ABSTRACT

We present a new hierarchical compression scheme for encoding light field images (LFI) that is suitable for interactive rendering. Our method (RLFC) exploits redundancies in the light field images by constructing a tree structure. The top level (root) of the tree captures the common high-level details across the LFI, and other levels (children) of the tree capture specific low-level details of the LFI. Our decompressing algorithm corresponds to tree traversal operations and gathers the values stored at different levels of the tree. Furthermore, we use bounded integer sequence encoding which provides random access and fast hardware decoding for compressing the blocks of children of the tree. We have evaluated our method for 4D two-plane parameterized light fields. The compression rates vary from 0.08 − 2.5 bits per pixel (bpp), resulting in compression ratios of around 200:1 to 20:1 for a PSNR quality of 40 to 50 dB. The decompression times for decoding the blocks of LFI are 1 − 3 microseconds per channel on an NVIDIA GTX-960 and we can render new views with a resolution of 512 × 512 at 200 fps. Our overall scheme is simple to implement and involves only bit manipulations and integer arithmetic operations.1

CCS CONCEPTS

• Computing methodologies → Image-based rendering; Image compression; Graphics processors; Graphics file formats; Virtual reality.

ACM Reference Format:

1 INTRODUCTION

Light fields create a photo-realistic rendering of extremely complex scenes that are difficult to achieve with other conventional rendering techniques. The photo-realistic rendering from a light field (LF) makes virtual reality (VR) content more immersive and improves the sense of presence in real world-scenes. Levoy and Hanrahan 1996 and Gortler et al. 1996 describe LF rendering methods by capturing a scene using a camera grid constructed using camera arrays. This has been an active area of research for more than two decades and many improved methods for capturing and rendering have been proposed. Several hand held plenoptic cameras have been developed to capture the high stereo light fields of real

1website: http://gamma.cs.unc.edu/LFC/
scenes [Ng et al. 2005; Perwass and Wietzke 2012] and to use for VR applications [Yu 2017].

One of the primary challenges in using image-based rendering (IBR) approaches based on LFI is the amount of data needed to capture the 3D scenes. Such IBR techniques generate a lot of images to sample the light rays for a given scene. Typically, the size of the uncompressed light field images can vary from 200MB to 10GB (or more) and can be even larger depending on the sampling rate and image resolution. Several methods and hardware techniques for interactive rendering of light fields have been proposed [Chen et al. 2002; Jones et al. 2007]. The need to use high resolution light fields has increased with recent demand for high resolution multimedia content (2K or 4K resolution). An efficient way to capture high resolution panoramic light fields is discussed in Birkbauer et al. [2013].

In order to store, transmit, and render LFI, it is important to develop good compression algorithms. Different schemes have been proposed for compressing LFI, as surveyed in [Viola et al. 2017]. The majority of these methods provide high compression rates, like standard 2D image compression methods, but they require decoding all the LFI samples into memory before rendering. During rendering, any of the sampled light rays (pixels) from the LFI may be used for computing new views. For interactive rendering in VR and mobile devices, it is necessary for the compressed LFI bitstream to have random access capabilities. Random access to the compressed LFI bitstream can reduce the memory footprint during rendering by a significant factor.

**Main Results:** We present a new hierarchical compression scheme (RLFC) for encoding light field images. The primary application of our method is for interactive rendering in VR and mobile devices, which require low latency and low memory footprint. Our method is based on clustering spatially close sampled images of light fields and constructing a tree. After the construction process is done, the root of the tree stores the common features or characteristics among the LFI, the rest of the children nodes of the tree store the specific low-level or high-frequency details of the LFI. In our tree construction process we compute two types of new images; images that capture the common details (redundancies) among the LFI are referred to as representative key views (RKVs) and the images that capture high-frequency details are referred to as sparse residual views (SRVs). We start the construction of the tree in a bottom-up manner computing multiple levels of RKVs and at the start of the building process the bottom level of the tree is initialized with the original LFI. Next, we proceed in a top-down manner and compute SRVs between alternate levels of RKVs in the tree. Once the tree is computed, only the top-level RKVs and the SRVs of the tree are stored.

The top-level RKVs are compressed using standard image compression techniques (e.g., JPEG2000). The SRVs correspond to low-level details of the LFI and they are sparse in terms of the features captured. At all levels, the SRVs are divided into blocks and only the blocks with significant details are stored. Moreover, significant blocks of the SRVs at each level of the tree are encoded using Bounded Sequence Integer Encoding (BISE) [Nystad et al. 2012]. The resulting compressed bitstream is represented to support selective decoding and provides random access to blocks of pixels with only one level of indirection. We have evaluated our method on the two-plane parameterization of light fields and present the results on the Stanford light field archive. The decoding time to decompress a block of pixels from our compressed stream is 1-3 microseconds on an NVIDIA GTX-960 GPU and RLFC can be used to render new views with a resolution of 512 × 512 at 200 fps. We obtain compression ratios of around 200 : 1 to 20 : 1 for a PSNR quality of 40 – 50 dB.

The remainder of the paper is organized as follows: Section 2 gives an overview of prior work in light field rendering and compression. Section 3 gives details about compression, decompression, and interactive rendering. In Section 4, we present a detailed analysis of our compression scheme and highlight the results on various benchmarks.

## 2 BACKGROUND

In this section, we present a brief overview of light field rendering and compression algorithms.

### 2.1 Light Field Rendering

The intensity of light rays through empty space remains constant. Levoy and Hanrahan [1996] use this observation to present a simple two-plane (4D) parametrization of the plenoptic function and describe a practical light field rendering scheme for real-time photo-realistic rendering of complex objects. In this two-plane parameterization, all the light rays between parallel planes are described using a pair of parameters (u, v) and (s, t). The light rays between the two planes are called light slab and the 4D parametrized plenoptic function is called light field (LF). Gortler et al. [1996] presents a different representation of the 4D parameterization called a lumigraph. In a lumigraph, a cube is used instead of two parallel planes to create a bound over a particular region of interest in space. The light rays in any light field parameterization [Davis et al. 2012; Ihm et al. 1997] are captured using a very large set of discretely sampled camera images. This large amount of data required creates a practical bottleneck for capturing LFs and for rendering methods.

### 2.2 Light Field Compression

The minimum sampling rate required for a good reconstruction using IBR is a well-studied problem [Chai et al. 2000; Chan and Shum 2000]. Even with a minimal sampling rate, the number of image samples required can be in the order of thousands for a good-quality reconstruction [Chai et al. 2000]. Many schemes have been proposed for LFI compression and we categorize the existing compression schemes into two categories: high efficiency encoding schemes, which include methods similar to standard image and video coding techniques (DCT, wavelet); and random access compression schemes, which include methods suitable for fast viewing and rendering as they provide fast random access to specific pixels, but provide lower compression ratios. In response to the growing interest in quality plenoptic content, the JPEG standardization committee launched JPEG Pleno [Ebrahimi et al. 2016]. The aim of JPEG Pleno is to define standards for the wide adaptability of 4D LF compression, like JPEG and MPEG standards.

#### 2.2.1 High efficiency LFI compression schemes

Earlier work on LFI compression is based on extending standard image and video...
coding methods (JPEG, JPEG2000, MPEG-2, MPEG-4). These methods employ techniques such as discrete-cosine transform, wavelet transform, predictive block encoding, and motion-vector compensation. Girod et al. [2003]; Jagmohan et al. [2003]; Magnor and Girod [2000] use disparity compensation instead of motion vectors for predictive coding of blocks of LFI. In disparity compensated approaches, a pre-fixed set of LF images is encoded independently (I-frames) and the rest of the LF image blocks are encoded predictively (P-frames) from the I-frame blocks. Due to the uniform camera motion in the sampling of LF in two plane parameterization, P-frame blocks can be predicted from I-frame blocks using a single disparity value. The compression rates of these methods are around 100:1 to 200:1, depending on the details in the original LFI. Kundu [2012] uses homography techniques to predictively encode LF images (P-frames) by warping them onto a set of I-frames, achieving compression rates of 10:1 to 50:1. Chang et al. [2006] use techniques that are based on using additional shape and geometry information about the object captured in the LF images. More recently, methods directly based on HEVC video coding have been proposed by Chen et al. [2018]; Liu et al. [2016]; Perra and Assuncao [2016] and they can achieve high compression rates of 100:1 to 1000:1. Liu et al. [2016] order the LF-Images using a pseudo-sequence temporal ordering and compress them using HEVC encoding. Chen et al. [2018] use a small set of views to predict the rest of the images using disparity based image-transformations and combine it with a pseudo-sequence method [Liu et al. 2016].

2.2.2 Random Access LFI compression: Levoy and Hanrahan [1996] present a compression technique using vector-quantization (VQ) that provides random access for interactive rendering. VQ results in compression rates of around 10:1 to 20:1, but the compression quality is low. Moreover, VQ-based compression fails to take advantage of the high correlation or coherence between LF images. Peter and Strafer [2001] describe an approach for random-access compression using a 4D wavelet hierarchical scheme, which provides compression rates of 20:1 to 40:1. However, this method requires multi-level caches for fast data access during rendering. This approach also makes assumptions about the scene captured in the light fields. Zhang and Li [2000] describe a technique that is similar to the high-efficiency compression schemes using multi-reference frame-based motion compensation and achieves compression ratios of 80:1. Overbeck et al. [2018] present a new end-to-end system for capturing and rendering very high resolution light fields with a new capturing system, a new spherical parameterization, and a new rendering approach for light fields. Overbeck et al. [2018] present a compression scheme using motion compensated prediction by modifying the VP9 video codec [Mukherjee et al. 2013] to allow for more reference frames and provide random access. They achieve compression rates of 40:1 to 200:1 for a high quality compression. Koniaris et al. [2017] describe an end-to-end system for rendering animated light fields. They present a temporal compression scheme that uses interpolation between the frames (in time dimension) to encode the animated light fields. On top of temporal compression, they utilize standard texture compression methods (DXT, BC6, BC7) to achieve more gains in compression. Overall they achieve compression ratios of 60:1 to 500:1 (temporal and spatial) with a spatial compression ratio of 6:1.

3 OUR METHOD: RLFC

In this section, we present an overview of our compression pipeline (Figure 2) and the details of our encoding method. The input to our method is sampled light field images of a scene and the output is a compressed stream that provides parallel decoding capabilities and random access. Our approach has the following components:

Representative Key Views (RKVs): At each level of the tree, these 2D images capture the redundancies present in the images in the level below it. In the final stream, only the top-level RKVs are stored.

Sparse Residual Views (SRVs): These 2D images are stored at each level and capture the specific details of the images in the current level.

Clusters (C): Clustering is performed to gather the LF samples (2D-images sampled while capturing the LF) that are spatially close...
to one another (localized cluster computation) and which exhibit coherency. We cluster the RKVs that are close one another at each level of the tree. Samples refer to the 2D images sampled while capturing the light field.

**Filtering:** Filtering is performed to compute a single representative view that captures the redundancies of the samples in a cluster. The representative key view at the next level is computed for samples that are close to one another using a weighted filtering of the samples in each cluster.

**Residue Computation, Thresholding, & Quantization:** The residue computation is performed to compute the SRV in a top-down fashion by taking the difference between alternate levels of RKVs, followed by thresholding and quantization of the SRVs at that level. The thresholding step is performed to remove the insignificant residual blocks in SRVs. The insignificant blocks are determined by a threshold set as an encoding parameter. The block size is also set as an encoding parameter. After thresholding the pixel values in the significant blocks are quantized to reduce the dynamic range of the pixel values in SRVs.

**JPEG2000:** All the top-level RKVs are compressed in a lossless mode and we use the JPEG2000 algorithm.

**BISE:** Bounded Integer Sequence Encoding (BISE) is a method used to encode a sequence of integer values in different ranges in an efficient hardware-friendly manner. It is used for texture compression and we use BISE to encode the integer values in the quantized SRVs blocks. It provides good compression rates and is well supported on current GPUs.

We construct a tree with the root node storing representative key views (RKVs) and the children nodes storing the sparse residual views (SRVs). Our construction process starts by setting the input indexed LFI as the bottom level (zero level) of the tree and then we recursively construct higher-level RKVs starting from the bottom level to the top level. At any given level, we cluster the RKV images that are spatially close; for all the clusters, we compute the RKVs of the next level of the tree. After the RKV tree is computed, we compute the SRVs for each level from top to bottom, reconstructing the image samples in an LF exhibit strong spatial correlation and we exploit this to find redundancies and compress the data. The first step is to cluster samples that are close to each other using the number of clusters and the number of images in each cluster set as an encoding parameters for all the levels:

\[
C^{l-1}_j = \bigcup_{k \in \{k_1, k_2, \ldots, k_n\} \mid \text{dist}(k_u, k_v) < \text{threshold}} RKV^{l-1}_k
\]

where \( k \) denotes the index RKV image on level \( l - 1 \). \( \text{dist}(k_u, k_v) \) computes the spatial distance between \( RKV_{k_u} \) and \( RKV_{k_v} \) at current level. Spatial distance corresponds to the Euclidean distance between the spatial locations (relative or absolute real-world) of the LFI provided in the input. For each cluster of image samples, an RKV image is computed. The RKV image is computed using a weighted filtering of all the images in a given cluster. Let \( (C_j)^{(l-1)} \) be the \( j^{th} \) cluster on level \( l - 1 \), \( I \) denote an RKV in the current cluster and the \((RKV^{(l-1)}_j)^I\) on level \( l \) is computed as:

\[
RKV^{(l-1)}_j = \sum_{I \in (C_j)^{(l-1)}} w^{(l-1)}_j \times I.
\]

\( I \) denotes the images in the cluster \( C^{l-1}_j \), \( w^{(l-1)}_j \) is the weight for image \( I \) for filtering images in the cluster \( C^{l-1}_j \). The new set of RKV images computed at a level \( l \) exhibit spatial correlations similar to the RKV images at the level below \( l - 1 \). This process is recursively repeated until a certain number of levels of the tree is computed. The number of levels is one of the encoding parameters. This process generates hierarchical levels of RKVs, which results in...
3.2 Sparse Residual Views

Once the RKV tree is constructed, the SRV tree is computed in a top-down manner, starting with the top-level of the RKVs tree. The $\text{SRV}^{(l-1)}$ at level $(l-1)$ is computed by subtracting the parent $(\text{RKV}_{\text{p}})^{l}$ image at level $l$ from the corresponding children $(\text{RKV})^{l-1}$ at level $l-1$:

$$\text{SRV}_{j}^{(l-1)} = (\text{RKV}_{j}^{(l)})^{l-1} - (\text{RKV}_{\text{p}})^{l}.$$  

The SRV images tend to be quite sparse and consist of regions of high intensity values. Figure 4 shows an example RKV and SRV computed using our approach. It can be seen that the SRV image is mostly empty with zero or small intensity and only certain regions have high intensity pixel values. Based on this observation, we divide all the levels of the SRV images into non-overlapping rectangular blocks and only store the blocks with high intensity pixel values. The SRV images are processed using a thresholding step followed by a quantization step. We process the SRV images with two levels of thresholding, pixel level thresholding and block energy thresholding, which are based on thresholds set as encoding parameters. The pixel level thresholding determines whether pixel values in a SRV image provides a significant contribution. If pixel values are below the set threshold the pixel is considered insignificant and the value is set to zero. The block energy based thresholding computes the sum of the absolute pixel values in a block and decides whether a block has block energy above the set threshold. If the block energy is below the set threshold, the block is considered insignificant and discarded. After thresholding, the dynamic range of the pixel values in the significant blocks is reduced by quantizing the pixel values. More details on the quantization are presented in

Figure 4: (a) Example of a representative key view (RKV) and (b) a sparse residual view image (SRV). We can see that the residual view image is very sparse with large intensity pixel values constrained in small regions. The images are generated from the Lego Knights LF from the Stanford light field archive.

The arrangement of the SRV tree in the final compressed stream. All the blocks at the same particular spatial location are gathered from all the nodes of the SRV tree. The BISE encoded streams of the gathered blocks are appended to the compressed stream in the serial order computed using BFS traversal. For each block, the start locations of the compressed stream are stored in a block offset array, shown on the right.

Figure 5: The suppl. material, Section-3 2. Once the $\text{SRV}^{(l-1)}$ at level $(l-1)$ is computed, the $\text{RKV}^{(l-1)}$ at the corresponding level is recomputed. We recompute $\text{RKV}^{(l-1)}$ so that the thresholding and quantization errors do not propagate down the tree to the levels below. Once the downward pass is finished, the RKVs at all levels are discarded except for the top level (n) $\text{RKV}^{n}$. In the end we are left with a tree with the root as the top-level $\text{RKV}^{n}$ and the children nodes are the sparse quantized SRV levels. The $\text{RKV}^{n}$ at the root of the tree is compressed using standard image compression techniques (JPEG2000). An example visualization of the computed trees (SRV images and RKV images) is presented in suppl. material, Section-5.

3.3 Bounded Integer Sequence Encoding

BISE was first introduced and used in the ASTC texture compression format by Nystad et al. [2012]. For a set of integer values that lie in the range 0 to $N - 1$ with equal probability, BISE addresses the problem of encoding them efficiently while allowing constant time decoding with a very limited hardware. Apart from trivial case of storing $\log_2 N$ bits, when $N$ is a power of two, BISE describes an efficient packing method for different ranges of $N$ as well. We use BISE to encode the SRV blocks for two reasons: (1) Traditional techniques like DCT or wavelet transform separate a given signal into high-frequency coefficients and low-frequency coefficients. We observe that SRVs are high-frequency signal images and that using traditional techniques does not work well (suppl. material, Section-1). (2) BISE provides good compression rates with hardware supported decoding available on most desktop and mobile commodity GPUs.

3.4 Compressed Stream Structure

We arrange the final compressed stream to enable progressive and random access decoding of the pixels. To create a compressed stream, the final tree (top-level $\text{RKV}^{n}$ and SRV) must be linearized. The first step in creating our compressed stream structure is assigning a serial indexing (linearization) to all the nodes in the final tree. We traverse the final tree from the root using breadth first search (BFS), indexing all the nodes in the order they are traversed by BFS.

After the SRV nodes are linearly ordered using BFS, all the blocks of each SRV node are assigned the same serial index as the node.
Algorithm 1: Compress light field image samples

<table>
<thead>
<tr>
<th><strong>Input:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original indexed light field images: LFI</td>
</tr>
<tr>
<td>Encoding parameters: enc</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>Compressed stream</td>
</tr>
</tbody>
</table>

```algorithm
def COMPRESSLFI(LFI, enc):
    RKV⁰ ← LFI
    RKV ← ComputeRKVTREE(RKV⁰, enc)
    SRV ← ComputeSRVTREE(RKV, enc)

    RKVStream ← JPEG2000(RKV⁰)
    SRVStream ← ReadSRVTREE(SRV)
    // Initialize the SRV tree compressed bitstream to empty
    SRVStream ← 0
    BlockOffsets ← { }

    for each block in blocks do
        // Travel the SRV tree in a level order fashion to gather the blocks
        BFS(srv, index)
        // Encode the blocks of SRV levels using BISE
        OrderedBlocksBISE ← BISEEncode(LvlBlocks)
        BlockOffsets ← Length(OrderedBlocksBISE)
        SRVStream ← SRVStream : OrderedBlockBISE
        // Append the streams and return the final stream
    return (RKVStream : BlockOffsets : SRVStream)
```

BISE encoded blocks in the same spatial location are gathered from all the nodes in the SRV tree and appended to the compressed stream in the corresponding serial order from BFS. In this manner, all the BISE encoded blocks of the SRV are processed starting with the block on the top-left and ending with the block on the bottom-right. An array of block offset values to the current block’s compressed stream is stored to facilitate parallel and random-access decoding. Figure 5 highlights our compressed representation.

### 3.5 Decompression & Random Access

The first step in decompression is decoding all the top-level RKV images. To decode a block of pixels at a particular location, we traverse the SRV tree and gather the BISE streams of all the corresponding blocks until the process reaches the bottom level. Next, the residual pixel values are gathered by decoding the BISE streams. Finally, the pixel values are computed by combining the residual pixel values with the corresponding top-level RKV image. While rendering, only a small set of pixels from a different LFI is required by the renderer. Using the tree traversal operation, we selectively gather and decode only the blocks required by the LF renderer. With selective decoding, our compressed stream stays the same in the memory. Only small parts of the compressed stream are decoded to get the pixel values required by the LF renderer.

Given a pixel’s location, we can compute the individual block location and the location of all parent-child blocks in the compressed stream by running the same BFS used during compression as an initialization step of decompression. The start location of the current block’s compressed stream is located using the block offset values. Using the tree traversal decompression, our approach provides random access to the pixel values at block level without decompressing the other parts of the compressed data.

### 3.6 Interactive Rendering

Our decompression scheme is designed to be compatible with several efficient IBR techniques [Isaksen et al. 2000; Levoy and Hanrahan 1996] because it supports block-based parallel and progressive decompression. It is evident from the arrangement of the compressed stream (Figure 5) that our method supports parallel decoding of multiple blocks at once. Our method inherently supports progressive decompression because the RKV tree constructed during the compression has filtered images of spatially close by LF images at different levels. For progressive decompression we can modify the tree traversal operation to stop and compute the high-level filtered pixel value at a certain level. The rendering scheme can progressively update the pixel values, as the decompressor computes the final pixel values in the background. Parallel and progressive decompression can significantly improve the rendering rate at runtime. The decompression scheme is also hardware friendly because the decoding operations consist of simple integer arithmetic operations.

During rendering, the pixel values of the new views are computed by interpolating several surrounding pixel values from several

<table>
<thead>
<tr>
<th><strong>Input:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>LFI compressed stream: CompLFI</td>
</tr>
<tr>
<td>Image index: ImgIdx</td>
</tr>
<tr>
<td>Block index: BlkIdx</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>Pixel values: PixVals</td>
</tr>
</tbody>
</table>

```algorithm
def DECOMPRESSLFI(ImgIdx, BlkIdx):
    // Get the start location of BlkIdx in bitstream
    StartOffset ← BlockOffsets[BlkIdx]
    // Read the top level filtered values from RKV
    RKVBLOCK ← ReadBlock(RKV⁰, BlkIdx)
    // Compute the location of the parent blocks in stream
    ParentIdx ← GetParentIndices(ImgIdx)
    // Read the BISE encoded stream of required blocks
    OrderedBlocksBise ← ReadBlocks(SRVStream, ParentIdx)
    // Decode the BISE blocks
    OrderedBlocks ← BISEDecode(OrderedBlocksBise)
    // Combine the residual values with filtered pixel values
    CombPixVals ← CombineBlocks(RKVBLOCK, OrderedBlocks)
    return CombPixVals
```
LF samples. Our method inherently supports the efficient reconstruction of new views by interpolating multiple samples. When we decode a pixel value, an entire block is decoded, which also provides surrounding pixel values. Therefore, while generating new views, a set of spatially nearby blocks from the LF samples is decoded. The intensity of the new pixel value \((u, v, s, t)\) ray in 4D parameterization) is computed by quadrilinear or bilinear interpolation of several pixel values from the decoded blocks.

During interactive rendering, the renderer requests pixels from several locations of the light field images. The GPU decomposition algorithm decodes blocks of the LFI in parallel to get the required pixel values. The first step of decoding involves reading the compressed data from the SRVStream, which is loaded into memory and the top-level BISE compressed blocks; (3) simple integer arithmetic operations to combine the SRV pixel values with the RKV pixel values. Our algorithm decodes blocks of the LFI in parallel to get the required pixel values from the decoded blocks.

We have implemented RLFC for the two plane parameterized LF. We measure the effect of encoding parameters on the resulting compression rate and quality varies with changes in tree height is highlighted. The block threshold is set to 80, the block size is set to 4, and quantization level is set to 2 for all the datasets under consideration. As the tree height is increased, the sparsity of the residual levels in the tree increases. With the block threshold fixed the thresholding errors increase and there is a slight decrease in the resulting PSNR and bpp.

4 EVALUATION & ANALYSIS

We have implemented RLFC for the two plane parameterized LF. We have tested and analyzed our approach on the Stanford light field archives [Levoy and Hanrahan 1996; Wilburn et al. 2005]. The LFI from the dataset from Wilburn et al. [2005] are high resolution images captured using large camera arrays. The input LF images are 24-bit RGB images. We use lossless YCoCg-R [Malvar et al. 2008] color space to decorrelate the color channels. During all stages of our compression scheme, we use lossless integer computations. We measure the compression rate using bits per pixel (bpp). In the current implementation, we compress the top level representative views using the JPEG2000 lossless algorithm. The quality is measured using peak-signal-to-noise-ratio (PSNR\(_{YCoCg}\)) as the weighted average [Ohm et al. 2012] of the PSNR individual components:

\[
PSNR_{YCoCg} = \frac{6 \times PSNR_Y + PSNR_C_0 + PSNR_C_g}{8}
\]

The PSNR of each component is measured dB using:

\[
PSNR = 10 \times \log_{10} \frac{255^2}{MSE}
\]

MSE is the mean square error between the original images and the decompressed images in the LF. The final PSNR is computed as the average of all the images in the LF.

Table 1 shows the compression rate and PSNR for several LF datasets for a tree height of 3. The encoding parameters are adjusted for each dataset to achieve a similar decomposition quality. The compression rate varies from 0.08 – 2.5 bpp for a similar PSNR quality, depending on the details of the scene captured in the LF.

We measure the effect of encoding parameters on the resulting compression rate, and quality. Table 3 highlights the effect of changing the block size on the bpp and PSNR with the block threshold and the tree height set as constants. For a fixed block threshold and quantization level, increasing the block size increases the energy.
Table 3: We highlight the variation in the resulting compression rates and qualities, as the block size changes. We set the block threshold to 50, the tree height to 3, and the quantization level to 2 bits for all the datasets. For a fixed block threshold, the thresholding errors decrease when the block size increases. As a result, we observe an increase in bpp and PSNR.

Table 4: We compare RLFC with motion compensation based compression scheme in terms of compression rates (bpp) and similar compression quality (PSNR). The last column indicates the improvement factor compared to RLFC. The cells are highlighted in green indicate cases where RLFC performs better or equal than the motion compensation scheme by the mentioned factor. The cells highlighted in red indicate the cases where motion compensation performs better than RLFC.

Decoding time and frame rates: We have implemented both GPU and CPU decoders to measure the decoding times. To compute a required final pixel value, each of the pixel values from all the channels (YCoCg26) are decoded independently and the corresponding RGB pixel value is computed. The average decode times to decode a block of pixels on an NVIDIA GTX-960 and an Intel Xeon 2.4GHz are: (1) Y-Channel: 2 – 3 microseconds (2) Co-Channel: 1 – 2 microseconds (3) Cg-Channel: 1 – 2 microseconds. We have implemented a parallelized GPU LF renderer that uses our compression scheme to render new views. It takes 3 – 7 milliseconds to generate a view with a resolution of 512 x 512 for a given new camera location, resulting in average frame rates of 200 fps. While rendering a new view (512 x 512), the average time taken by all the GPU threads are as follows: memory read-gather operations, about 4 milliseconds; decoding operations, about 2 milliseconds; ray-tracing and other computations, close to 1 millisecond. For higher resolutions of 1024 x 1024, our renderer takes 8 – 13 milliseconds per frame. The rendering is performed and the decoding times are measured on a file compressed using the encoding parameters: block size 4, tree height 3, and on the Lego Knights benchmark. It turns out that the total number of blocks that must be decoded grows linearly with the number of pixels. Our algorithm performs parallel decoding on the GPU using our compressed stream structure and the frame rate decreases at a sub-linear rate.

Comparison with random access LFI schemes: Our hierarchical approach is orthogonal to the motion compensation schemes because we use totally different series of steps to exploit the redundancies among the LFI to achieve compression. RLFC offers several advantages compared to other methods:

(1) The steps used in our compression scheme are simple and enable fast real-time encoding.

(2) The hierarchy computed in our method provides progressive streaming and decoding capabilities.

(3) BISE decoding used in our approach is currently supported on all major mobile platforms and thereby making it easy to implement our approach on mobile platforms.

Our method offers at least 10X better compression ratios over the VQ method [Levoy and Hanrahan 1996] and the 4D wavelet scheme [Peter and Straßer 2001]. To compare our method with motion compensation schemes that provide random access we have implemented a motion compensation scheme that provides random access based on Zhang and Li [2000]. The comparison of the resulting bit-rates for similar PSNR quality on different datasets is presented in Table 4. For some LFI (Amethyst, Bracelet, and Bunny) RLFC achieves similar or better compression rates in comparison with the motion compensation scheme. In other cases (Lego Knights, Tarot Cards), the motion compensation scheme achieves better compression rates compared to RLFC for similar quality. More comparisons similar to Table 4 are included in suppl. material, Section-4.

On an LF dataset (Lego Bulldozer) from the Stanford LF archive with complex details in the scene and with a high resolution of 1536 x 1152 Overbeck et al. [2018] reports a compression ratio of 178:1 for PSNR of 45 dB. On the same LF dataset, our method achieves a compression of a ratio 60:1 for a PSNR of 42 dB. In comparison with RLFC motion compensation schemes provides better compression rates.
(~ 2 × 3x) compression ratios on LFI with intricate details and large areas of high-frequency components. In other cases, RLFC provides better (~ 2x) or similar compression ratios compared to the motion compensation scheme. In the case of LFI with intricate details and large areas of high frequency regions the SRVs (computed as difference of filtered RKV between alternate levels) are less sparse resulting in higher compression ratios. This can be noticed in the example visualization of the RLFC trees presented in suppl. material, Section-5.

5 CONCLUSIONS, LIMITATIONS & FUTURE WORK

Conclusion: We present a new method (RFLC) that encodes LFI by constructing a hierarchy based on computing new sets of images (key views). Our method provides random access to the LF pixel values with one level of indirection and supports parallel and progressive decompression. We have implemented our method on the two-plane parameterization of the light fields and highlight its performance. The average time to decode a block of pixels is 1–3 microseconds per channel and can be used it for interactive rendering. Our method is simple, general, and also hardware friendly.

Limitations: Our approach has some limitations. The reconstruction quality of our method for a sparsely sampled light fields can be low. Our algorithm uses filtering of spatially close by LF samples to compute the representative key views. In a sparsely sampled light field, the nearby samples may not exhibit a high level of spatial correlation. Moreover, our method doesn’t provide fine grained control over the reconstruction quality with respect to the encoding parameters. Therefore, a change in the parameters can affect the quality. Our current implementation of a GPU based LF renderer is unoptimized in memory access patterns of GPU threads and can be improved.

Future Work: We would like to extend and evaluate RFLC on other parameterizations including spherical [Ihm et al. 1997] and unstructured LF [Davis et al. 2012]. The performance of our algorithms can be further improved using a dedicated hardware implementation. In the future, we would like to evaluate our method for very high-resolution LF datasets (2160 × 1200 for HTC Vive and 1080 × 1200 for Oculus Rift) on commodity VR headsets and mobile devices. Currently, we use a uniform and localized clustering step. It will be useful to investigate better or global clustering schemes to improve the performance. We compute the representative key views at each level using weighted filtering, and it would be useful to explore other techniques based on motion vectors on a block level or image warping methods. Compared to motion compensation schemes our approach is complimentary and involves a different set of steps to exploit redundancies in LFI to achieve good compression ratios. It would be useful to explore techniques to combine our hierarchical method with the motion compensation techniques to further improve the compression ratios [Pratapa and Manocha 2018]. However, including motion compensation techniques into our scheme may add more overhead to the decoding algorithm and we need to evaluate the pros and cons carefully. The fast performance of RLFC makes it possible to integrate it with an interactive LF capturing and rendering system for a realtime LF capture and rendering system.

6 ACKNOWLEDGEMENTS

This research is supported in part by Intel.

REFERENCES


Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xin, and Ronald Buley. 2013. The latest open-source video codec VP9—an overview and preliminary results. In Picture Coding Symposium (PCS), 2013 IEEE, 390–393.


I3D ’19, May 21–23, 2019, Montreal, QC, Canada
Figure 6: The zoomed in comparisons between original and decoded LF images from our method are shown. 32 × 32 small regions as highlighted in red boxes are scaled to 512 × 512 to compare the visual quality between original and compressed images. We observe that the visual quality is not degraded in the images compressed using our method.

Figure 7: The variation of the compression rate (bpp) with change in the block threshold is plotted. The encoding parameters are set as: tree height to 3, block size to 2, quantization level to no quantization, and the block threshold is varied. With an increase in the block threshold the thresholding errors and the bpp decreases for all the LF datasets.

Figure 8: The variation of the decompression quality (PSNR) with change in compression rate (bpp) highlighted. The tree height to 3 and block size to 2 and vary the block threshold and resulting bpp and PNSR values are plotted for the LF datasets.

Figure 9: The variation of the decompression quality (PSNR) with change in compression rate (bpp) highlighted. The tree height to 3 and block size to 2 and vary the block threshold and resulting bpp and PNSR values are plotted for the LF datasets.


