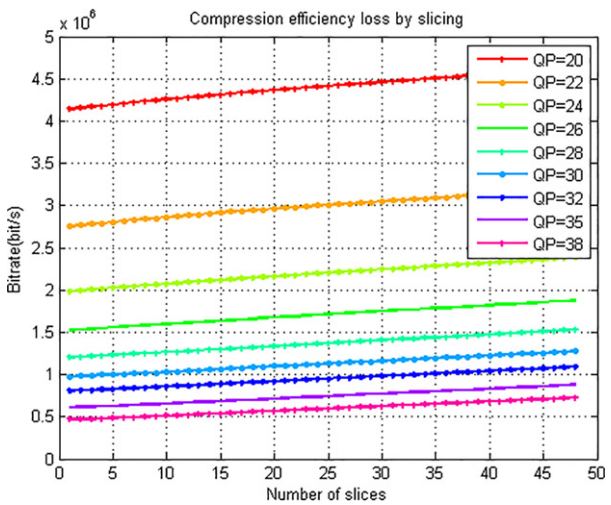

Figure 8. Bitrate increase caused by slicing a video frame into an increasing number of slices, shown for different QPs.

The experiments were performed on the Big Buck Bunny light field sequences,[27] more specifically the Flowers scene. Other content (i.e., Balloons sequence[39] in Figures 9 and 10) is used only for illustration purposes.

H.264

H.264[40] can partition a full frame into regions called slices, which are groups of macroblocks (please note these are different from light field slices). An encoder is commonly configured to use a single slice per frame, but can also use a specific number of slices (see Figure 7). This configuration may also specify the way macroblocks are partitioned into slices. Slices are self-contained in the sense they can be parsed from the bitstream without having the other slices. Slicing originally serves the purposes of providing error resilience and also to facilitate parallel processing. Slicing introduces some increase in bitrate, but this is minor compared to the overall bitrate, as seen in Figure 8.

To see how H.264's slicing feature can support partial decoding of video streams, we need to consider how it is implemented in the encoder and decoder. In H.264 we can differentiate Intraframes (I-frames) which are encoded independently from other frames in the video sequence, and hence employ prediction only inside the frame. Predicted frames (P-frames) and Bidirectionally predicted frames (B-frames) use image information of previously encoded/decoded frames in the encoder/decoder, exploiting similar blocks of pixels in subsequent frames moving across the image in time, typically representing a moving object.

When using multiple slices, encoders disable intraframe prediction across slice boundaries when encoding I-frames. Therefore, it is possible to decode only parts of an I-frame by decoding the respective slices and skipping the other slices (see Fig. 9). On the other hand, when performing interframe prediction in P-frames and B-frames, all encoders we have checked disregard slice boundaries, and perform motion prediction across slice boundaries. This means that the decoding process will also assume that the frame to be used for motion compensation is fully available in the

Figure 9. When using I-frame only encoding in H.264, dropping a slice has no effect on the remaining parts of the image. This image has been subdivided into three slices, and the middle slice dropped prior to decoding.

Figure 10. When using P- and B-frames motion vectors pointing out from the undecoded region (middle slice) propagate bogus colors into the slices which we intend to decode.

decoder. If, due to partial decoding of the previous frames, this condition is not met, the image regions that correspond to skipped slices will contain bogus color in the Decoded Picture Buffer.

Subsequently the motion compensation process will copy bogus colors from the buffer to the image being decoded, whenever a motion vector goes across the boundary of a missing slice (see Fig. 10). When such an erroneous decoded frame is used as a basis of motion compensation, the error further propagates from the undecoded regions as the decoder proceeds further in the stream. Only on the next I-frame the partially decoded image will be correct again, as the decoder does not use any other frames for decoding I-frames.

Therefore, skipping the decoding of slices is possible only if the encoder is instructed not to perform motion prediction across slice boundaries (see Figure 11). As this is not available as an option in any encoder we have evaluated, this functionality has been implemented by modifying the

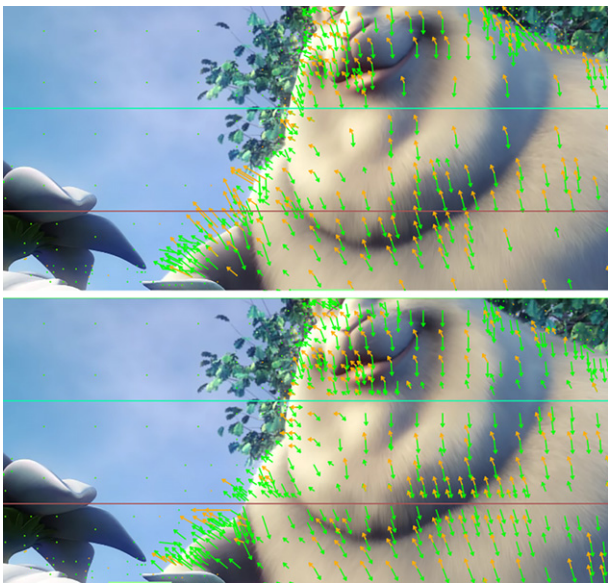

Figure 11. Difference of motion vectors in normal encoding and with self-contained slices. Notice that in the normal case (top) motion vectors cross slice/tile boundaries. In the self-contained case (bottom) no motion vectors cross the slice/tile boundaries.

reference encoder JM 18.6.[41] Our implementation is similar to that of Ref. 42, although serving a different purpose.

With restricting the motion vectors, we can achieve truly self-contained slices in the sense that decoding only selected slices becomes possible even for P- and B-frames, while still maintaining a standard conforming bitstream, though with a minor loss in coding efficiency due to partially restricting motion vectors. Implementation details have been described in our previous article.[43]

Once the encoded bitstream with self-contained slices is available at the decoder, there are several options to achieve partial decoding and thus saving decoding time. It is possible to modify the bitstream by dropping NAL units that correspond to the slices we do not wish to decode, like they were dropped by the network. While this results in a corrupt bitstream, some decoders (e.g., ffmpeg) can decode the remaining parts of the image, resulting in sublinear speedup, as shown in Figure 12. In this case, error resilience options have been disabled in the decoder to the extent possible; however, we suspect that the missing slices in the bitstream still result in some decoding time overhead. Another option is to modify the decoder to explicitly skip the decoding of specific slices, which requires decoder modification (as this functionality was also unavailable in all decoders we have tested).

HEVC

There are many new coding features introduced in HEVC[44] compared to H.264, and even a short summary of these novelties is out of scope in this article. Therefore, we only focus on differences relevant for our use case, and we refer the interested reader to Ref. 45 for a summary of other changes and novel features.

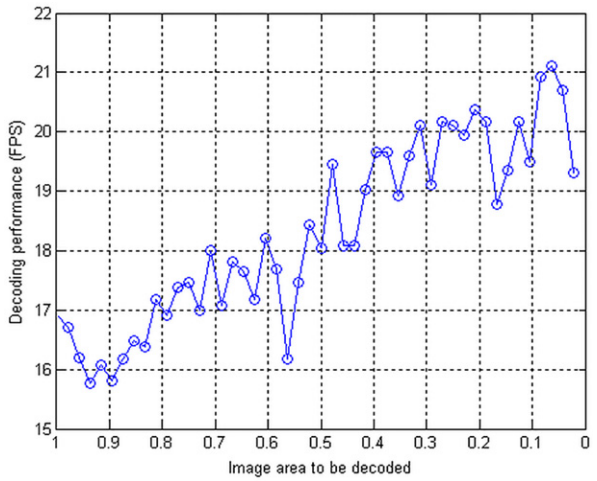

Figure 12. Speedup of ffmpeg H.264 decoder when performing partial decoding on videos sliced in 48 slices. The vertical axis shows frames per second values for decoding 80 views.

Tiles in HEVC are a new concept,[46] serving parallelization and packetization purposes. Tiles partition a video frame into a multiple rectangular partitions. Compared to slices, tiles provide more flexibility to partitioning and appear to incur less compression penalty since tiles do not contain any header. Furthermore, tiles are independent in terms of entropy coding, as the coder is reinitialized at the beginning of each tile.

In addition, intraframe prediction is limited within tile boundaries. In the HEVC interframe prediction scheme, however, the tile boundaries are disregarded. Similar to the H.264 case, motion search has to be restricted to remain inside tile boundaries to ensure partial decoding in P- and B-frames. Tiling is thus suitable for enabling partial decoding, and this has been implemented in the HTM 14.0 reference software,[47] alongside a tool for bitstream manipulation that removes selected tiles from the bitstream before decoding.[48]

As the coded tiles may be interleaved with other coded data in the bitstream and as parameter sets and headers (e.g., slice segment header) are for the entire bitstream, a dedicated decoding process is defined for decoding particular tiles, while omitting the decoding of other tiles. For that, a tile-based extractor is designed to construct an HEVC full-picture-compliant bitstream corresponding to the desired tiles such that a standard decoder can cope with it.

JPEG

Images stored in JPEG[49] files are typically subdivided into 8×8 blocks (Minimum Coding Unit, MCU). The sequence of these MCUs is basically coded in two parts: the DC component, which is stored as a difference from the previous block, as well as the AC components, which are stored as a sequence of values in a specific ordering, block by block. As all the values and coefficients are Huffman coded, one needs to read all Huffman codes to be able to interpret the ones that are actually needed. Some steps, however can be skipped

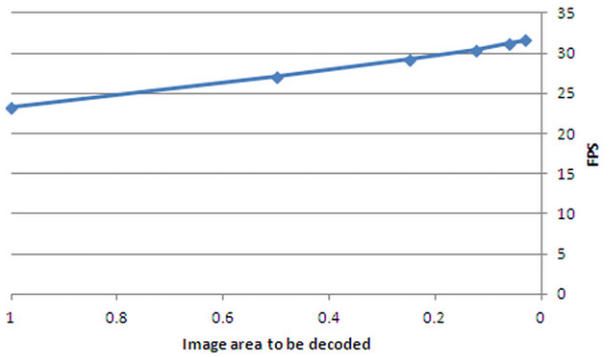

Figure 13. Speedup of the customized libjpeg-turbo when skipping all steps after Huffman decoding for the unnecessary image parts.

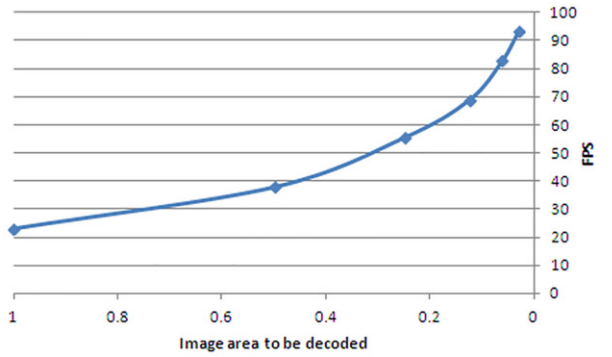

Figure 14. Speedup of the customized libjpeg-turbo when skipping through the bitstream using restart markers, also eliminating unnecessary Huffman decoding.

for the MCUs that are not needed, for example, inverse DCT, dequantization, color conversion, etc. Unfortunately, Huffman decoding takes the major part of JPEG decoding time, according to our profiling of libjpeg-turbo. When forcing the decoder to skip all the steps after Huffman decoding for the unnecessary image portions, one can only gain a rather insignificant speedup, as shown in Figure 13.

There is however an optional feature in JPEG to facilitate separately entropy coded segments, called Restart intervals/Restart markers, which allow resetting the bitstream after every N MCUs, letting the decoder skip N MCUs at a time without decoding anything, by just looking for specific bytes. If the JPEG encoder can be instructed to use a frequent restart interval, this feature can facilitate fast skipping of unnecessary MCUs.

We have implemented this approach too, by modifying the libjpeg-turbo decoder to skip the unused MCU rows by skipping through restart markers, as well as skipping all other processing steps for those MCUs. The speedup in this case is significant, as shown on Figure 14. For example, by decoding only the quarter of an image, a $2.4\times$ speedup can be gained. While this is still not linear, it is a nice addition to scalability.

The ordering of MCUs in the JPEG stream is fixed; therefore, restart markers can be only used to create horizontal strips that can be skipped. If, like in our use case, vertical strips are necessary, then the images need to

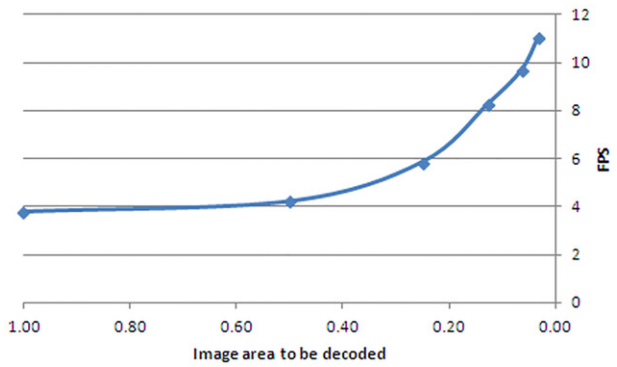

Figure 15. Speedup of Comprimato JPEG2000 CUDA-based codec when performing partial decoding of a single view. The vertical axis shows frames per second values for decoding 80 views.

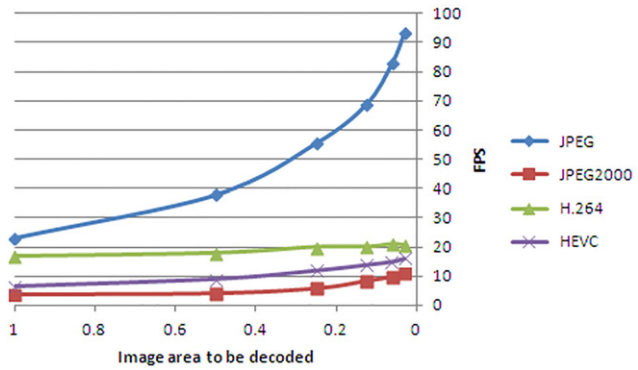

Figure 16. Comparison of overall speed and speedup of different decoders when decoding partial views. In case of JPEG, restart markers are used. In case of H.264, our custom self-contained slices are used. In case of JPEG2000 no special features are used.

be rotated 90° before encoding. Also, as mentioned earlier, the encoder needs to be set up to include restart markers at regular intervals, which might not be possible in every JPEG encoder (for example, JPEG capable cameras often have a compression quality settings, but no option for restart markers).

JPEG 2000

JPEG2000[50] images can also be partially decoded. This functionality has been recently introduced in the proprietary Comprimato JPEG2000 codec,[51] the CUDA-based version of which has been used for testing. Decoding 80 views takes 263 ms when decoding entire 1280 × 768 images, 8 bpp, 4:2:0 subsampling and default compression and decompression parameters, which accounts for 3.8 FPS operation taking only decoding into account.

On the other hand, when decoding only selected areas of each image, decoding time decreases significantly, although not strictly proportional with the image area, especially when decoding tiny image areas, as shown on Figure 15. Assuming that the application needs only the quarter of each image, a speedup of 54% can be observed, making decoding speed of 80 views 5.85 FPS. According to Comprimato, the fastest available GPUs at the time of writing can decode ∼2.37 times faster, which corresponds to almost 14 FPS. Thus, very soon (after gaining a factor of two speedup over current GPUs) we can expect this solution to work in real time for the benchmark use case.

The main advantage of using JPEG2000 for this purpose is that no special considerations are necessary during encoding, and the partial decompression feature is already built in the decoder as an option.

COMPARATIVE ANALYSIS

Architectures

The presented two-layer architecture is massively scalable, as the number of decoders, and the number of renderers can be chosen according to the performance of the individual nodes, while the bandwidth required by each rendering node is upper bounded by the number of pixels driven by each rendering node. Using this architecture, an arbitrary number of views with arbitrary resolution can be supported. On the other hand, the high-speed network necessary between the two layers can make this setup quite costly in a practical case, as the required bandwidth is more than what can be provided by a Gigabit Ethernet network. Any codec can be used that can be decoded by the decoding layer in real time.

The one-layer architecture requires less number of processing nodes (as the separate decoding layer is eliminated) and does not need a high-speed network to connect the two nodes. On the other hand, to achieve real-time performance special considerations have to be taken in the video codec used for feeding the incoming views. The suitability of different codecs for this architecture is analyzed next.

Codecs

Out initial aim was to find a solution for real-time decompression of the (partial) views necessary for light field conversion.

Therefore, our discussion mainly focused on the runtime performance of decoding, and its improvement when partial decoding is allowed. Here we directly compare the different solutions from this perspective. Also important is the bandwidth required by each codec to achieve same or similar image quality; therefore, a brief analysis of bandwidth requirements is presented.

As described before in detail, none of the evaluated codecs is capable of decoding image portions in a time proportional with the area to be decoded. While JPEG and JPEG 2000 show reasonable speedup, this speedup is not linear; therefore, full scalability cannot be guaranteed for any number of views or any resolution, as scalability is only partial. In the case of JPEG, this scalability can only be exploited if the encoder supports it, and is configured appropriately.

It is apparent from Figure 16 that JPEG is the clear winner when it comes to decoding performance, as even today, >24 FPS decoding can be achieved with using a publicly available codec with a modification to support partial decoding for the targeted use case.

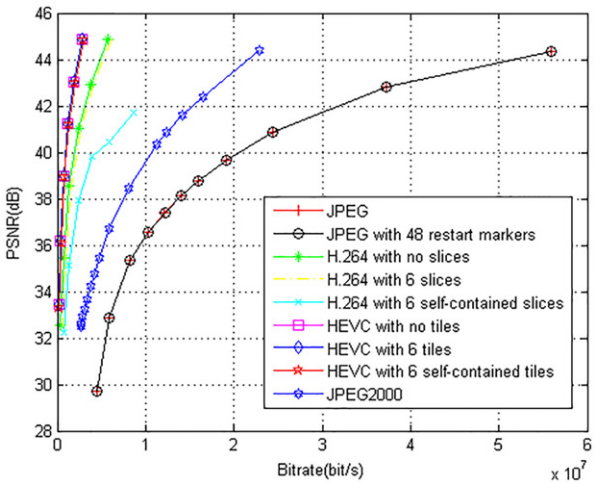

Figure 17. Comparison of overall quality versus bitrate of the different codecs using the configurations discussed in the article. Please note reported bitrates are for a single view. JPEG and JPEG with 48 restart markers overlap. The three HEVC curves also overlap.

It comes as no surprise that the decoding performance advantage of JPEG comes at a cost in bandwidth (see Figure 17). The bandwidth required by JPEG to achieve the same quality can easily be 10× higher than, for example, H.264. JPEG2000 is in between the two in terms of bandwidth consumption. Please note this comparison of bandwidth requirements is by no means meant as a comprehensive comparison of codec efficiency, but just a rough guideline to see the order of magnitude difference between the codecs under consideration. For a more general comparison of different codecs, the reader is referred to works such as Refs. 52–54.

In all cases except JPEG2000, partial decoding capability requires some kind of subdivision of the images into independent regions, which increases the necessary bitrate. In case of JPEG, the difference is negligible (<0.2%). In case of H.264 and HEVC, the difference is bigger due to the restriction of motion vectors.

When restricting motion vectors in H.264 and HEVC, the more independent areas are defined, the smaller these areas will become, being even more restrictive in the selection of motion vectors. Smaller independent regions however result in less pixels decoded unnecessarily, and thus a possible higher decoding speedup.

That is, the granularity of the subdivision determines the tradeoff between bitrate increase and the compactness of the bounding rectangle that needs to be decoded to gain access to an arbitrary image region.

Alternatives

To avoid using special codec features and encoder/decoder side modifications, one may also choose a very simple alternative solution: subdivide the source videos into vertical stripes of specific size, and encode them separately, in parallel. This approach has been used in Ref. 55, and the authors show good results with this approach for the use case of panoramic video. This however requires even more bitrate than the solutions outlined above, as well as a large number of video streams to be handled synchronously.

Advanced codecs targeting multiview and 3D video, such as MV-HEVC and 3D-HEVC[56] could be used as well, provided real-time decoders would exist, but unfortunately this is not the case at the time of writing.

Scalado RAJPEG[57] is said to enable instant random access/partial decoding to images. However, RAJPEG was not made available for testing.

CONCLUSION

We have shown two different approaches to introduce scalability in real-time image-based light field-based applications. While the two-layer approach can work with any codec (taking into account the necessary networking load), the one-layer architecture with partial decoding can only be achieved by modifying the way well known codecs typically work (except for the case of JPEG2000). While customization of codecs is possible to suit this requirement, the possibilities of forcing off-the-shelf hardware (for example, a camera capable of providing compressed output) to use this custom codec are rather limited.

The special use of video codecs as outlined above indicates that current video compression technology is lacking an important feature. Therefore, we have made steps[58,59] to ensure that next generation video technologies have low-overhead random access among their requirements, and this has been accepted in the MPEG FTV requirements document.[60] This work was done in the hope that next generation video codecs will support this use case natively.

ACKNOWLEDGMENTS

The authors are grateful to Attila Barsi of Holografika for his support in profiling the light field conversion process.

The research leading to these results has received funding from the PROLIGHT-IAPP Marie Curie Action of the People program of the European Union's Seventh Framework Programme, REA grant agreement 32449.

REFERENCES

[1] D. Ezra, G. J. Woodgate, B. A. Omar, N. S. Holliman, J. Harrold, and L. S. Shapiro, "New autostereoscopic display system," Proc. SPIE **2409**, 31–40 (1995).

[2] J. L. Fergason, S. D. Robinson, C. W. McLaughlin, B. Brown, A. Abileah, T. E. Baker, and P. J. Green, "An innovative beamsplitter-based stereoscopic/3D display design," Proc. SPIE **5664**, 488–494 (2005).

[3] D. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, and T. DeFanti, "The Varrier autostereoscopic virtual reality display," ACM Trans. Graph. **24**, 894–903 (2005).

[4] C. van Berkel, D. W. Parker, and A. R. Franklin, "Multiview 3D-LCD," Proc. SPIE **2653**, 32–39 (1996).

[5] G. E. Favalora, J. Napoli, D. M. Hall, R. K. Dorval, M. Giovinco, M. J. Richmond, and W. S. Chun, "100 Million-voxel volumetric display," Proc. SPIE **4712**, 300–312 (2002).

[6] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar, "Tensor displays, compressive light field synthesis using multilayer displays with directional backlighting," ACM Trans. Graph. **31**, 80 (2012).

[7] T. Balogh, "The HoloVizio system," Proc. SPIE **6055** (2006).
[8] M. Lucente, "Interactive three-dimensional holographic displays: seeing the future in depth," ACM SIGGRAPH Comput. Graph. **31**, 63–67 (1997).
[9] "Method & apparatus for displaying 3D images. by T. Balogh," US Patent 6,201,565, EP0900501 (1997).
[10] N. Inoue, M. Kawakita, and K. Yamamoto, "200-Inch glasses-free 3D display and electronic holography being developed at NICT," *Lasers and Electro-Optics Pacific Rim (CLEO-PR)* (IEEE, Kyoto, 2013), pp. 1–2.
[11] K. Nagano, A. Jones, J. Liu, J. Busch, X. Yu, M. Bolas, and P. Debevec, "An autostereoscopic projector array optimized for 3D facial display," *Proc. SIGGRAPH* (ACM, New York, NY, 2013), 2013, Article No. 3.
[12] E. Adelson and J. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*, edited by M. Landy and J. A. Movshon (MIT Press, Cambridge, MA, 1991).
[13] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," *Proc. SIGGRAPH '96* (ACM, New York, USA, 1996), pp. 43–54.
[14] M. Levoy and P. Hanrahan, "Light field rendering," *Proc. SIGGRAPH '96* (ACM, New York, USA, 1996), pp. 31–42.
[15] R. Bregović, P. T. Kovács, and A. Gotchev, "Optimization of light field display-camera configuration based on display properties in spectral domain," Opt. Express **24**, 3067–3088 (2016).
[16] D. Nguyen and J. Canny, "MultiView: spatially faithful group video conferencing," *Proc. SIGCHI Conf. on Human Factors in Computing Systems, Oregon* (ACM, New York, USA, 2005), pp. 799–808.
[17] J.-H. Lee, J. Park, D. Nam, S. Y. Choi, D.-S. Park, and C. Y. Kim, "Optimal projector configuration design for 300-mpixel light-field 3D display," Opt. Express **21**, 26820–26835 (2013).
[18] P. T. Kovács, Zs. Nagy, A. Barsi, V. K. Adhikarla, and R. Bregović, "Overview of the applicability of H.264/MVC for real-time light-field applications," *Proc. 3DTV Conf. 2014* (IEEE, Budapest, 2014).
[19] TileSlice Films [Online]. Available https://vimeo.com/timeslice/videos.
[20] A. Dricot, J. Jung, M. Cagnazzo, B. Pesquet, F. Dufaux, P. T. Kovacs, and V. Kiran, "Subjective evaluation of Super Multi-View compressed contents on high-end 3D displays," *Signal Processing: Image Communication* (Elsevier, 2015), Vol. 39, Part B, pp. 369–385.
[21] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," ACM Trans. Graph. **24**, 765–776 (2005).
[22] T. Fujii, K. Mori, K. Takeda, K. Mase, M. Tanimoto, and Y. Suenaga, "Multipoint Measuring System for Video and Sound—100-camera and microphone system," *Int'l. Conf. on Multimedia and Expo* (IEEE, Toronto, Ontario, 2006), pp. 437–440, DOI: http://dx.doi.org/10.1109/ICME.2006.262566.
[23] T. Balogh and P. T. Kovács, "Real-time 3D light field transmission," Proc. SPIE **7724** (2010).
[24] A. Jones, J. Unger, K. Nagano, J. Busch, X. Yu, H.-Y. Peng, O. Alexander, and P. Debevec, "Creating a life-sized automultiscopic Morgan Spurlock for CNNs 'Inside Man'," *Proc. SIGGRAPH '14, no. 2* (ACM, New York, USA, 2014).
[25] C. Kim, H. Zimmer, Y. Pritch, A. Sorkine-Hornung, and M. Gross, "Scene reconstruction from high spatio-angular resolution light fields," ACM Trans. Graph **32**, 73 (2013).
[26] A. Jones, I. McDowall, H. Yamada, M. Bolas, and P. Debevec, "Rendering for an interactive 360° light field display," ACM Trans. Graph. **26**, 40 (2007).
[27] P. T. Kovács, A. Fekete, K. Lackner, V. K. Adhikarla, A. Zare, and T. Balogh, "Big Buck Bunny light-field test sequences," ISO/IEC JTC1/SC29/WG11/M36500, Warsaw, 2015.
[28] P. Goorts, M. Javadi, S. Rogmans, and G. Lafruit, "San Miguel test images with depth ground truth," ISO/IEC JTC1/SC29/WG11/M33163, Valencia, 2014.
[29] F. Zilly, C. Riechert, M. Müller, P. Eisert, T. Sikora, and P. Kauff, "Real-time generation of multi-view video plus depth content using mixed narrow and wide baseline," J. Vis. Commun. Image Represent. **25**, 632–648 (2014).
[30] A. Ouazan, P. T. Kovacs, T. Balogh, and A. Barsi, "Rendering multi-view plus depth data on light-field displays," *Proc. 3DTV Conf. 2011* (IEEE, Antalya, 2011).
[31] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," Proc. SPIE **5291** (2004).
[32] N. Stefanoski, O. Wang, M. Lang, P. Greisen, S. Heinzle, and A. Smolic, "Automatic view synthesis by image-domain-warping," IEEE Trans. Image Process. **22**, 3329–3341 (2013).
[33] M. Solh and G. AlRegib, "Hierarchical hole-filling for depth-based view synthesis in FTV and 3D video," IEEE J. Sel. Top. Signal Processing **6**, 495–504 (2012).
[34] C. Guillemot and O. L. Meur, "Image inpainting: overview and recent advances," IEEE Signal Process. Mag. **31**, 127–144 (2014).
[35] libjpeg-turbo [Online]. Available http://libjpeg-turbo.virtualgl.org/.
[36] x264[Online]. Available http://www.videolan.org/developers/x264.html.
[37] FFmpeg [Online]. Available https://www.ffmpeg.org/.
[38] Introduction to InfiniBand™ for End Users—Mellanox [Online]. Available http://www.mellanox.com/pdf/whitepapers/Intro_to_IB_for_End_Users.pdf.
[39] M. Tanimoto, N. Fukushima, T. Fujii, H. Furihata, M. Wildeboer, and M. P. Tehrani, "Moving multiview camera test sequences for MPEG-FTV," ISO/IEC JTC1/SC29/WG11/ M16922, Xian, China, 2009.
[40] Information technology—Coding of audio-visual objects—Part 10: Advanced Video Coding, ISO/IEC 14496-10, 2003.
[41] H.264/AVC Software Coordination [Online]. Available http://iphome.hhi.de/suehring/tml/.
[42] P. Quax, F. Di Fiore, P. Issaris, W. Lamotte, and F. Van Reeth, "Practical and Scalable Transmission of Segmented Video Sequences to Multiple Players using H.264," *Motion in Games 2009 (MIG09)*, Lecture Notes in Computer Science LNCS Series, LNCS 5884 (Springer, Zeist, the Netherlands, 2009), pp. 256–267.
[43] A. Zare, P. T. Kovács, and A. Gotchev, "Self-Contained Slices in H.264 for Partial Video Decoding Targeting 3D Light-Field Displays," *Proc. 3DTV Conf.* (IEEE, Lisbon, 2015).
[44] Information technology—High efficiency coding and media delivery in heterogeneous environments—Part 2: High efficiency video coding, ISO/IEC 23008-2, 2013.
[45] G. J. Sullivan, J. Ohm, H. Woo-Jin, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Trans. Circuits Syst. Video Technol. **22**, 1649–1668 (2012).
[46] K. M. Misra, C. A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles," IEEE J. Sel. Top. Signal Process. **7**, 969–977 (2013).
[47] HEVC Test Model [Online]. Available https://hevc.hhi.fraunhofer.de/HM-doc/.
[48] A. Zare, P. T. Kovács, A. Aminlou, M. M. Hannuksela, and A. Gotchev, "Decoding complexity reduction in projection-based light-field 3D displays using self-contained HEVC tiles," *Proc. 3DTV Conf. 2016* (IEEE, Hamburg, 2016).
[49] Digital compression and coding of continuous-tone still images, ISO/IEC 10918-1, 1994.
[50] JPEG2000 image coding system, ISO/IEC 15444, 2000.
[51] Comprimato JPEG2000 encoder and decoder [Online]. Available http://www.comprimato.com/.
[52] S. Grgic, M. Mrak, and M. Grgic, "Comparison of JPEG Image Coders," *Proc. 3rd Int'l. Symposium on Video Processing and Multimedia Communications* (IEEE, 2001), pp. 79–85.
[53] A. Al, B. P. Rao, S. S. Kudva, S. Babu, D. Sumam, and A. V. Rao, "Quality and complexity comparison of H.264 intra mode with JPEG2000 and JPEG," *Image Processing, 2004, Int'l. Conf. on ICIP '04. 2004* (IEEE, 2004), Vol. 1, pp. 525–528.
[54] M. T. Pourazad, C. Doutre, M. Azimi, and P. Nasiopoulos, "HEVC: The New Gold Standard for Video Compression: How Does HEVC Compare with H.264/AVC?," *IEEE Consumer Electronics Magazine* (IEEE, 2012), Vol. 1, pp. 36–46.

[55] P. Quax, P. Issaris, W. Vanmontfort, and W. Lamotte, "Evaluation of distribution of panoramic video sequences in the eXplorative television project," *Proc. 22nd Int'l. Workshop on Network and Operating System Support for Digital Audio and Video* (ACM, New York, USA, 2012), pp. 45–50.

[56] G. Tech, Y. Chen, K. Müller, J.-R. Ohm, A. Vetro, and Y.-K. Wang, "Overview of the Multiview and 3D Extensions of High Efficiency Video Coding," IEEE Trans. Circuits Syst. Video Technol. **26**, 35–49 (2015).

[57] RAJPEG Technology – Scalado [Online]. Available http://www.scalado.com/display/en/RAJPEG+Technology.

[58] P. T. Kovács, T. Balogh, J. Konieczny, and G. Cordara, "Requirements of Light-field 3D Video Coding," ISO/IEC JTC1/SC29/WG11/M31954, (San Jose, 2014).

[59] P. T. Kovács, Z. Nagy, A. Barsi, V. K. Adhikarla, and R. Bregovic, "Proposal for additional features and future research to support light-field video compression," ISO/IEC JTC1/SC29/WG11/M37434, (Geneva, Switzerland, 2015).

[60] S. Shimizu, G. Bang, T. Senoh, M. P. Tehrani, A. Vetro, K. Wegner, G. Lafruit, and M. Tanimoto, "Use Cases and Requirements on Free-viewpoint Television (FTV) v.2," ISO/IEC JTC1/SC29/WG11/N15732, (Geneva, 2015).