CGI2014 manuscript No.
(will be inserted by the editor)

Hand-Held 3D Light Field Photography and Applications

Daniel Donatsch · Siavash Arjomand Bigdeli · Philippe Robert · Matthias Zwicker

Abstract We propose a method to acquire $3D$ light fields using a hand-held camera, and describe several computational photography applications facilitated by our approach. As our input we take an image sequence from a camera translating along an approximately linear path with limited camera rotations. Users can acquire such data easily in a few seconds by moving a hand-held camera. We include a novel approach to resample the input into regularly sampled $3D$ light fields by aligning them in the spatio-temporal domain, and a technique for high-quality disparity estimation from light fields. We show applications including digital refocusing and synthetic aperture blur, foreground removal, selective colorization, and others.

Keywords $3D$ light fields, computational photography, disparity estimation, digital refocusing

1 Indroduction

Modern smartphones and tablet computers with their ever increasing computational power provide fascinating opportunities to implement computational photography applications without resorting to off-line computation. In this paper, we describe a method for hand-held 3D light field photography. As input we take image sequences captured with a hand-held camera along approximately linear trajectories. Capturing such data is a matter of a few seconds and does not require any extra equipment. At the core of our approach then is an efficient method to resample the input image sequence into a regularly sampled 3D light field, that is, the light field corresponds to a linear camera motion with equidistant views. This light field then opens up the possibility for a variety of further processing. First, we present a high quality algorithm for disparity estimation. Based on the disparity map, we then propose applications for digital refocusing, foreground removal, segmentation, object insertion, and multiview autostereo output.

Our approach shares similarities with recent techniques that attempt to perform multi-view 3D reconstruction [1] and 4D light field acquisition [2] on mobile devices. The main goal of multi-view reconstruction techniques is to produce full 3D models, which can then be used, for example, for 3D printing. While these techniques produce impressive results, they require several minutes of user interaction to obtain high quality reconstructions. Similarly, unstructured 4D light fields require the acquisition of many images from viewpoints distributed over a 2D domain, for example roughly on a hemisphere around an object of interest. In contrast, data capturing for our approach takes just a few seconds. The focus of our approach is not on full 3D reconstruction or image based rendering, but on providing advanced computational photography tools. In summary, we make the following contributions:

- An efficient technique for resampling image sequences along an approximately linear camera trajectory into 3D light fields.
- A high quality disparity estimation technique based on 3D light fields.

Daniel Donatsch
University of Bern, Switzerland
donatsch@iam.unibe.ch

Siavash Arjomand Bigdeli
University of Neuchatel and 3D Impact Media, Switzerland

Philippe Robert
3D Impact Media, Switzerland

Matthias Zwicker
University of Bern, Switzerland

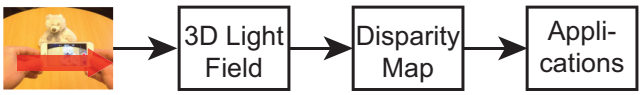


Fig. 1 Overview of our processing pipeline.

- A technique to generate out-of-focus blur leveraging 3D light field data.
- A proof-of-concept implementation demonstrating feasibility of our approach on a mobile device.

Figure 1 shows an overview of our pipeline. Given an input image sequence from a hand-held camera under a roughly linear trajectory, we first resample the data into a regularly sampled $3D$ light field (Section 3) and then perform disparity estimation (Section 4). Finally, we leverage this data for several computational photography applications (Section 5), including digital refocusing, foreground removal, segmentation, object insertion, and multiview autostereo output. Finally, we present results from a proof of concept application for mobile devices in Section 6.

2 Related Work

Resampling image sequences from approximately linear camera motions into 3D light fields is similar to video stabilization. Our approach is most related to the work by Feng et al. [3]. They proposed to use a linear analysis of feature tracks in the input video to recover a lower dimensional subspace, where the projection into the subspace is related to the camera motion. By smoothing the projection matrix they then obtain smoothed feature tracks. In contrast to their approach, we solve an optimization problem to obtain a linear camera trajectory that best approximates the input camera motion. We also resample the input images temporally to obtain a camera motion with constant speed. Similarly to their technique we render the output views using content preserving image warps [4]. Video stabilization can also be solved by reconstructing the 3D camera path [4], or by smoothing 2D feature trajectories under additional constraints [5]. Subspace analysis is attractive for us because it avoids the complexities and robustness issues with reconstructing the full 3D camera motion, but it provides enough information to achieve a linear camera motion at constant speed.

Our disparity estimation algorithm is inspired by the recent work of Rhemann et al. [6] and Kim et al. [7], whereas the latter represents the state-of-the art for disparity estimation from light fields. Kim et al. showed that very high quality disparity estimation is possible from light fields with high spatio-angular resolution by estimating disparity scores for single pixels. We use a similar approach to obtain initial estimates for disparity scores. Then we use an efficient edge aware filter to remove noise in our initial score volume of disparity hypotheses as proposed by Rhemann et al. While they apply the guided image filter [8] for this purpose, we are building on domain transform filtering [9], which allows us to easily include additional confidence values for the disparity hypotheses in the filtering process. We present a comparison of our approach and these techniques using standard datasets in Section 4, demonstrating the improved quality of our method.

Digital refocusing is one of the main applications of our framework. Ng [10] and Isaksen et al. [11] showed in their seminal work how 4D light fields can be used to refocus digital images after the fact. Unfortunately, applying the same techniques directly to 3D light fields would lead to unnatural one-dimensional out-of-focus blur. In our approach, we leverage our disparity maps to combine 3D light field refocusing with an image based blur to achieve convincing results. An even simpler approach to achieve digital refocusing would be to use a focus stack, which has been implemented in commercial mobile applications [12]. These techniques, however, cannot increase the defocus beyond the limits imposed by the aperture of the camera. Our approach allows for a very large synthetic aperture, and we provide additional functionality such as completely removing thin foreground objects, inspired by the work by Joshi et al. [13]. Defocus blur can also be manipulated using image processing techniques [14], but the quality of this approach is limited since it is purely image based, and it produces artifacts in particular when foreground objects are out of focus.

Beyond refocusing, our technique enables other light field processing techniques such as alpha matting [15]. We found, however, that in practice a simpler approach using edge aware filtering is more robust. Finally, the 3D light fields produced by our technique can also be used for multiview autostereo displays [16].

3 Spatio-Temporal 3D Light Field Resampling

The input to our method is an image sequence from a camera sweep, similar to a sweep panorama. The sweep should be a left-to-right (or right-to-left), approximately linear camera motion without significant camera rotation. The user then picks one view as a reference image, which we will use to resample the 3D light field as described below, compute a disparity map (Section 4), and perform our applications (Section 5). A camera sweep acquired using a hand-held device is unlikely to be perfectly linear, and the images usually are

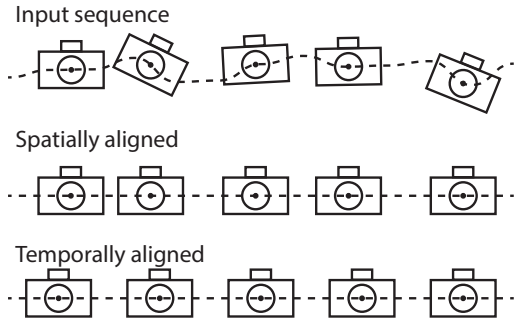


Fig. 2 Overview of our resampling. From the input sequence (top), we first search for a horizontal camera path (middle). Then, we resample this path regularly and compute equidistant views (bottom).

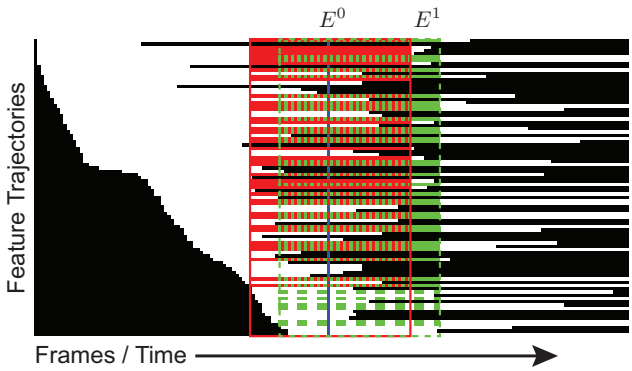


Fig. 3 We show a part of the feature trajectory matrix. Trajectories (white) are ordered according to their first appearance. The initial factorization window is red, and a second window is green. With the C^0 matrix entries from the first window (green dotted lines) and the additional frames of these trajectories (solid part of green lines) in the next window we compute E^1. Next we compute coefficients for trajectories that span the second window, but did not span the previous one completely (dashed green lines).

non-equidistant samples along the camera path. Hence we perform a linearization of the camera path in a first step (Figure 2, Sections 3.1 to 3.3). In a second step, we produce new views from equidistant camera positions along this linear path (Section 3.4).

3.1 Feature Trajectory Matrix

Our stabilization and resampling process is based on Liu et al.'s subspace video stabilization [3]. We begin with feature tracking and feature matching, and obtain a collection of feature trajectories $\{(x_t^i, y_t^i)\}$, where i is the feature index, and (x_t^i, y_t^i) are the coordinates of the feature on frame t. We collect the trajectories in a trajectory matrix,

$$M = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_F^1 \\ y_1^1 & y_2^1 & \dots & y_F^1 \\ & & \vdots & \\ x_1^N & x_2^N & \dots & x_F^N \\ y_1^N & y_2^N & \dots & y_F^N \end{bmatrix}, \tag{1}$$

where F is the number of frames of the input sequence and N the number of trajectories we found. Not all features can be tracked over the full duration of the video in general, and for missing features we set their corresponding entries in M to zero.

3.2 Factorization

The seminal work by Irani [17] showed that the trajectory matrix M can be approximated by a matrix with rank 9. Irani factorizes M into two matrices C and E. The feature coefficient matrix $C \in \mathbb{R}^{2N \times 9}$ describes the $3D$ structure of the N feature points, and the camera matrix $E \in \mathbb{R}^{9 \times F}$ represents the F camera positions and the projections of the features onto the frames. We will exploit this in Section 3.3 where we search for a new camera matrix which describes a linear camera motion. Since CE is a full matrix, we multiply it element-wise with a binary matrix W consisting of ones where M has a non-zero entry, and zeros elsewhere. Hence, the matrix factorization we look for becomes

$$M \approx W \odot CE, \tag{2}$$

where $\odot$ denotes element-wise multiplication.

We incrementally factorize M with the moving factorization method described by Liu et al. in [3]. In our approach, we select our initial window such that the reference frame F_m is in its center. The initial window is depicted in red in Figure 3. We then collect all trajectories that span the whole window in a trajectory matrix M^0, which we decompose using SVD. By truncating the resulting matrices to 9 rows resp. columns and distributing the square roots of the 9 largest eigenvalues to the left and right matrices we get a camera matrix E^0 and a coefficient matrix C^0. Next, we move the window forward as depicted in green in Figure 3, and we search again for the trajectories that span the whole window. Since now we have some trajectories that spanned the previous window, too, we already have coefficients in C^0 for them. These cases are depicted with green dotted lines in Figure 3. With these coefficients we can compute the missing entries for the camera matrix E^1, which corresponds to the frames that are not covered by the previous factorization windows. The camera matrix is then complete for the current window, and we

can compute the feature coefficients C^1 for the new trajectories that completely cover the current factorization window (and have not been computed before). We mark these trajectories with green dashed lines in Figure 3. Finally, we repeat this process forward and backward in time until all frames are processed. For a more formal description we refer to [3].

In the process above, we compute the coefficients in C for each feature with the knowledge of only one factorization window, although most feature tracks extend beyond a single window. The restriction to single windows may fail if, for example, the camera moves only very little during this window and does not constrain C enough. Therefore, we verify the validity of the coefficients of each feature by checking if the difference between the approximation using the factorization and the input feature location ever exceeds 3 pixels. If this test fails, we recompute the feature coefficients by taking into account the whole feature trajectory and test the factorization error again. In the end, we keep only trajectories for which the approximation never differs more than 3 pixels.

3.3 Linear Camera Motion

To construct a 3D light field we require a linear camera path and completely horizontal feature trajectories. In addition, the camera and the features should stay as close as possible to the input. Hence we seek trajectories with constant y-coordinates, and the x-coordinates along the linearized camera path should stay as close as possible to the input. Remember that we factorized the trajectory matrix into a feature coefficient matrix C and a camera matrix E. Further, we can split the coefficient matrix into submatrices C_x and C_y, which give rise to the x- and y-coordinates of the feature trajectories, respectively. With that in mind, we now search for a new matrix $\hat{E}$, such that $C_y\hat{E}$ is row-wise constant. Since our desired camera motion is linear, the columns of $\hat{E}$ representing the cameras in each frame must follow a linear model. This is, $\hat{E} = E^sT + E^b$, where E^s and E^b are both column vectors of height 9, and T is a row vector of length F. Intuitively, the vector T marks the points in time each frame was captured. Our goal is now to determine the unknowns E^s, E^b, and T.

We further reduce the degrees of freedom of the system by holding the reference frame F_m fixed. As a consequence, the y-coordinate for all trajectories is given by that frame. We then create a matrix δM containing the differences of the feature coordinates in the trajectory matrix M to the location in the reference frame F_m. Note that the column m of δM is all zero. We conclude that $T_m = 0$ and $M_m = W_m \odot CE^b$. The subscript

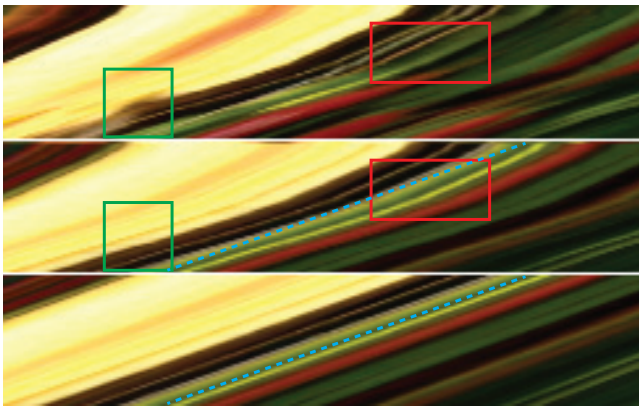


Fig. 4 We show the EPI of the input sequence on top. Lines may become thinner or wider (green box) or may disappear (red box). In the middle is the EPI after the linearization of the camera path. The structure of the light field is now clearly visible. Still, the lines are curved as the comparison to the blue line shows. In the bottom EPI the lines became straight after temporal resampling.

m denotes the m-th column of M and W respectively, and the m-th entry of T. It follows that E^b is equal to the m-th column of E. Hence, our problem reduces to the minimization

$$\arg\min_{T,E^s} ||C_y(E^sT)|| + \alpha||\delta M_x - W_x \odot C_x(E^sT)||. \quad (3)$$

The first term pushes the y-coordinates towards the ones in F_m. The second term keeps the x-coordinates where they were on the input frames and prevents the system from returning the trivial solution, and α is a factor to balance the two terms. We usually obtained best results with $\alpha = 1$.

3.4 Rendering of Output Views

We finally compute the output feature locations and render the views of the regularly sampled $3D$ light field. With T and E^s given, and $\{(x^i_m, y^i_m)\}$ the feature locations on the reference frame, the feature locations on frame j on a perfectly linearly moving camera are

$$\{(x^i_m, y^i_m) + (C^i_x E^s T_j, C^i_y E^s T_j)\}, \quad (4)$$

where T_j denotes the j-th entry of T. Still, it is possible that the camera changes speed along its linear trajectory. This leads to curved lines in the EPI as we show in Figure 4. To avoid this, we manipulate T. We set

$$\Delta t = \min(|\min(T)|, |\max(T)|)/n \quad (5)$$

where n is an arbitrary number of views we want to create on each side of the reference frame. For the l-th output image, with $l \in \{-n, ..., n\}$, we compute its time

$t = l\Delta t$, and we use t to compute the new x-coordinates. For the y-coordinates we use the location on the reference frame F_m directly. This leads to the output feature locations $\{(x^i_m, y^i_m) + (C^i_x E^s t, 0)\}$.

We render the output view by searching for t's next smaller and larger entry in T. We warp the corresponding two input frames with the content-preserving warp from Liu et al. [4], and linearly blend the two warped frames to produce the output image. After rendering all $2n$ images our $3D$ light field is complete and the EPIs show straight lines as we show in Figure 4.

4 Disparity Map

For most of our applications we need a disparity map for our reference image. We compute this disparity map using the $3D$ light fields that we obtain as described in the previous section. We first construct a score volume that holds a score for a set of disparity hypotheses at each pixel, where larger scores indicate higher quality matches (Section 4.1). We then filter each disparity slice of the score volume using a structure preserving filter to increase the robustness of our initial score estimates (Section 4.2). We assign a disparity to each pixel with a winner-takes-all strategy over the filtered disparity hypotheses at each pixel. Finally, we apply a bilateral median filter to get our output disparity map.

4.1 Score Volume Computation

We construct our score volume using the stabilized images I_j from the previous section as input. For each disparity hypothesis from a predetermined set of hypotheses, we shift all the images horizontally with respect to the hypothesized disparity. We then compare the pixels in the shifted images with the reference image I_m and compute a score for each pixel. We adopt the similarity measurement from Kim et al. [7] using an Epanechnikov kernel. Additionally, we also take into account the horizontal image gradients.

More precisely, we define the initial score for pixel (x, y) and disparity hypothesis d as

$$S(x,y,d) = \sum_{j \neq m} K(I_j(x_s, y) - I_m(x,y)) \cdot K(\nabla_x I_j(x_s, y) - \nabla_x I_m(x,y)), \quad (6)$$

where j is the index of the input image, $x_s = x + (j - m)d$ is the shifted pixel position under disparity d, and ∇_x is the horizontal image gradient. The similarity kernel K is the Epanechnikov kernel $K(z) = 1 - \|z/h\|^2$ if $\|z/h\| < 1$ and 0 otherwise. We set the threshold h to $h = 9$, where pixel values are in the range $[0, 255]$.

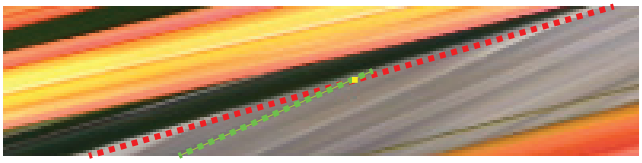


Fig. 5 The accumulated score of the red disparity hypothesis is larger than the one of the green one because we find more pixels with non-zero scores along the red line. On the other hand, the green hypothesis has fewer but higher non-zero scores. Our normalization gives preference to the correct hypothesis in green.

We observe, however, that in regions containing occlusions as in Figure 5 the raw score from Equation (6) is biased towards the foreground disparity. For the pixel marked in yellow the correct disparity corresponds to the green ray, which belongs to the background. Along this ray, however, we have fewer non-zero scores in the sum of Equation (6) because of the occlusion by the foreground in some of the views. Hence the sum of the scores of the disparity of the foreground, drawn in red, is higher, although the value of the individual scores in Equation (6) are smaller. To avoid this effect we include a normalization step in our approach.

We first define a confidence measure $C(x, y)$, which captures at each pixel (x, y) the ratio by which the highest score outperforms the average score, that is,

$$C(x,y) = \frac{\max_d(S(x,y,d))}{\frac{1}{D}\sum_d S(x,y,d)}. \quad (7)$$

This ratio indicates how unique the maximum score is with respect to the average score. In an ideal case the score is non-zero only for a single disparity hypothesis and the confidence takes on the value D, the number of disparity hypotheses. The confidence goes to 1 as the maximum score gets closer to the average.

Situations as in Figure 5 lead to low confidence values, because the scores of the disparity hypotheses of the yellow pixel exhibit several peaks instead of a single one. Therefore, if the confidence is low we divide each score by its corresponding number of non-zero values in the sum in Equation (6). This favors disparity hypotheses with fewer, but higher scores, and allows us to more robustly detect the background disparity. If confidence is high there is likely a single peak in the scores and the normalization is not necessary. It may even be counterproductive, since it reduces the prominence of the peak. Hence in this case we normalize all scores for a pixel by the same factor, which is the number of non-zero values in the highest score in Equation (6). We obtained all our results with a threshold of $D/4$ on the confidence.

4.2 Score Volume Filtering

The purpose of the score volume filtering step is to reduce the noise in our initial per pixel score estimate described above. We apply an edge preserving filter to the (x, y)-slice of each disparity hypothesis similar as proposed by Rheman et al. [18]. Instead of the guided image filter, however, we use the domain transform filter (DTF) introduced by Gastal and Oliveira [19], whose computational complexity is linear in the number of pixels to be filtered and independent of the filter support size, similar as for the guided filter. The most attractive property of the DTF for our problem is that its support adaptively shrinks or expands according to the image structure. In particular, in highly uniform areas where disparity estimation is notoriously difficult, the filter takes on a large support. In regions with rich structure, in contrast, the filter support shrinks. Intuitively, the DTF weights pairs of pixels by their distance (according to some metric) along a path connecting them in the image. This is similar to geodesic filtering, and indeed the domain transform approach can be interpreted as an iterative approximation of geodesic filtering. Here we focus on our extensions of DTF for filtering our score volumes. Please see the original publication [19] for more details.

We first give a simplified explanation of the basic DTF in 1D. Assume the input is a $1D$ function $I(x) : R \to R$. The DTF weight for two neighboring pixels at locations x and $x + h$ is defined as $g(|ct(x) - ct(x + h)|)$, where g is a filter kernel, and $ct : R \to R$ is the *domain transform function*, which is at the core of the approach. The main idea is to define ct in a way such that the absolute value $|ct(x) - ct(x + h)|$ is related to a l_1 distance in $2D$ between the two $2D$ points given by the pixels and their function values. This l_1 distance is defined as $\sigma_s h + |\sigma_r(I(x) - I(x + h))|$, where σ_s and σ_r are filter parameters similar to the spatial and range parameters of the bilateral filter. The key observation is that if these $2D$ distances are large, ct "scales up" the argument $|ct(x) - ct(x + h)|$ to the filter, leading to a quick fall-off of filter weights, and preserving the structure in the input. The opposite happens for small distances. Generalizing to color images with three r, g, b channels, one can show that the above constraints on ct lead to the definition

$$ct(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \sum_{k \in r,g,b} |I'_k(x)| dx, \tag{8}$$

where I'_k is the derivative of the k-th color channel. In addition, $2D$ images can be filtered by iterating over several $1D$ passes.

In our application, we filter the disparity hypotheses scores obtained in the previous section using the color image of the reference view as a "guide" to define the domain transform function, which is similar to cross-bilateral filtering. We observed, however, that we can improve the quality of our filtered output by including the confidence C, Equation (7), from the previous Section. The intuition for including the confidence in the DTF is that if we found a clear winner among the disparity hypotheses at a pixel, meaning we get a high confidence value, the filter does not need to extend further. On the other hand, if we have low confidence in the winning disparity hypothesis, the filter should expand until we accumulated enough evidence.

We include the confidence into the DTF as an additional channel in the guide, forcing the filter support to stop where we have enough confidence. We achieve this by plugging the logarithm of Equation (7) into Equation (8),

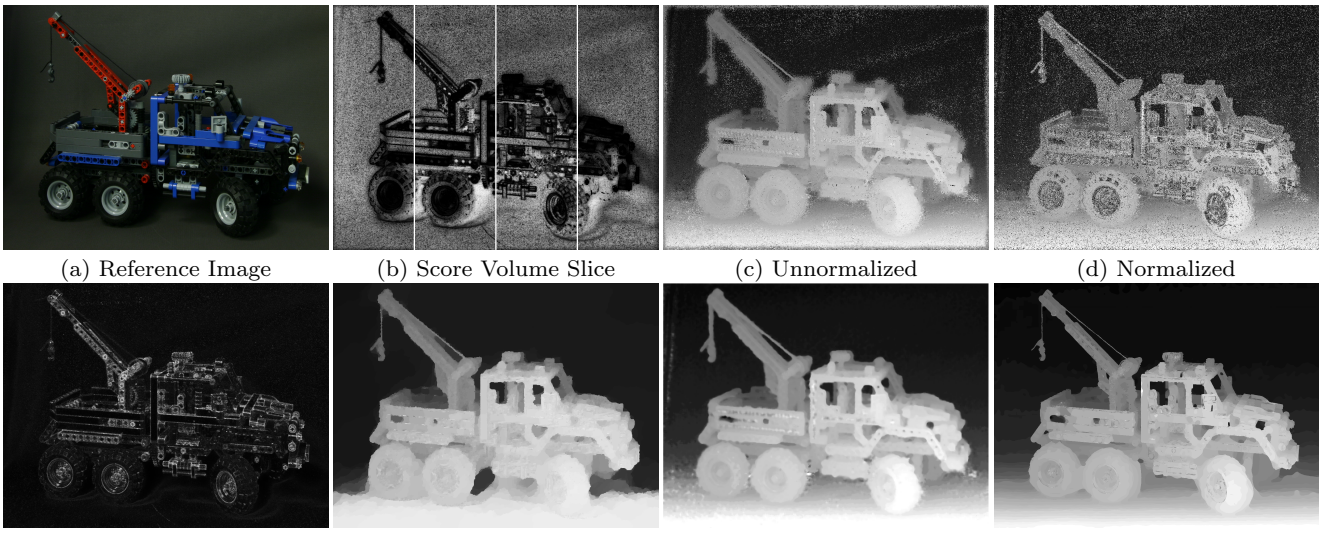
$$ct(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \sum_{k \in \{r,g,b\}} |I'_k(x)| + \frac{\sigma_s}{\sigma_c} \log(C(x)) \, dx. \tag{9}$$

Due to the non-linearity of our confidence estimate, we use $\log(C)$ as an upper bound on the confidence of the filtered score volume that will be accumulated by the filter. We can easily show that using the logarithm guarantees that the filter support never accumulates more than the user specified confidence σ_c. We use $\sigma_r = 178.5$, set σ_s to one fifth of the image width, and $\sigma_c = \log(D)$ to produce all our results.

After cost volume filtering, we select the disparity with the highest score in a winner-takes-all manner. We finally apply a bilateral median filter to remove remaining spike noise within a 9×9 block. To compute this weighted median, we calculate its bilateral weights [20] according to the corresponding colors in the reference image. Then a histogram is created using the computed weights as accumulation factor of the neighboring disparities. The median value of this histogram is assigned to the pixel's disparity. We compare our approach to two other recent methods [21, 7] in Figure 6.

5 Applications

In this section we present several applications of our reconstructed 3D light fields, most of them relying on disparity maps constructed as described above.



(a) Reference Image (b) Score Volume Slice (c) Unnormalized (d) Normalized

(e) Our confidence map (f) Wanner et al. [21] (g) Kim et al. [7] (h) Ours

Fig. 6 Visualization of the disparity map creation: (a) the reference view, (b) parts of four slices of the score volume S for disparity hypothesis -1.5, 0, 1.5 and 3 (from left to right), (c) the disparity with maximum score in S, (d) the disparity with the normalized scores, (e) the confidence map, (f) final result of [21], (g) final result of [7], (h) our final disparity map.

5.1 Refocusing using Synthetic Apertures

Shallow depth of field effects, as often used in professional portrait photography for example, are beyond the reach of devices like smart phones because of size restrictions on the optical design. Light fields acquired by translating a camera, however, make it possible to simulate synthetic apertures whose size is only limited by the range of camera translations. Light fields also facilitate digital refocusing after the fact, that is, changing the focal depth after image acquisition. We exploit our 3D light fields to achieve refocusing using potentially large synthetic apertures.

Given a $4D$ light field, it is straightforward to simulate a synthetic aperture by simply filtering over its two angular dimensions, where the filter represents the shape and extent of the desired aperture. The main challenge we need to overcome is that in our $3D$ light fields we only have one angular dimension, restricting synthetic apertures to horizontal $1D$ slits. We solve this problem by observing that we can model any separable $2D$ aperture as a superposition of vertical $1D$ apertures over the $1D$ angular domain of our $3D$ light fields. Hence, we use a two step procedure to obtain synthetic $2D$ apertures. First, for each view in our $3D$ light field we approximate the effect of the vertical $1D$ aperture. In the second step, we filter these processed views over the angular domain of the light field.

We leverage our disparity maps to compute the vertical $1D$ synthetic apertures using a depth-aware blur. We assume a two layer model consisting of a foreground and a background layer at each pixel, where the foreground contains all neighboring pixel closer to the camera, and the background all other pixels. We compute the colors for both layers separately, and blend them using alpha compositing. We obtain the depth-aware blur by splatting each foreground pixel to its vertical neighbors, where the splat size is given by the difference of the pixel's disparity to the disparity corresponding to the desired focal distance, and we use a $1D$ Gaussian splat kernel. More precisely, we splat the color of pixel $\mathbf{q}$ to a vertical neighbor $\mathbf{p}$ using the Gaussian weight

$$G(\mathbf{p}, \mathbf{q}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\|\mathbf{p}-\mathbf{q}\|^2}{2\sigma^2}}, \tag{10}$$

where the variance

$$\sigma_{\mathbf{q}} = \frac{a|d(\mathbf{q}) - d_f| + 1}{\sqrt{2\log(255)}} \tag{11}$$

is defined by the difference of the disparity $d(\mathbf{q})$ of pixel $\mathbf{q}$ to the disparity d_f of the object in focus and the user given aperture size a.

We compute the foreground color $F(\mathbf{p})$ of a pixel $\mathbf{p}$ by accumulating the splat contributions of all foreground pixels $\mathbf{q}$, that is, pixels with larger disparities than $\mathbf{p}$,

$$F(\mathbf{p}) = \frac{\sum_{\{\mathbf{q}|d(\mathbf{q})>d(\mathbf{p})\}} G(\mathbf{p}, \mathbf{q}, \sigma_{\mathbf{q}}) I(\mathbf{q})}{W(\mathbf{p})}, \tag{12}$$

where we normalize by the sum of the weights

$$W(\mathbf{p}) = \sum_{\{\mathbf{q}|d(\mathbf{q})>d(\mathbf{p})\}} G(\mathbf{p}, \mathbf{q}, \sigma_{\mathbf{q}}). \tag{13}$$

(a) Reference Image (b) Vertical $1D$ Aperture (c) Foreground in Focus (d) Background in Focus

Fig. 7 We create our synthetic aperture in two steps. First we enlarge the aperture vertically only (b). This we do for several views along the horizontal camera path. Summing them up extends the aperture horizontally (c). To get (d) we apply the same procedure by focusing on the background.

Note that the normalization weight $W(\mathbf{p})$ can be considered as an opacity value. We similarly compute the background color using all background pixels, that is, pixels with the same or smaller disparities than $\mathbf{p}$. Note that here we calculate the filter size according to the disparity of $\mathbf{p}$ for all background pixels. We do this because $\mathbf{p}$ itself belongs to the background, and it should not be splatted with colors from pixels which are behind it when $\mathbf{p}$ itself is in focus. Then,

$$B(\mathbf{p}) = \frac{\sum_{\{\mathbf{q}|d(\mathbf{q})\leq d(\mathbf{p})\}} G(\mathbf{q},\mathbf{p},\sigma_{\mathbf{p}})I(\mathbf{q})}{\sum_{\{\mathbf{q}|d(\mathbf{q})\leq d(\mathbf{p})\}} G(\mathbf{q},\mathbf{p},\sigma_{\mathbf{p}})}. \tag{14}$$

Finally, we composite the foreground and background using alpha blending with $\alpha = \min(1, W(\mathbf{p}))$,

$$V(\mathbf{p}) = \alpha F(\mathbf{p}) + (1-\alpha)B(\mathbf{p}), \tag{15}$$

where we clamp foreground coverage to one. We show an example in Figure 7(b).

Note that to apply the depth-aware blur to each light field view, we need a disparity map for each one. Instead of recomputing disparity maps for each view, we simply propagate the disparities from the reference image by following them to the other views. Hence, we propagate the disparity d of pixel (x, y) on the reference view m to pixel $(x + d(i - m), y)$ on the i-th disparity map. For pixels that receive several disparity values we keep the largest one, since this is the one belonging to the frontmost object. On the other hand, gaps will appear in background regions that were occluded in the reference view. We fill these holes with the lower disparity of its left respectively right border.

Once we computed all the vertically blurred views V_i, we shift them according to the in-focus disparity d_f and compute a weighted sum

$$I_{synthApp}(\mathbf{p}) = \sum_i G(i, m, \sigma)V_i(\mathbf{p_s}) \tag{16}$$

as the output image, where $\mathbf{p_s} = (x_s, y)$ with $x_s = x + (i-m)d_f$. We use again the Gaussian weights $G(i, m, \sigma)$, where i is the index of the view, m is the index of the reference image and $\sigma = (a + 1)/\sqrt{2\log(255)}$.

5.2 Further Applications

In this section we illustrate the usefulness of our processing pipeline by discussing further computational photography applications.

Foreground Removal. We can automatically remove thin foreground obstacles by exploiting our light field data and disparity map. This is useful to remove unwanted objects that may spoil a shot, as illustrated in Figure 8. Our approach is inspired by previous work that exploits light fields to "see through" foreground objects that partially occlude the scene behind [13]. The main idea is that digitally refocusing on a background layer using a very large synthetic aperture will make the foreground almost transparent. Since we have a disparity map at our disposal in addition to the light field, we are even able to completely disregard foreground objects based on their disparity when digitally refocusing on the scene behind. We simply mask out the disparity map using a threshold given by the disparity of the obstacle. Then we refocus the light field on the background and integrate only where the mask is non-zero. We apply the same disparity propagation to the non-central light field views as in Section 5.1.

Segmentation and Alpha Matting. We can use our disparity map to segment foreground objects by thresholding the disparities. The user sets the threshold simply by selecting the desired object. In addition, we obtain an alpha matte by filtering the resulting binary segmentation mask with the guided image filter as proposed by He et al. [8]. The filtering step produces a "guided feathering" effect where alpha values preserve detailed image structures while smoothly blending between foreground and background. Although algorithms for alpha matting using light fields have been proposed [15], we found that these approaches are less robust and more sensitive to parameter settings and scene characteristics.

We can also use the resulting segmentation and alpha matte to generate a selective gray scale effect where

(a) Reference Image (b) Synthetic Aperture (c) Large Synthetic Aperture (d) Infinite Aperture

Fig. 8 We show the user selected reference image in (a). In (b) and (c) we applied our synthetic aperture focusing on the background with aperture size 5 and 10, respectively. In (d) we show the result of our "infinite aperture" by removing the fence.

Fig. 9 An application of our computed alpha matte: For the reference image (top left) an alpha matte is generated for the foreground (top right). The object is inserted into the scene and blended with the foreground (bottom).

(a) Reference Image (b) Grayscale

Fig. 10 An application of image segmentation: pixels with a disparity value below a threshold are converted to gray scale.

the selected region stays colorful while we convert the rest of the scene to a gray scale image, as shown in Figure 10. Leveraging the disparity map, we can further provide functionality to insert new objects in the scene while respecting occlusions and performing alpha compositing with the foreground and background.

Multiview Autostereo Output. With the method from Section 3.4 we are able to render views from any point on the camera baseline. Hence it is straightforward to produce the appropriate views for autostereoscopic displays or lenticular prints. We adjust the zero-disparity plane to focus on desired scene elements by horizontally shifting the created views, where we read the required shift directly from the disparity map.

6 Mobile Application

To demonstrate the feasibility of a mobile app targeting advanced computational photography we implemented digital refocusing with synthetic apertures on iOS. The app lets the user record short movies and then processes the video frames as explained in Section 3. The user can then refocus the image as described in Section 5.1 using a touch gesture.

The iOS implementation shares most of the underlying source code with its desktop sibling, which keeps the porting effort at a minimum. To improve performance on the mobile device we vectorized the math-libraries using ARM NEON, perform more complex operations asynchronously to avoid freezing the user interface, and use an OpenGL ES 2 based off-screen renderer to increase the performance of our image-based warper (Section 3.4). Last but not least, we tuned all quality settings for speed to minimize the runtime complexity when computing synthetic apertures aimed at mobile device screen resolutions. This includes the number of tracked features[1], the number of rendered views (10), and the input frame resolution (720p).

We benchmarked our prototype on two devices, an iPhone 5 powered by Apple's ARM-v7s A6, and an iPad Air powered by Apple's ARM-v8 64bit A7. The results are shown in Table 1. Apparently, preprocessing the input material is the most time consuming part, notably feature detection, whereas refocusing is relatively quick. It is thus advisable to use as few frames as possible, and then to store the preprocessed data for later reuse. This enables us to provide a similar experience as with the Lytro light field picture files.

[1] Using `cv::goodFeaturesToTrack()`

	iPhone 5			
Movie	Preprocess	Disparity	Warp	Refocus
Fence	17.9s	44.7s	4.3s	8.8s
Rava	30.0s	41.4s	3.6s	5.7s
Yasmin	11.4s	41.4	4.1s	12.6s
	iPad Air			
	Preprocess	Disparity	Warp	Refocus
Fence	8.8s	18.3	2.2s	3.8s
Rava	14.1s	16.1	1.6s	2.5s
Yasmin	5.9s	16.5	1.8s	6.1s

Table 1 Results for the 2D synthetic aperture as explained in Section 5.1.

7 Conclusions and Future Work

We presented a method for hand-held 3D light field photography and described several computational photography applications enabled by our framework. The main advantage of our approach over previous techniques for capturing light fields using hand held devices is that it requires only a simple and short user interaction, making it practical for casual users. Our work hinges on a novel technique for spatio-temporal resampling of image sequences from approximately linear camera paths into regularly sampled $3D$ light fields. We also developed a novel disparity estimation technique leading to state-of-the-art results on standard datasets. Finally, we introduced a digital refocusing approach using synthetic apertures that leverages our light field data and disparity maps, and several other applications. We believe our approach opens exciting avenues for further computational photography applications on mobile devices. However, more low-level performance optimization is needed to provide a desirable level of interactivity on current devices.

Acknowledgements

This project was partially supported by funding from the Swiss *Commission for Technology and Innovation CTI* through project no. 15592.1 PFES-ES.

References

1. Petri Tanskanen, Kalin Kolev, Lorenz Meier, Federico Camposeco, Olivier Saurer, and Marc Pollefeys, "Live metric 3d reconstruction on mobile phones," in *IEEE ICCV*, December 2013, pp. 65–72.
2. Abe Davis, Marc Levoy, and Fredo Durand, "Unstructured light fields," *Computer Graphics Forum*, vol. 31, no. 2pt1, pp. 305–314, 2012.
3. Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala, "Subspace video stabilization," *ACM Trans. Graph.*, vol. 30, no. 1, pp. 4:1–4:10, Feb. 2011.
4. Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala, "Content-preserving warps for 3d video stabilization," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 44:1–44:9, July 2009.
5. Yu-Shuen Wang, Feng Liu, Pu-Sheng Hsu, and Tong-Yee Lee, "Spatially and temporally optimized video stabilization," *IEEE Trans. Vis. and Comp. Graph.*, vol. 19, no. 8, pp. 1354–1361, Aug. 2013.
6. C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *IEEE CVPR*, June 2011, pp. 3017–3024.
7. Changil Kim, Henning Zimmer, Yael Pritch, Alexander Sorkine-Hornung, and Markus Gross, "Scene reconstruction from high spatio-angular resolution light fields," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 73:1–73:12, July 2013.
8. Kaiming He, Jian Sun, and Xiaoou Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013.
9. Eduardo S. L. Gastal and Manuel M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 69:1–69:12, July 2011.
10. Ren Ng, "Fourier slice photography," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 735–744, July 2005.
11. Aaron Isaksen, Leonard McMillan, and Steven J. Gortler, "Dynamically reparameterized light fields," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 2000, SIGGRAPH '00, pp. 297–306, ACM Press/Addison-Wesley Publishing Co.
12. Nokia, "Refocus app," `https://refocus.nokia.com/`, Feb. 2014.
13. N. Joshi, S. Avidan, W. Matusik, and D. Kriegman, "Synthetic aperture tracking: Tracking through occlusions," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, Oct 2007, pp. 1–8.
14. Soonmin Bae and Frédo Durand, "Defocus magnification," *Computer Graphics Forum*, vol. 26, no. 3, pp. 571–579, 2007.
15. Neel Joshi, Wojciech Matusik, and Shai Avidan, "Natural video matting using camera arrays," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 779–786, July 2006.
16. Lippmann, G., "Épreuves réversibles donnant la sensation du relief," *J. Phys. Theor. Appl.*, vol. 7, no. 1, pp. 821–825, 1908.
17. Michal Irani, "Multi-frame correspondence estimation using subspace constraints," *Int. J. Comput. Vision*, vol. 48, no. 3, pp. 173–194, July 2002.
18. C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *IEEE CVPR*, Washington, DC, USA, 2011, CVPR '11, pp. 3017–3024, IEEE Computer Society.
19. Eduardo S. L. Gastal and Manuel M. Oliveira, "Domain transform for edge-aware image and video processing," in *ACM SIGGRAPH 2011 Papers*, New York, NY, USA, 2011, SIGGRAPH '11, pp. 69:1–69:12, ACM.
20. C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth International Conference on Computer Vision*, Washington, DC, USA, 1998, ICCV '98, pp. 839–, IEEE Computer Society.
21. S. Wanner and B. Goldluecke, "Globally consistent depth labeling of 4d light fields," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 41–48, June 2012.