D3.4 Accelerated Tools for Creating Smart Assets

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	December 31st 2019 [M24]
Actual delivery date	December 20th 2019
Leader	BRNO University of Technology [BUT]
Reply to	Marek <isolony@fit.vutbr.cz>
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	Smart Asset re-Use in Creative Environments
Document name	D3.4 Accelerated Tools for Creating Smart Assets
Security (distribution level)	PU
Contractual date of delivery	December 31st 2019 [M24]
Actual date of delivery	December 20th 2019
Deliverable name	Accelerated Tools for Creating Smart Assets
Type	Other
Status & version	Submission Version
Number of pages	27
WP / Task responsible	BUT
Other contributors	USAAR, TCD
Author(s)	Marek Solony, Pavel Smrz
EC Project Officer	Ms. Adelina Cornelia DINU Adelina-Cornelia.DINU@ec.europa.eu
Abstract	This document contains the implementations and reports of the accelerated tools for smart asset transformations for LF processing, based on the baseline tools described in the deliverable D3.2.
Keywords	Lightfield, Compression, Calibration, Tools, Depth, Superresolution, Denoising
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 28-10-19	Document created by Marek Solony
1.1 10-12-19	Version for internal review
1.2 12-12-19	Final version for submission

Table of Contents

1 EXECUTIVE SUMMARY

This deliverable is a part of the work package 3 (WP3) - New Technologies for Asset Creation - and elaborates on the deliverable D3.2 Analysis of Requirements and Prototypes of the Tools. It describes asset creation and transformation tools for Light Field (LF) processing from the acceleration perspective. The need for fast processing stems from the large amount of captured data which needs to be pre- and post-processed. Fast tools allow for immediate on-set processing and preview which is highly useful for quality assurance and reduction of the total amount of the data that needs to be captured to guarantee a successful output.

The tools and algorithms are accelerated on multiple levels: data sub-sampling, parallel processing and hardware acceleration (especially data processing on graphics cards). A part of this deliverable is also dedicated to compressing methods of the LF data, developed as a part of the SAUCE project, that aim to achieve an optimal storage size or transfer formats for huge LF assets.

2 BACKGROUND

The LF assets consist of four-dimensional (4D) light field encoding intensity and angular direction of a light ray intersecting the image sensor. Capturing the set of 4D rays at specific times extends the LF view to the temporal domain forming 5D LFs, where the 5th dimension represents time. The details about the LF assets are included in deliverable D3.1.

The amount of data depends on the capturing device. For example, the camera array constructed at USAAR contains a grid of 8x8 cameras, each capable of capturing video at a framerate of 41 frames per second. Processing and storing such amount of data is a challenging task and without accelerated algorithms can take a significant amount of time.

One of the requirements defined in D3.2 is a calibration of the LF camera array. Proper calibration allows image rectification and transforming the data into assets that are directly usable and in the format that the other tools can understand. Those assets are therefore generally easier to work with, in applications such as depth estimation, refocusing or virtual image generation. Fast on-set calibration and rectification allows inspection and previewing of the captured data using effective visualization algorithms such as a shift-sum algorithm.

Fast, accelerated implementation of the tools and prototypes is necessary for further post-processing of the captured assets in the production pipeline.

3 INTRODUCTION

This document summarizes the current stage of the development of accelerated prototypes and tools developed to address the requirements described in the previous deliverable D3.2. Each tool is described from the standpoint of its functionality, relevance for the project and acceleration compared to baseline implementation.

3.1 Main objectives and goals

The main objective of this deliverable is to provide a description of the set of accelerated versions of tools introduced in D3.2, including the levels of the acceleration. The tools will be made available for the partners to be integrated into their respective asset management frameworks, and will eventually become publicly available.

3.2 Methodology

The prototypes and initial tools have been created and described in the deliverable D3.2. Using the newly recorded datasets, the algorithms were refined and accelerated to achieve technology improvement. The performance of some of the algorithms has been measured and was included in this deliverable.

3.3 Convention

The deliverables in WP3 will use the following conventions:
We will use *italics* for emphasis, underlined for items that directly relate to the topic of the deliverable
(i.e. asset names and locations in the current deliverable) and `monospace` for code and pseudo code.

3.4 Relation to the Self-Assessment Plan

The deliverable refers to work package 3 "New Technologies for Asset Creation". Applicable success indicators are:

- Advancement of the state-of-the-art
- Laboratory testing of the prototypes and the accelerated tools

For the testing of the prototypes and accelerated tools, multiple datasets with structured and unstructured scene content have been recorded using the multi-camera array of USAAR. The tools have been developed and tested using the recorded datasets and some results are included in this deliverable. Further testing and evaluations are planned for M25-M36.

4 ACCELERATED TOOLS

4.1 Camera Array Calibration Tool (BUT)

The multi-camera array technology consists of multiple synchronized physical cameras arranged in a grid pattern with constant distances between cameras called *baseline*. Due to the manual setup of the multi-camera system, small misalignments of the cameras, either in position or in rotation can occur. In an ideal scenario, the centers of cameras would lie in the exact positions in the grid and the optical axis of cameras would be parallel. In this case, the projection planes of all cameras would be coplanar and the images produced by the cameras would be rectified.

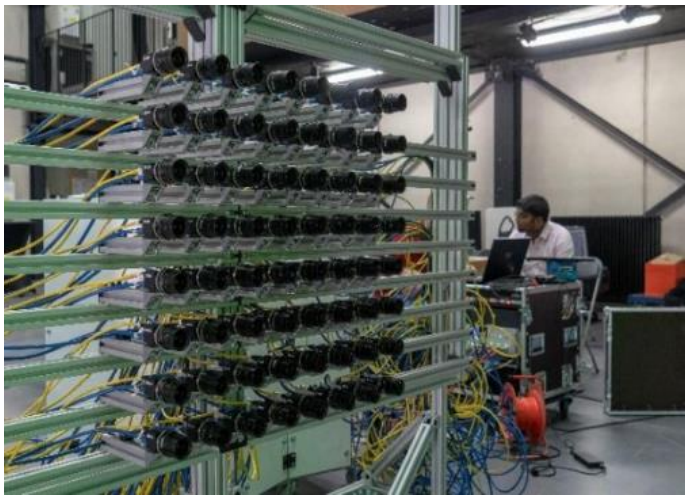


Figure 1: Multi-camera array for capturing Light Field assets.

To correct the error of manual camera setup, camera *extrinsic* parameters (position and rotation of the cameras in the world coordinate frame) need to be estimated. The parameters of the cameras can be obtained through the process called (extrinsic) camera calibration. This process exploits the mutual information about the scene that the cameras observe and applies geometry constraints to estimate the relative poses between cameras. Calibration algorithm relies purely on the visual information extracted from the scene, and therefore it is favourable to calibrate the multi-camera system using a *structured* scene - a scene with rich visual information like texture, colors, good lighting.

The calibration tool is a crucial part of the pre-processing pipeline, because the successful calibration parameters estimation affects rectification and therefore also further algorithms such as *shift-sum* or *depth estimation*. The following section describes the parts of the calibration pipeline and their acceleration.

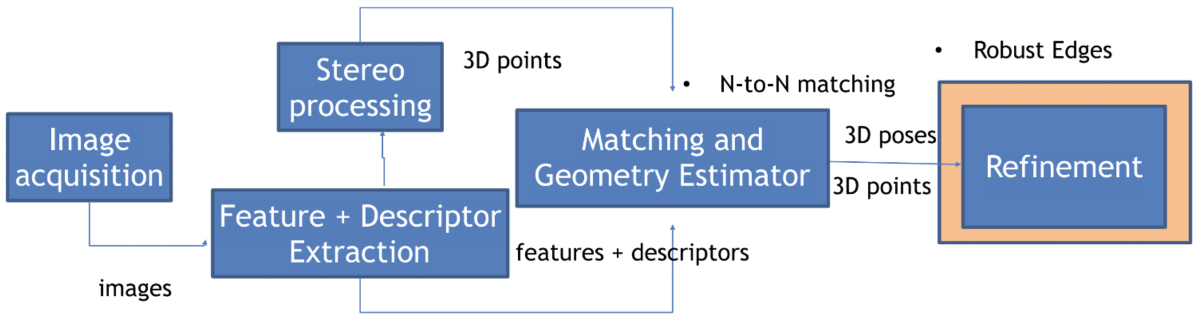


Figure 2: Calibration pipeline consists of two main parts - Initialization part that estimates the initial geometry of the cameras and Refinement part that further improves the initialization by nonlinear optimization.

BUT calibration algorithm is available for partners on sourceforge[1].

4.1.1 Image Preprocessing and Geometry Estimation

Due to the grid camera alignment and the nearly parallel optical axis, the camera frustum overlap is large between the cameras, and corresponding parts of the scene can be established. The initial step of the extrinsic camera calibration algorithm is based on image feature and descriptor extraction, and geometry estimation. This initialization step estimates the poses of the cameras and relative transformations between them. This solution can be further refined using *non-linear* optimization algorithms.

4.1.1.1 Feature extraction

Feature detectors detect the feature points on visual distinctive parts of the scene such as corners, edges of textured objects. The estimation of the relative pose between two cameras requires a set of reliable 2D point correspondences. The features from each camera are selected using a feature detection algorithm such as SIFT, SURF, KAZED, Harris, ORB [Shaharyar2018] and the descriptors are computed for each detected feature point.

The feature and descriptor extraction process can be accelerated by utilizing a graphics processing unit (GPU). Generally, the image operations that can be easily parallelized achieve substantially better performance on GPUs than on CPUs. In our implementation, we utilize OpenCV GPU_SURF, which speeds-up the process by *1000%*. All evaluations were performed on a PC with Ryzen 7 1700X processor and NVidia GTX980 graphic card.

	Classroom [s]	Unfolding [s]	HaToy [s]
CPU only feat&desc. extraction	209.82	169.39	198.33

[1] https://sourceforge.net/projects/slam-frontend-calibration2/

GPU feat&desc. extraction	19.01	12.13	17.73

Table 1: Comparison of processing times of feature and descriptor extraction task on CPU and GPU. Evaluated on SAUCE datasets described in deliverable D3.3 and D3.1.

Feature matching algorithm finds the corresponding points between the sets of feature points extracted from images. The quality of the matches is important for the estimation of 3D geometry. For SURF-like descriptors, the matching pair can be found by analysing the metric e.g., Euclidean distance of the descriptors - determining the nearest neighbour. The simplest matching algorithm computes the metrics between all possible feature points from images and the pairs with best scores are selected. Although the algorithm promises the best possible matches, the processing time can be significant with a large number of feature points.

OpenCV provides *brute-force* matching algorithm implementation on GPU, which speeds up the matching by 100%.

	Classroom [s]	Unfolding [s]	HaToy [s]
CPU only matching	697.19	132.15	707.42
GPU matching	197.49	51.06	233.36

Table 2: Comparison of processing times of feature matching task on CPU and GPU. Evaluated on SAUCE datasets described in deliverable D3.3 and D3.1.

4.1.1.2 Geometry estimation

Geometry estimation algorithms compute the relative transformation between two cameras based on visual information from the camera images. Based on the projective camera model, two cameras capturing a scene from different positions are constrained by geometric relations between camera centres, 3D points and their 2D images defined by *epipolar* geometry.

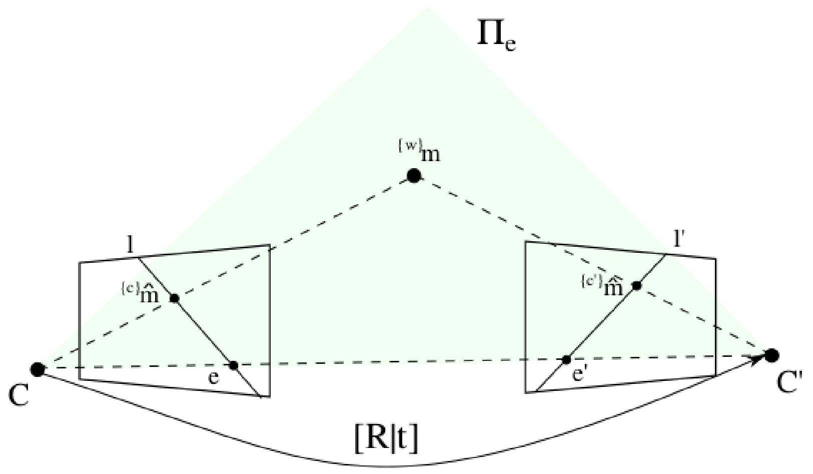


Figure 3: Epipolar geometry defines relations - camera centers, 3D point and its images in 2D camera projection plane lie on common epipolar plane.

According to epipolar geometry, to mathematically describe the relationship between the images of a 3D point observed in two cameras, without loss of generality, we can assume that the centre of the first camera lies in the origin of world coordinate system and its rotation matrix is identity. The second camera is positioned according to a rigid transformation [R | t]. We can relate the images of the 3D point by epipolar equation:

$${}^{\{c\}}\hat{m}^T E \quad {}^{\{c'\}}\hat{m} = 0$$

Where the *E* matrix is called an *essential matrix*. The essential matrix is computed by solving a set of linear equations given by the positions of the corresponding 2D images [Hartley2004]. The essential matrix is further decomposed to rotation *R* and translation *t*, defining the relative transformation between cameras.

Given the relative transformations between neighbour cameras, the global poses of the cameras can be recovered. Non-linear optimization methods can be further employed to find the configuration of parameters that minimize the sum of squared errors which is usually defined as a nonlinear function that projects the 3D scene points into the camera images and measures the distance from the observed 2D feature in the image.

4.1.2 Refinement

Refinement problem is defined as a solving of non-linear least squares problem [Brown1976]. This problem is usually addressed by repeatedly solving a sequence of linear systems. The efficient solving of this problem has been researched in [Lourakis2009], employing Cholesky factorization of the system matrix. This solution is efficient for solving small to mid-scale problems.
We utilize a hyper-graph structure to represent the optimization problem. SLAM++ implements *variables* to define sensor poses and points in 2D or 3D space and *edge* structures to impose constraints between the variables. The configuration of the system consists of variables such as sensor poses and structure points. Each variable is defined by a number of parameters according to the number of its degrees of freedom. The initial estimation of the variables is provided by the geometry estimator.

The measurements impose relations between variables, represented by edges connecting the variables involved in the measurement. Each edge gives rise to residual and the goal of the optimizer is to find the configuration of the variables that minimize the sum of squared residuals by solving the non-linear least squares problem.

4.1.2.1 SLAM ++

The joint pose and structure refinement is implemented on our open-source, non-linear graph optimization library, called SLAM++ [Ila2017]. This C++ library is a very efficient implementation of several nonlinear least squares solvers, based on fast sparse block matrix manipulation for solving the linearized problems. SLAM++ was primarily developed for efficient solving of Simultaneous Localization and Mapping (SLAM) problems in robotics, which can be formulated as a non-linear least squares problem, where variables represent robot trajectory and/or landmark positions, and the edges consist of relative measurements of the landmarks from robot positions. SLAM problem is mathematically equivalent to Bundle Adjustment (BA). The general implementation allows for the definition of variables and edges for solving BA problems as well. SLAM++ produces fast, but accurate estimations, which most of the time outperforms similar state-of-the-art implementations of graph optimization systems [Kaess2007][Kaess2012].

Implementation Details

In order to efficiently cope with very large nonlinear systems, the process of assembling and solving the sequence of linear systems must be as fast as possible. The data structure has to allow for both, efficiently recomputing the values of the system matrix every time a new linearization point is available as well as efficiently updating the system when new measurements are available. One important characteristic of those matrices is their sparse block structure [Ila2013].

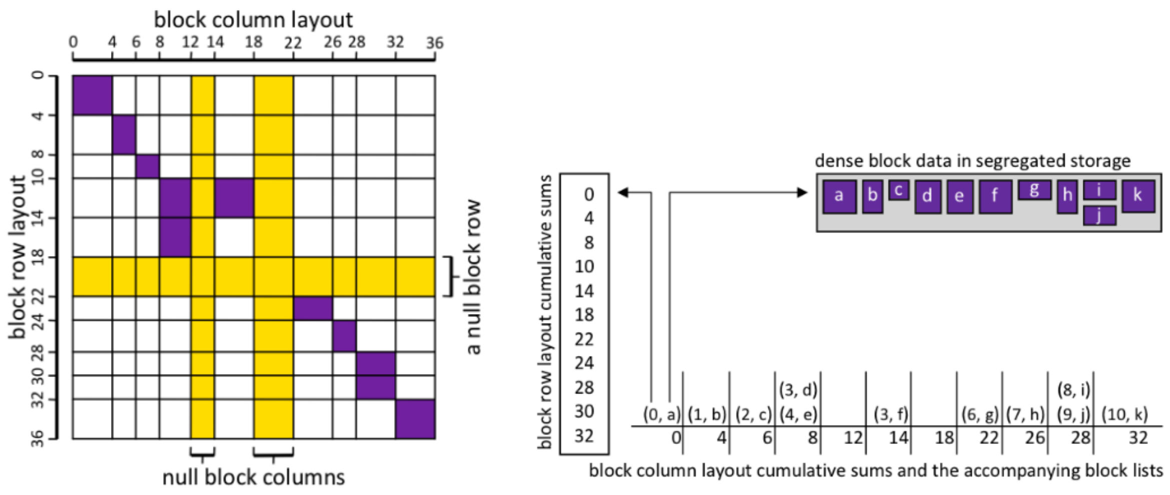


Figure 4: Block row/column layout of a block matrix. An example of a sparse block matrix and the actual values of the cumulative block sum (left, on top and left side). Non-zero dense blocks are shown in violet. Yellow shows null rows/columns. Dense block data in segregate storage (right).

SLAM++ block matrix implementation, block row and block column layouts are described using the Compressed Column Storage (CCS) [Davis2006], except that the columns also contain the non-zero matrix blocks. The structure is implemented as a sorted list of cumulative sums of block sizes (see Fig. 4). The matrix blocks are also stored in a sorted list. Each matrix block contains a row index and a pointer to matrix data. The data itself is allocated in forward-allocated segregated storage (see Fig. 4), a storage model similar to a pool but only permitting allocation and deallocation of elements from the end of the storage, in the same manner stacks do. This yields fast allocation and improves cache coherency.

In order to enable the unusually fast O(1) block random access in arithmetic operations and also to facilitate error checking for incorrectly placed blocks, one important restriction on block and column layouts must be applied. The whole area of the matrix needs to be represented, which means that the layout of null rows or columns needs to be represented as well.

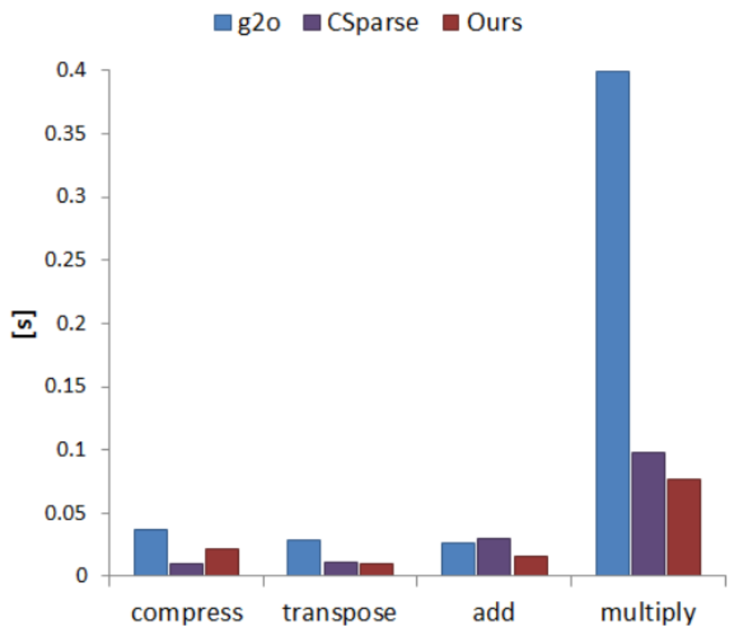


Figure 5: Block matrix arithmetic operations performance on matrices.

Times for elementary sparse matrix operations, such as compression, transpose, addition and multiplication were measured. Performance of other solvers such as CSparse [Davis2006], g2o [Kummerle2011] and our implementation were compared. The results are shown in Figure 5.

System Constraints

The hyper-graph structure of the optimization problem allows introducing various constraints on the system. Base constraints consist of the reprojection constraint - the 2D image of 3D structure points, given the camera model and parameters should lie at the same pixel positions as measured 2D features corresponding to given 3D point.

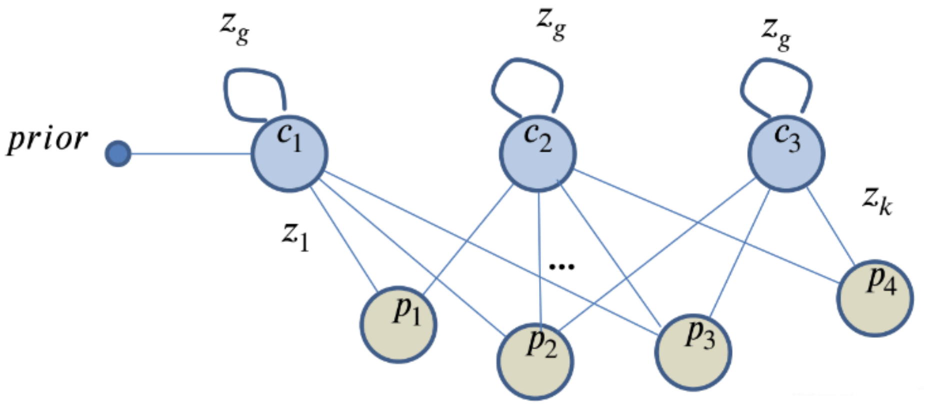


Figure 6: Graph structure of camera parameters (nodes c_1-c_3) and 3D structure (nodes p_1-p_4) optimization with reprojection constraints (edges z_1-z_k) and grid constraints (edges z_g).

Furthermore, the knowledge about the physical poses of the cameras can be introduced as a constraint to the system. Those constraints are incorporated into the system by defining a unary edge (edge joining just one variable, in this case, a pose of the camera), which represents an error function that rises when the camera pose drifts away from the expected pose. Optimizer refines the parameters of the cameras to satisfy those constraints and at the same time minimizes the reprojection error of 3D points.

	Average Translation Error [mm] (sd)	Average Rotation Error [rad] (sd)
No Grid Constraints	34.8 (13.1)	0.014 (0.006)
Grid Constraints	4.2 (2.1)	0.012 (0.005)
Robust Edges	2.3 (1.2)	0.0025 (0.001)

Table 3: Accuracy evaluation comparison with grid constraints and robust edges. The experiment was evaluated on synthetic dataset Classroom with known ground truth data.

Robust Estimation

When processing image data, a situation often arises when some of the measurements (2D feature and 3D structure points correspondences) introduced into the system are not affected by normally distributed noise, but rather have a significantly larger error. This can happen when some 2D correspondences between cameras are mismatched. One way to deal with these outliers is to introduce additional variables to the optimized system, which decide on the validity of the measurements. Alternatively, it is possible to calculate the weights of the measurements directly, without any additional variables by using standard *robust* estimators [Prasad2018].

The appealing property of robust estimators (or M-estimators) is their simple integration into the ordinary nonlinear least squares (NLS) framework. In fact, NLS is a special case of an M-estimator where the loss function happens to be the L2 norm or squared Mahalanobis norm. Other loss functions that are less susceptible to the outliers are possible, we use Huber's pseudo-L1 function. To integrate it in the NLS framework, each observation is assigned a weight.

In the context of minimization of reprojection error of the array of cameras producing 1080p images, we set the parameters of the Huber's function to *23.72,* which sets the threshold of outlier pixel to 16 pixels.

4.2 Image Rectification Tool (BUT)

Rectification algorithm is used to align the images from cameras in such a way, that the epipolar lines (horizontal or vertical, depending on the poses of cameras) of the images are parallel, therefore the cameras have a common projection plane. The corresponding 2D points in the images are located along the epipolar lines. The rectification also produces new camera projection matrices reflecting the image transformation.

4.2.1 Image Warping

The rectification algorithm first computes the closest common plane to all cameras by minimizing the sum of relative rotations of the cameras to this plane. This step will assure minimal cropping on the edges of the image. Subsequently, the homographies from camera projection planes to the common projection plane are computed. This information is then fed to the GPU accelerated image warping algorithm.

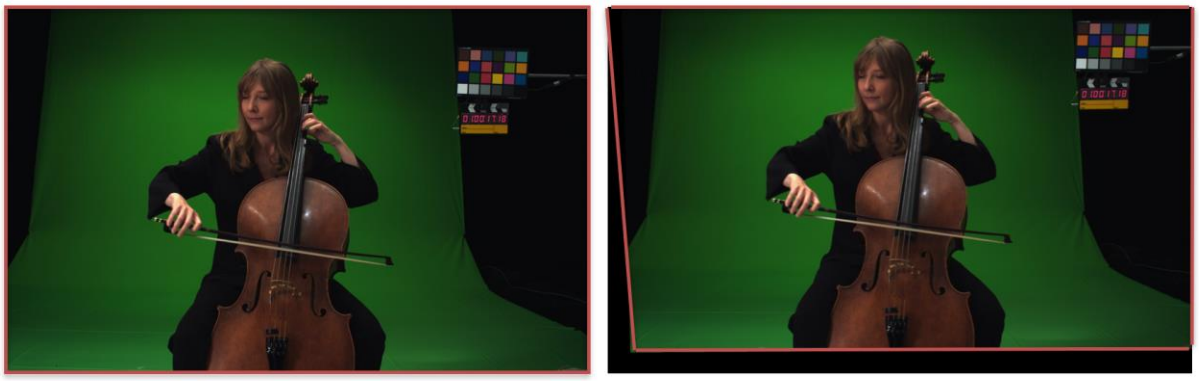


Figure 7: Original image (left) and rectified image (right).

	Classroom [s]	Unfolding [s]	HaToy [s]
CPU rectification	8.221	8.255	8.495
GPU rectification	5.145	5.321	5.498

Table 4: Image rectification evaluation. The time dependency depends only on the image size, therefore the speed-up is similar for all datasets.

4.3 Light Field Visualization (BUT)

There are two main features that light field renderer has to offer in order to fully utilize the light field data attributes when compared to the classic 2D image. The scene is captured from multiple positions so light field data contain additional spatial information about the scene. This fact should allow the renderer to change the viewing position and change the focusing plane of the rendered image. The light field player should provide the user with a way to play an interactive light field video.

The current use case for this player is an interactive preview of the recorded video in the shooting location. The player uses OpenGL for light field rendering so another possible use case is light field integration in a 3D scene such as atmospheric effects or views from windows (Figure 8) or some inaccessible areas in computer games or graphic demos.

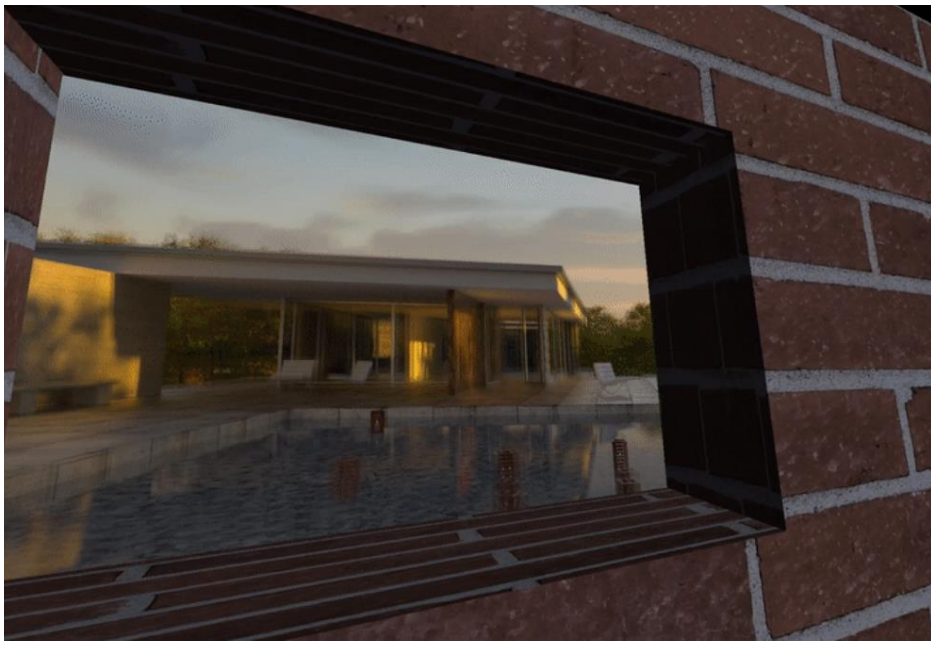


Figure 8: Light field data rendered along with a 3D geometry as a view from the window.

4.3.1 Video streaming on GPU

The main problem when utilizing GPU for light field rendering is the amount of data being transferred to the GPU memory and the expensive memory read/write GPU instructions. This leads to three constraints that need to be considered when implementing the player.

1. Minimizing the data transfer between CPU and GPU
2. Minimizing the amount of read/write operations on the GPU memory
3. Using a limited amount of GPU memory

The implemented player solves these problems by keeping no more than two light field video frames on the GPU at the same time, by reading the data for one resulting pixel only from relevant images from the camera grid and by using a video compression algorithms for data transfer from CPU to GPU.

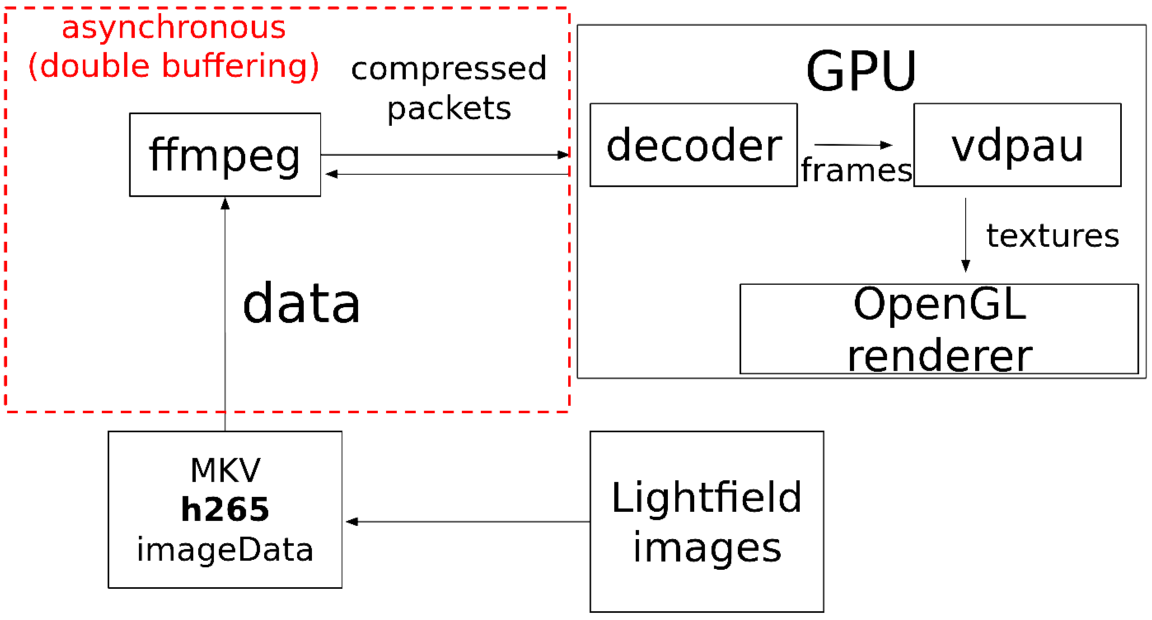


Figure 9: Scheme of light field video streaming pipeline in the player describing how lightfield images are first compressed using a video compression methods, streamed on the GPU where the decompression happens and the final frames are converted into renderable textures.

4.3.1.1 Pipeline

Light field images from the array grid are first compressed using h265 codec into a video sequence. Changes in compression ratio when choosing various image orders are not significant so images from the light field array are compressed in the video line by line starting at the left top corner. The h265 video is the input for the light field player so the compression has to be executed in advance. The video stream is read using ffmpeg framework and each packet is sent to the GPU where the hardware accelerated decompression happens. VDPAU API is used for accessing the GPU decompression units. The resulting frame is then being converted from the VDPAU video surface format to OpenGL texture using NV_vdpau_interop OpenGL extension provided by Nvidia. The final image is then rendered in shaders from the array of textures representing the camera views from the original grid. To optimize the process, the decompression and data transfer is happening asynchronously to the rendering using two OpenGL contexts. The pipeline is described in Figure 9.

4.3.1.2 Current performance status

So far the player can stream light field videos from 8x8 grid (64 images per one video frame) in FullHD (1920x1080) quality in 15.5 fps and HD (1280x720) quality in 27.85 fps (measured on GeForce RTX 2070). Additional optimizations are yet to be implemented.

4.3.2 Rendering methods

The first simple method that was initially implemented in the player was simple angle based interpolation of the resulting image. The idea is to choose the right image from the camera array according to the virtual camera position like in Figure 10 and optionally blend the nearest images. This method however is limited to only simulating the 3D view of light field data with no focusing abilities.

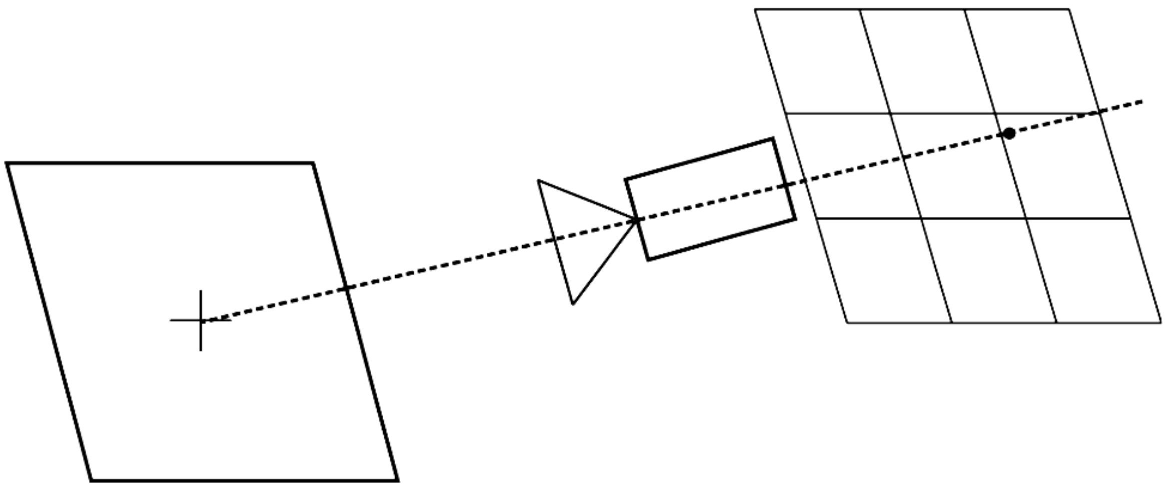


Figure 10: Simple angle based method that chooses the best image from the camera grid according to the angle between a line connecting the center of the lightfield plane with the virtual camera and the plane itself.

Another implemented method was the two planes parameterization approach [Levoy1996] where a ray from virtual camera intersects two planes that are creating an incomplete bounding volume around the light field scene. The first intersection determines the closest camera from the camera grid and the second intersection chooses which pixel from the given view is being taken into the final image. The situation is described in Figure 11. Usually four closest images are taken into account with appropriate weights.

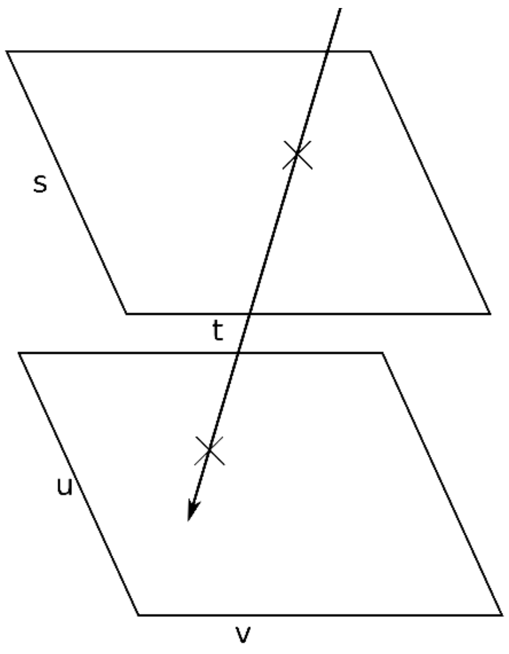
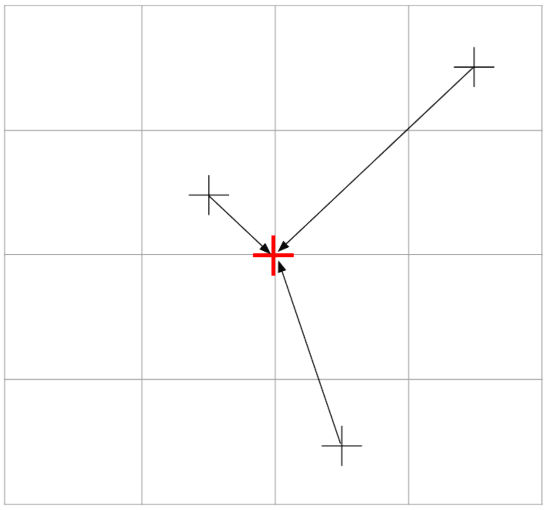


Figure 11: Two planes parameterization of light field scene. The intersection point of a ray from virtual camera on the *st* plane determines the correct input image from the camera grid and the point on *uv* plane chooses the correct part of the image to be taken into the final result.

The third method is based on shift-sum algorithm that moves the images from the camera grid to align them into one image using weighted sum described in Figure 12. The offset of each image translation is calculated from the position in the grid and chosen focusing distance. The advantage of this algorithm is that it is the most general one and can be extended by integrating the first method for choosing the right image from the grid and setting the weights of summed images according to that.



$$p(x,y) = \frac{\sum_{i=1}^{n} p_i(x + ox_i, y + oy_i) \cdot w_i}{\sum_{i=1}^{n} w_i}$$

Figure 12: Shift-sum algorithm that sums all the images from the grid translated according to their position in the grid and chosen focusing distance. The images are being aligned so the objects in the scene that are in the same distance from the camera grid and lie in the focusing distance are overlapping.

4.3.3 Future work

In terms of performance a speedup of the light field video streaming is needed for being able to stream FullHD video in at least standard 25 fps or even being able to play 4K light field videos. Some optimizations were already implemented to achieve that but the VDPAU mixer and OpenGL/VPDAU interop operations still consume a lot of computing time. We will further focus on improving these two parts of the pipeline.

Currently, only one focusing plane is being supported when rendering the light field scene such as shown in Figure 13. To improve the visual quality depth of field effect and multiple focusing planes (aka adaptive per-pixel focusing) will be implemented along with a method to determine the right focusing distance for various parts of the scene. One of the options is to use precalculated depth data which would however increase the amount of data being transferred on the GPU memory thus slowing down the streaming speed.

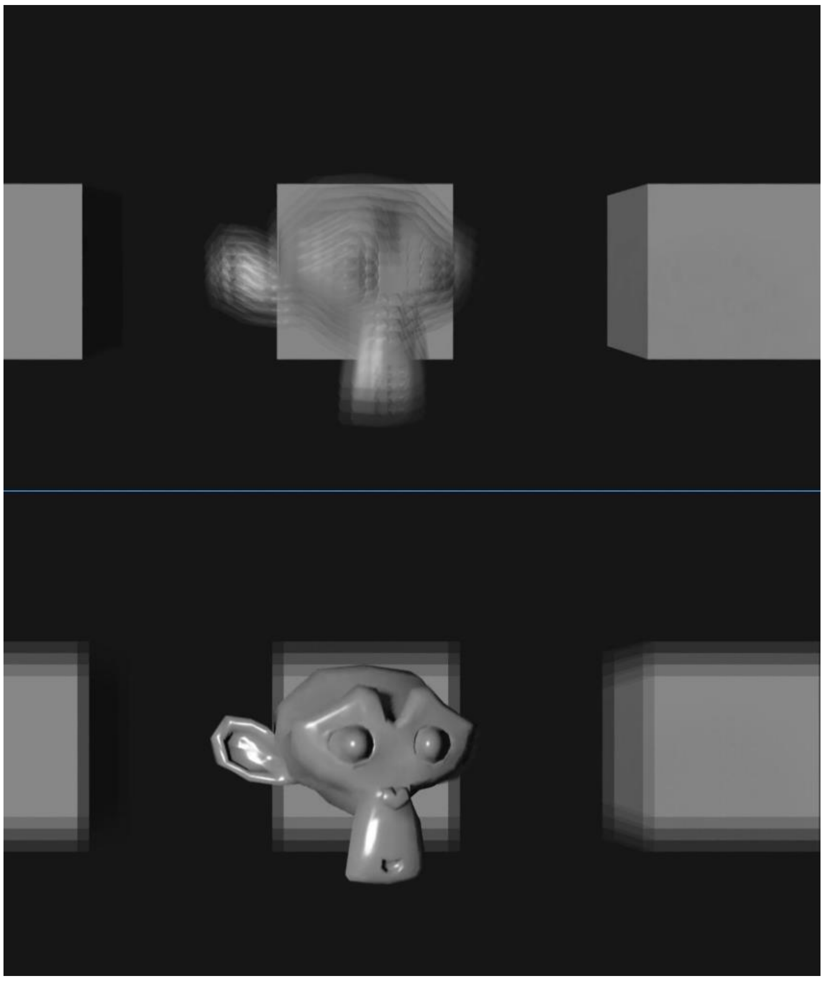


Figure 13: Refocusing of the light field scene by moving the focusing plane on the z-axis.

4.4 Light Field Assets Compression Methods

In this activity, BUT evaluated the impact of state-of-the-art image and video compression methods on the quality of images rendered from light field data. The methods include recent video compression standards, especially AV1 and XVC finalised in 2018. To fully exploit the potential of common image compression methods on four-dimensional light field imagery, we have extended these methods into three and four dimensions.

The individual views from a light field are usually never displayed. Therefore, it is not very meaningful to compare the original and decompressed light field directly, even though such methodology is usual to assess a single view compression performance. For this reason, we adopt the compression performance assessment methodology for multi-focus rendering from [Alves2016]. This methodology basically lies in assessing the quality of the rendered views for multiple focal points. The rendered views are obtained by combining pixels from different 4D light field views for various focal planes. The average distortion is computed as the mean of the PSNR for multiple rendered focal plane views. This situation is shown in Figure 14. Note that the PSNR is computed from the MSE over all three colour components.

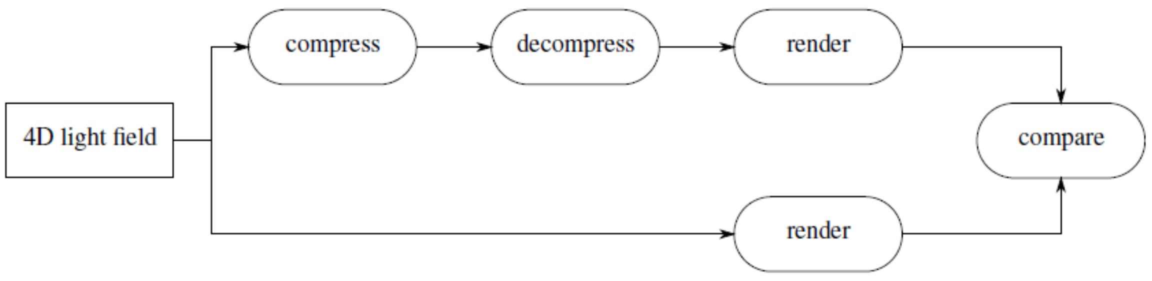


Figure 14: Data flow diagram of the compression performance assessment methodology.

The 4D light field comprises a two-dimensional grid of two-dimensional views. The baseline between individual views ranges from a few millimeters (microlenses, Lytro camera) to several centimeters (camera array, SAUCE project). It is, therefore, natural to expect a high similarity of views adjacent in any of two grid directions. This similarity opens the door to understanding the 4D light field data as a video sequence navigating between the viewpoints. Another possible point of view is to see the 4D light field as the three- or directly four-dimensional body. The above approaches can also be reflected in light field compression by using either an image, video, volumetric, or four-dimensional coding system. Although other approaches (like 3D video) are also possible, we are not aware of generally available coding systems for such cases.

The digital refocus of the images at the virtual focal plane is achieved using shift-sum algorithm [Ng2005]. This algorithm shifts the sub-aperture images (views) according to camera baseline with respect to the reference frame and accumulates the corresponding pixel values. The refocused image will be an average of the transformed images. We performed a linear interpolation in the last two 4D dimensions to convert the sampled light field function into a continuous one.

At the beginning, we wondered whether it is really necessary to assess the image quality on views rendered for multiple focal points rather than the original views (i.e. compare the original and decompressed LF directly). A quick experiment revealed that a big difference exists between the former and the latter (see Figure 15). This difference is about 10 decibels in the PSNR, depending on the bitrate and compression method. This can be explained by the fact that any pixel in the rendered view is a sum of pixels from the 4D LF so that this sum all together suppresses compression artifacts. In other words, we can afford to compress the 4D light fields much more than independent images, while maintaining the same visual quality of a screened picture.

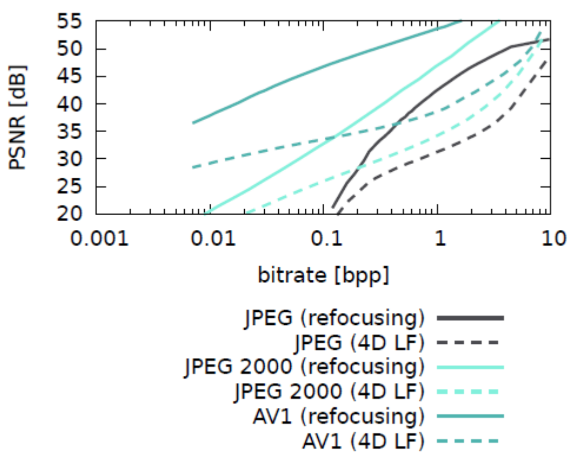
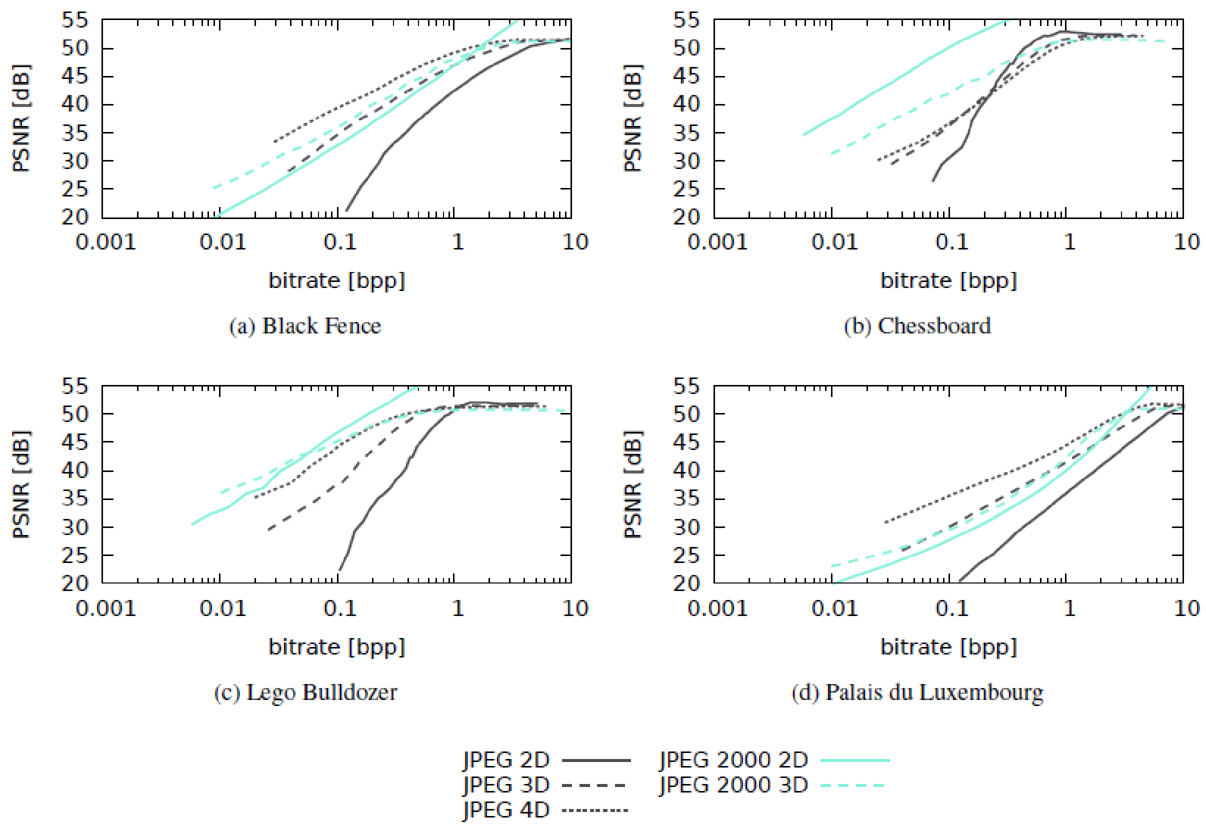


Figure 15: The difference in the quality assessment using the 4D light field directly vs. using images rendered at virtual focal planes. Shown on the Black Fence from Lytro camera.

4.4.1 Camera array vs. Lytro camera

Most current LF compression approaches handle either 2D data or their sequence (video compression). Since 4D LF are sequences of 2D images (views), the 2D compression methods may be used to code the views independently. However, such methods fail to exploit pixel correlations in all four dimensions. Similar reasoning can be used for 3D methods. In our next step, we were interested in examining the effects of LF compression in three and four dimensions. To evaluate the compression performance fairly, identical compression method must be used for the 2D, 3D, and 4D case. Thus, we have created a custom implementation of the JPEG compression method with the ability to process either the 2D, 3D, or 4D data [Barina2019]. Additionally, we are aware of the existence of the JPEG 2000 standard, with the ability to compress the 2D and 3D data in the same manner. Since the similarity of adjacent pixels in the third and four dimensions strongly depends on the camera baseline, different results can be expected depending on the baseline distance. The result of this experiment is shown in Figure 16. The horizontal axis shows the bitrate (bits per pixel), whereas the vertical axis shows the mean of the PSNR for multiple rendered focal plane views. On light fields with a small baseline (Lytro camera), both 3D compression methods clearly outperform their 2D counterparts over a whole range of bitrates. Similarly, the 4D JPEG method clearly outperforms its 3D counterpart. This is not so surprising because pixels at the same spatial position in adjacent views are strongly correlated. However, the situation changes with increasing baseline. With increasing baseline (Chessboard from SAUCE project), adjacent views are less and less similar, which results in higher amplitudes of the underlying transform coefficients. Consequently, the tide is turning in favor of the less-dimensional compression methods. Considering the JPEG method, the Lego Bulldozer is a special case because it contains large areas of blackness (black pixels). It turns out that it is more efficient to compress these solid areas at once using a single 4D block than using multiple 3D blocks. Similarly, it is more efficient to use a single 3D block than multiple 2D blocks.



(a) Black Fence (b) Chessboard

(c) Lego Bulldozer (d) Palais du Luxembourg

Figure 16: Comparison of image compression methods against their extensions into three and four dimensions.

We were also interested in whether it be better to compress the 4D light fields as a sequence of 2D frames, or as multi-dimensional body. We, therefore, measured the performance of all the above-mentioned video compression standards. The results can be seen in Figure 17. Interestingly, the XVC codec has shown better compression performance than HEVC and AV1.

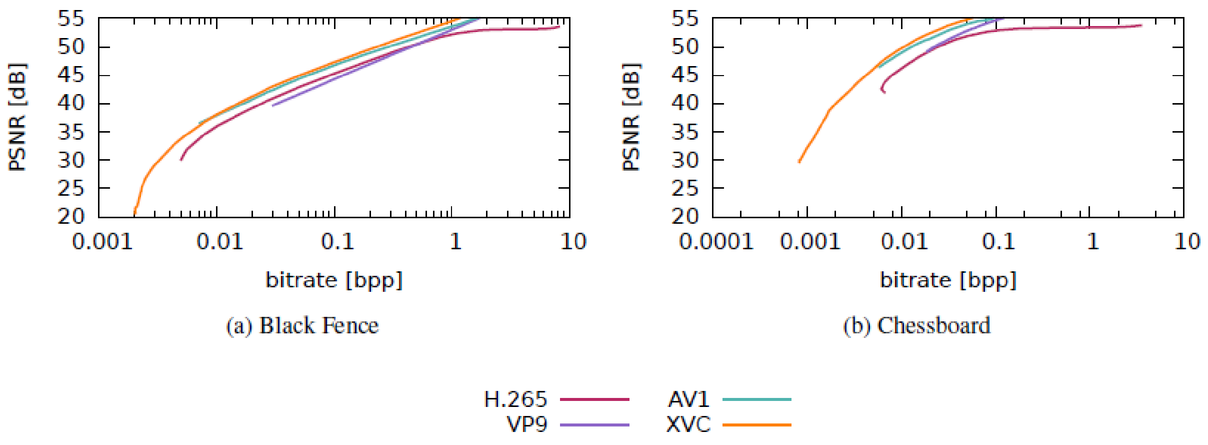


Figure 17: Performance of video compression methods.

We have further compared the performance of all above-investigated methods. The overall comparison is shown in Figure 18. Video compression methods exhibit better compression performance than all image compression methods, even better than their 3D and 4D extensions.

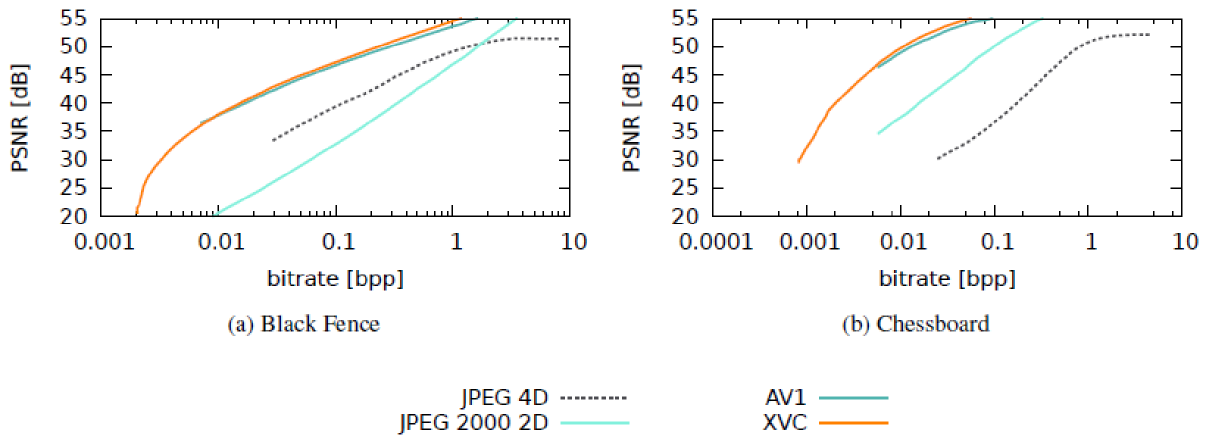


Figure 18: Overall performance of the best compression methods.

4.4.2 Focus shifted compression

We were also interested whether shifting individual 2D views (focusing on a particular point) of the 4D light field can lead to a better compression performance than direct compression of unfocused 4D light field. On all light fields, this preprocessing really leads to a significantly better performance. On LFs with a small baseline (Lytro camera), our 4D JPEG method overcomes all other methods, including H.265 and AV1. However, on light fields used in the SAUCE project, the H.265 is still unsurpassed. A quick experiment on the HaToy light field is shown below (our method is labeled the 4D, whereas the H.265 is labeled by the x265).

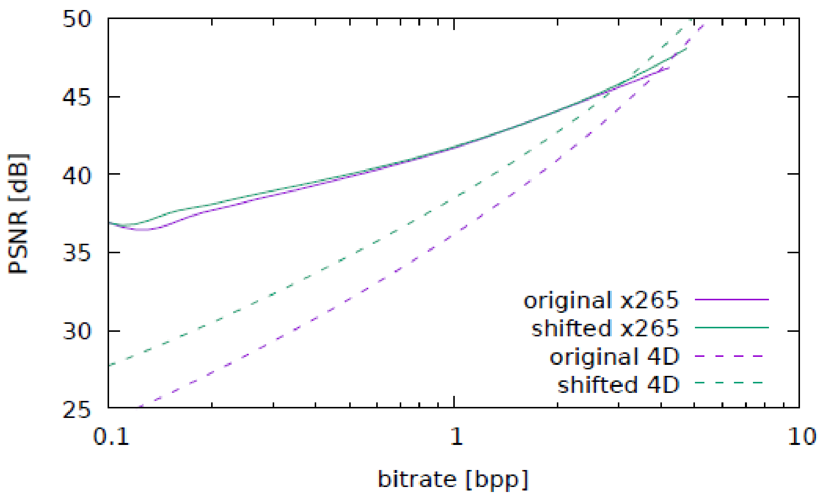


Figure 19: Focus shifted compression on HaToy light field from the SAUCE project.

4.5 Light Field depth estimation code optimization (TCD)

A light field depth estimation tool has been previously shared with SAUCE partners, which has since been optimized, both in terms of algorithm and implementation.

The main steps of the original depth estimation method consisted in computing sparse pairwise matching between views of a row or column of the light field using the Coarse-to-Fine PatchMatch (CPM) method, followed by an edge-aware filtering step using the permeability filter (PF), ending with a variational refinement step using successive over-relaxation (SOR). The output consisted in optical flow images connecting all pairs of views from the input row or column. In addition, all intermediate results were written on disk as output. An additional step using MATLAB was necessary to convert the optical flow to disparity.

In the new version of the code, the major algorithmic change consists in applying the steps described above on the input views downsampled by a factor 2. The output disparity maps are then upsampled to the original using simple bicubic interpolation, followed by the PF in order to sharpen edges. The final disparity maps are obtained after a variational refinement step. This avoids computing the CPM on the full resolution images, which is costly computationally. In addition, the code was optimized to avoid writing every intermediate result on disk if desired, and directly outputs disparity maps in .pfm format which bypasses the MATLAB post processing step, which is not only faster but also more convenient.

Objective tests conducted on the HCI benchmark, which contains 9x9x512x512 synthetic light fields with ground truth disparity maps, show that faster but also more accurate disparity map can be obtained thanks to this optimization, as illustrated in Figure 20. For the 8x8x1920x1200 light field captured during the Unfolding shoot, the computation time is now reduced to ~6s per view, while ~40s were required before optimization.

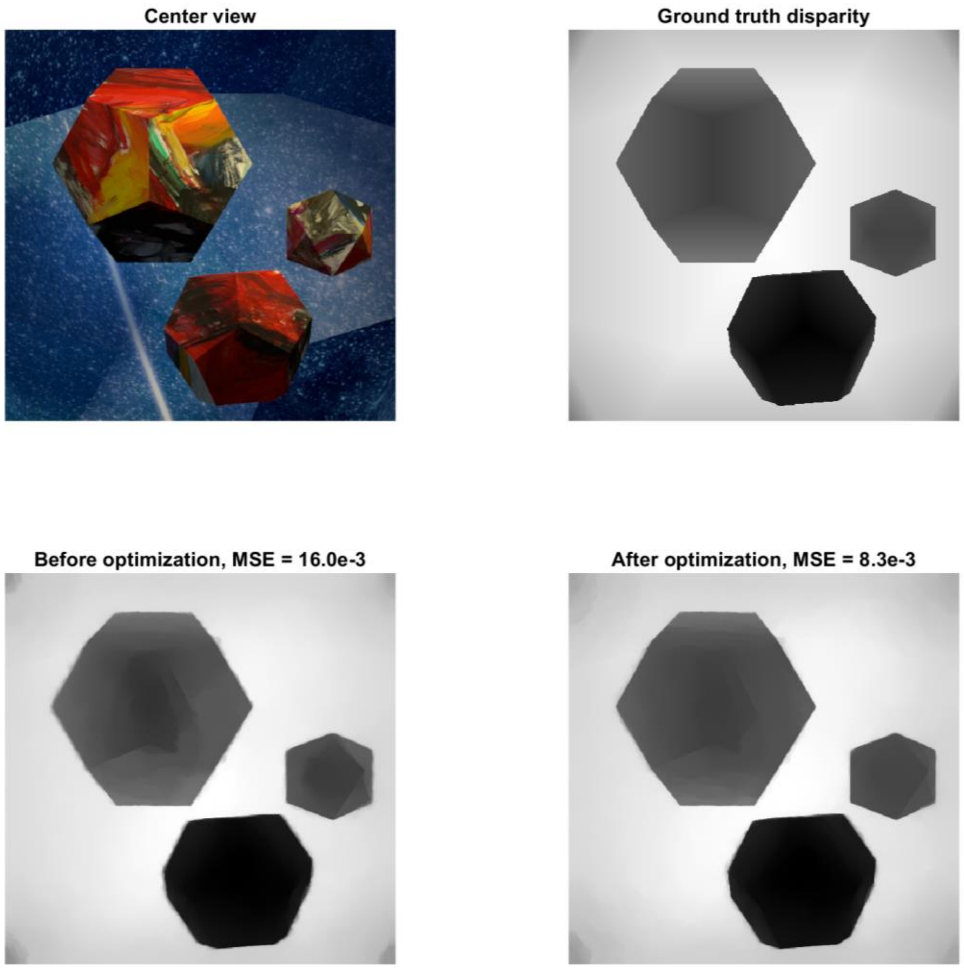

Center view
Ground truth disparity
Before optimization, MSE = 16.0e-3
After optimization, MSE = 8.3e-3

Figure 20: Disparity maps for the 'Platonic' light field from the HCI benchmark. Computation time per view is ~1.2s before and ~0.4s after optimization, while precision is also improved.

4.6 Light Field capturing pipeline

This section discusses the efforts to accelerate the capturing and processing pipeline used in the camera array used by USAAR to capture the light field assets used and produced in the SAUCE project, namely Unfolding, LF Elements and HaToy.

4.6.1 LF capture

During the first LF test shoot on the premises of the Filmakademie(FA) it became apparent that one of the most time consuming steps during capturing, besides the initial setup and mechanical camera alignment, was the transfer of the captured images from the buffers in the camera nodes to persistent storage.

The first level of caching in the camera nodes is necessary because the cameras produce about 132MB of raw data per second at the maximum resolution, colour depth and frame rate. This amount of data can't be transferred in real time to the persistent storage in real-time through the 1Gbit connection of the camera nodes and they do not process enough processing power for real-time lossless compression of the data. Unfortunately, the theoretical limit of 1Gbit per camera node is never reached when all nodes transfer data simultaneously due to bottlenecks in other parts of the network and due to the initial storage architecture.

At the beginning of the project the persistent storage consisted of five 12TB drives in a RAID5 configuration in the central control server of the array with an effective capacity of 48TB. With 64 clients streaming data from SSDs, the five disks just could not keep up. In the beginning of a transfer they up to 300MB per second, but they slowed down to about 30MB per second as soon as the hard disk buffer in the RAM of the control server was full. This meant that while the scene was only about 7 minutes long, it took several hours to transfer everything. During that time no new recording was possible and it slowed down the whole capturing process significantly.

To accelerate the transfer process, it was decided to move the persistent storage from the control server to a separate storage cluster containing 4 storage nodes with eight 12TB HDDs in each node. Even though a node level mirroring of the raw data was applied to protect against potential hardware failures, it significantly increased the transfers from the buffers to the storage. During the Unfolding shoot the system achieved constant transfer speeds of nearly 2GB per second, equivalent to the maximum throughput of the 20Gbit connection between the storage cluster and the rest of the system as shown in Figure 21. Even though it is not enough to do the transfers in real-time, it reduced the transfer times to about 3 times the capturing time which allowed for way more scenes being shot during a single day. In total the new storage cluster the use of the storage array increased the average transfer time after capturing about 60-fold.

Additionally, the storage cluster also enabled us to distribute the post-processing operations over the available hardware more efficiently as shown in the following section.

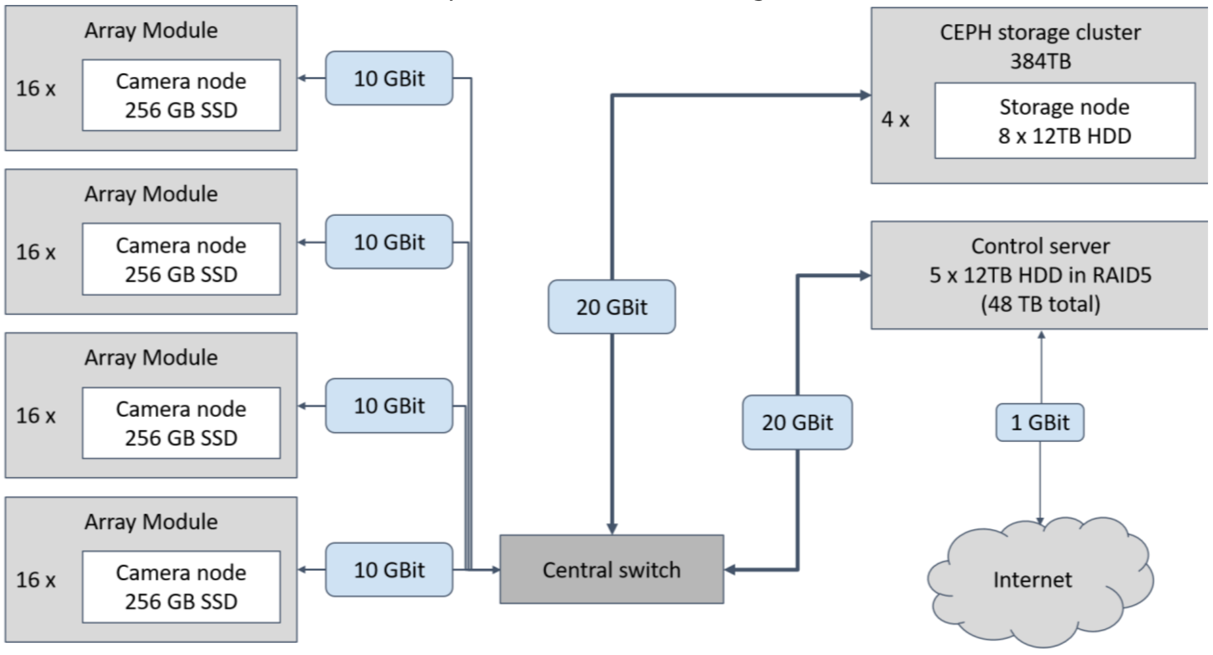


Figure 21: Storage elements present in the USAAR camera array

4.6.2 LF post-processing

After every shoot the storage of the camera array contains a large number of grayscale frames representing the raw sensor data of every camera. Even for very short scenes with a length of two seconds, over 5000 frames need to be processed. The processing pipeline is usually comprised of the following steps: First, the raw data is debayered or demosaiced to get a full colour image from the raw sensor data. Second, the colors in the images are aligned and corrected so that colours in the captured scene produce the same values in every camera. Third, the images are rectified to correct for the errors in mechanical alignment of the cameras to make the images more compatible with existing light field algorithms.

Before the post-processing can start, the optimal parameters for every step have to be determined. This process can be quite time intensive, but since this only has to be done once per scene it is not prohibitive for the overall performance, but still there have been acceleration efforts described in earlier sections. The set of colour images required as input for these parameter calculations are debayered on the control server in a non-distributed way.

The final processing of the frames of a scene was always intended to be distributed over the available camera nodes in the camera array. When the persistent storage was still located in the control server, this was very inefficient because just like in the section before, the hard drives were overwhelmed by the 64 clients requesting data to process and writing back the results at the same time. This resulted in an awful overall performance and the camera node mostly being idle and waiting for data to transfer. In total only between 10 and 15 frames were being processed per second which meant processing a very short scene of 5000 frames already took at least 5.5 minutes, and often longer, to finish.

With the storage cluster this improved significantly. The camera nodes were granted read access for the raw data and write access to the storage space for the results. They now determine independently which camera they are responsible for and start local threads which download a file, run it through the required steps, and upload the result back to the cluster as shown in figure 22. The number of threads highly depends on the complexity of every processing step and the number of available processor cores in the camera nodes. At the moment it is most efficient to process two frames at every camera node in parallel. With this architecture every camera node is capable of processing between 1.5 and 2 frames per second which results in an overall performance of 96 to 120 frames per second. Overall, we achieved a speedup of at least 6.4, which is especially noticeable for longer scenes.

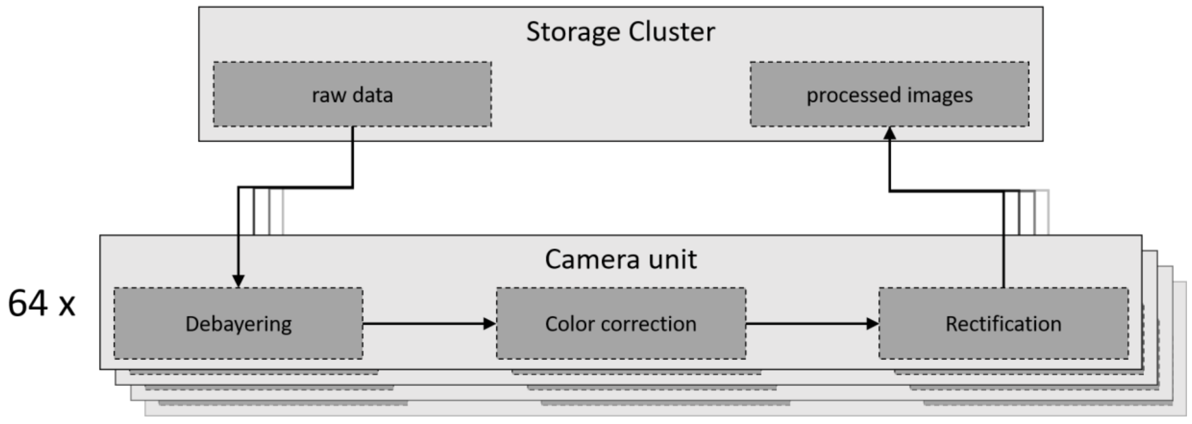


Figure 22: Data paths during post-processing

4.6.3 Real-time H.264/MVC encoding

For application which do not require the highest image quality, but can cope with compressed footage, we developed a real-time streaming approach which allows to stream the footage from all cameras in H.264/MVC encoded streams. Even though those streams have been standardized nearly 10 years ago, no optimized implementations exist for more than two views per stream, neither for encoders nor for decoders. In order to use the standard more optimally and to facilitate real-time streaming of the camera footage, we created a real-time H.264/MVC encoder and decoder chain that exploits the similarities of H.264/MVC to H.264/AVC for which optimized algorithms exist in hardware and software.

After a version which simply multiplexed multiple pre-coded AVC streams into a standard-compliant MVC stream and ran in real-time, we created two prototypes for the addition of inter-view prediction into the encoder. This was important because inter-view prediction is the main enhancement of MVC over AVC and we lose a lot of encoding efficiency without it. The idea of the first prototype is shown in Figure 23. It starts with the extraction of the I-frames from every input stream, decoding them and combining them into an additional AVC stream with normal inter-frame prediction which reduces the overall size of these frames significantly. In a second step, the transcoder replaces the I-frames from the original streams with their respective recoded versions and multiplexes the input streams into a single MVC stream. The main drawback of this approach is the fact that all frames which are not replaced by the transcoder now reference a slightly different image than when their predictions were calculated during precoding and therefore the overall quality of the final stream slightly decreases. In total we lose about 1 dB on all frames compared to the reference MVC encoder.

The second prototype mainly follows the same structure as the first. The main difference is that it pauses the precoders after the first frame, sends the results to the transcoder where the inter-view prediction is calculated. These new reference frames are transferred back to the precoders and spliced into their respective decoded picture buffers (DPB). Then the precoders resume work as normal and the transcoder just multiplexes the pre-coded streams without further modifications. This leads to a higher overall quality because the frames following the I-frames now use their original reference.

The two prototypes have only been evaluated to be real-time capable based on the amount of added complexity to the different stages compared to the simple multiplexer. A productive version of these prototypes requires two or three slightly modified versions of an existing optimized AVC encoder, respectively. Since their implementation requires a significant amount of resources and there was no need for a production version, it has not been undertaken until now.

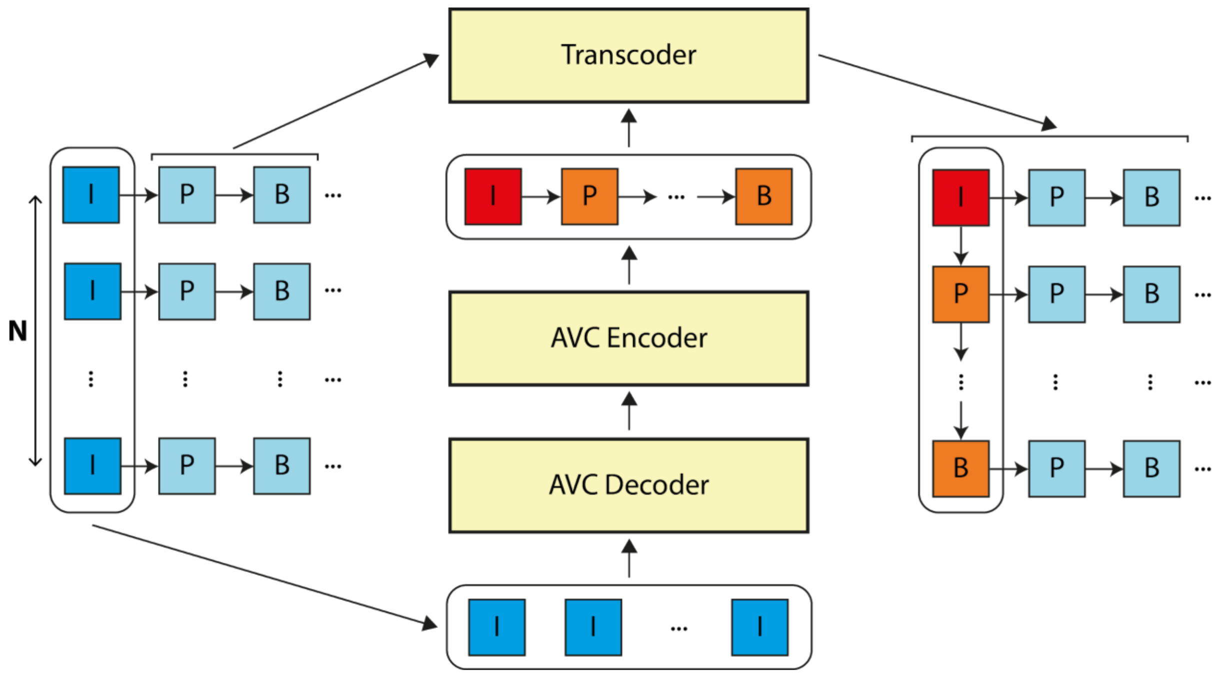


Figure 23: Structure of a real-time MVC encoder with fake inter-view prediction

5 Conclusion

This document describes the acceleration of the baseline tools developed in the deliverable D3.2. They include pre-processing tools such as precise multi-camera array calibration, LF capture and storage, rectification and also post processing tools (depth estimation) and visualization. Due to a large amount of captured and processed data, the focus lies in the acceleration of data processing and effective data storage. Multiple algorithms take advantage of fast, parallel image processing on the graphics processing unit (GPU). The acceleration is especially important for tasks that process a large amount of LF data such as rectification and depth estimation to decrease the amount of time required to prepare the datasets for further use. Additionally, effective implementation and code optimization further improve the processing time.

Future steps will include tasks aimed towards performance testing, workflow and usability evaluation in experimental production in Work Package 8:

- Further improvements in acceleration
- Integration of the tools to the processing pipeline and coordination between the partners
- Testing and evaluation in experimental production

6 References

[Shaharyar2018] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, 2018, pp. 1-10.

[Lourakis2009] M. I. A. Lourakis and A. A. Argyros. Sba: A software package for generic sparse bundle adjustment. ACM Trans. Math. Softw., 36(1):2:1–2:30, March 2009.

[Brown1976] D. C. Brown. The bundle adjustment - progress and prospects. 1976.

[Ila2017] V. S. Ila, L. Polok, M. Solony and P. Svoboda. SLAM++-A Highly Efficient and Temporally Scalable Incremental SLAM Framework. *The International Journal of Robotics Research*, vol. 2017, no. 1, pp. 210-230. ISSN 1741-3176.

[Kaess2007] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. pages 1670–1677, Rome, Italy, April 2007.

[Kaess2012] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research, 31(2):216–235, 2012.

[Ila2013] V. S. Ila, L. Polok, P. Smrz, M. Solony and P. Zemcik. Incremental Cholesky Factorization for Least Squares Problems in Robotics. In: *Proceedings of The 2013 IFAC Intelligent Autonomous Vehicles Symposium*. Gold Coast: IEEE Computer Society, 2013, pp. 1-8. ISBN 978-3-902823-36-6.

[Kummerle2011] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard,"g2o: A general framework for graph optimization," inProc. of theIEEE Int. Conf. on Robotics and Automation (ICRA), Shanghai, China,May 2011.

[Prasad2018] A. Prasad, A. S. Suggala, S. Balakrishnan and P. Ravikumar. Robust Estimation via Robust Gradient Estimation, arXiv, 2018.

[Davis2006] T. A. Davis,Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2). Society for Industrial and Applied Mathematics, 2006.

[Hartley2004] R. I. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Barina2019] D. Barina, T. Chlubna, M. Solony, D. Dlabaja and P. Zemcik. Evaluation of 4D Light Field Compression Methods. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG), Part I*. Computer Science Research Notes (CSRN), vol. 2901.

[Alves2016] Alves, G., Pereira, F., and da Silva, E. A. Light field imaging coding: Performance assessment methodology and standards benchmarking. In IEEE International Conference on Multimedia & Expo Workshops (ICMEW), pages 1–6, 2016. doi:10.1109/ICMEW.2016.7574774.

[Ng2005] Ng, R., Levoy, M., Bredif, M., Duval, G., Horowitz, M., and Hanrahan, P. Light field photography with a hand-held plenoptic camera. Technical report, 2005. Stanford Tech Report CTSR 2005-02.

[Levoy1996] Levoy M, Hanrahan P. Light field rendering. InProceedings of the 23rd annual conference on Computer graphics and interactive techniques 1996 Aug 1 (pp. 31-42). ACM.

[Herfet2018] Herfet, T., Lange, T., & Hariharan, H. P. (2018, December). Enabling Multiview-and Light Field-Video for Veridical Visual Experiences. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)* (pp. 1705-1709). IEEE