The Visual Computer manuscript No.
(will be inserted by the editor)

Light Field Variational Estimation using a Light Field Formation Model: Supplementary Material

Julien Couillaud · Djemel Ziou

This supplementary material accompanies the paper "Light Field Variational Estimation using a Light Field Formation Model" published in *The Visual Computer*. It provides additional information, demonstrations, proofs, and results.

1 Light ray irradiance demonstration

A light ray is emitted by a scene point and passes through planes ST and UV located at a distance z_l from the camera. This light ray intersects plane ST at coordinates (s, t) and plane UV on a point P at coordinates (u, v), as illustrated in Fig.1. The light ray also crosses the camera aperture plane Ap at coordinates (a_u, a_v) and travels towards an image plane Ip located at a distance d_1 from Ap. The light ray intersection with Ip is located on a point p at coordinates (i_x, i_y), where its radiance is transformed into irradiance. To describe the radiance transformation into irradiance, some geometric features have to be defined. Let dO be a small area around P, da be a small area around the light ray intersection with Ap, and dI be a small area around p on Ip. The solid angles subtended by dO, da, and dI are, respectively, $\Delta\Omega O$, $\Delta\Omega$, and $\Delta\Omega I$. Moreover, we introduce three angles noted θ, α, and β. The first angle θ is the angle between the object normal at P and the light ray emitted by P. The second one α is the angle between the incident light ray and Ap normal. The third one β is the angle between the emerging light ray and Ap normal. Horn [7] writes the irradiance E at p as a ratio between the light power W emitted by P and dI. The same ratio is used to calculate the light ray irradiance, but with dW, the light power of a light ray emitted by P. This ratio is given by:

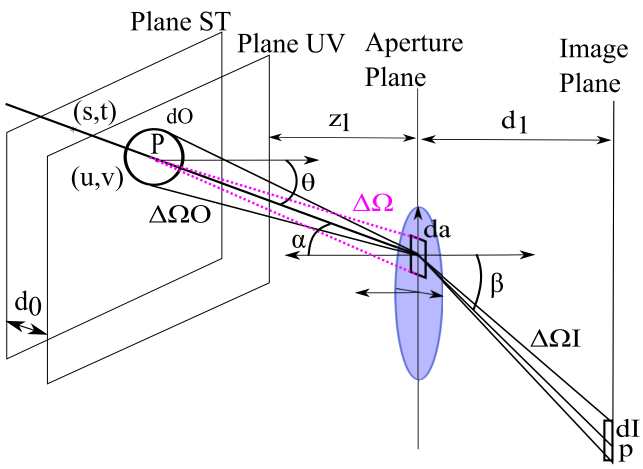

Fig. 1 Projection of a light ray from a light field into a camera

$$E = \frac{dW}{dI} \tag{1}$$

The light ray power dW depends on the radiance L, which a light ray emitted by P carries. It is determined by [7]:

$$dW = L dO \Delta\Omega \cos(\theta) \tag{2}$$

By using basic geometry principles, $\Delta\Omega$, dO, and dI are described by:

$$\Delta\Omega = \frac{da \cos(\alpha)^3}{z_l^2} \tag{3}$$

$$dO = \frac{\Delta\Omega O z_l^2}{\cos(\theta)\cos(\alpha)^2} \tag{4}$$

Julien Couillaud · Djemel Ziou
Université de Sherbrooke, MOIVRE,
2500, boulevard de l'Université Sherbrooke (Québec)
J1K 2R1
E-mail: julien.couillaud@usherbrooke.ca

$$dI = \frac{\Delta\Omega I d_1^2}{\cos(\beta)^3} \tag{5}$$

By replacing the terms dW, $\Delta\Omega$, dO, dI in (1) with their equations, the irradiance of a light ray emitted by P is given by:

$$E = L\frac{\Delta\Omega O}{\Delta\Omega I}\frac{\cos(\alpha)\cos(\beta)^3}{d_1^2}da \tag{6}$$

The solid angles $\Delta\Omega O$ and $\Delta\Omega I$ are assumed to be proportional. Indeed, these solid angles are not the same, but they could be proportional. Thus, the ratio $\frac{\Delta\Omega O}{\Delta\Omega I}$ is equal to a coefficient of proportionality k. This hypothesis simplifies the irradiance (6), which is rewritten as:

$$E = Lk\frac{\cos(\alpha)\cos(\beta)^3}{d_1^2}da \tag{7}$$

By using basic geometry principles, the angles α and β are determined by:

$$\alpha = \cos^{-1}\left(\frac{z_l}{\sqrt{(a_u-u)^2+(a_v-v)^2+z_l^2}}\right) \tag{8}$$

$$\beta = \cos^{-1}\left(\frac{d_1}{\sqrt{(i_x-a_u)^2+(i_y-a_v)^2+d_1^2}}\right) \tag{9}$$

Finally, da can be calculated by using the coordinates (a_u, a_v) described in the section 3.1 of the paper and given by:

$$\begin{cases} a_u = \frac{z_l+d_0}{d_0}u - \frac{z_l}{d_0}s \\ a_v = \frac{z_l+d_0}{d_0}v - \frac{z_l}{d_0}t \end{cases} \tag{10}$$

The equation (10) is derived from the Thales's theorem used on the triangles formed by a light ray, the planes ST and UV as well as a a straight line. This line is parallel to the optical axis and passes through the intersection of the light ray with the plane Ap. The unit area da can be described as the product of the unit length da_u and da_v on Ap. By deriving a_u and a_v with regard to u and v, da_u and da_v are determined by:

$$\begin{cases} da_u = \frac{z_l+d_0}{d_0}du \\ da_v = \frac{z_l+d_0}{d_0}dv \end{cases} \tag{11}$$

where the variable du and dv can be considered as unit lengths on the plane UV. Therefore, the irradiance (7) is rewritten as:

$$E = Lk\frac{\cos(\alpha)\cos(\beta)^3}{d_1^2}\left(\frac{z_l+d_0}{d_0}\right)^2 du\,dv \tag{12}$$

2 Light ray projection coordinates demonstration: the case of a thick lens

A thick lens is an optical device, which allows users to focus light rays in one point of a surface. This device is characterised by two principal planes, two focal points and two nodal points. The location of the principal planes Pl_1 and Pl_2 depends on features of the lens, such as its refraction index, its thickness, and its shape [10]. These planes are spaced by the effective lens thickness ρ, which may be negative [8]. The focal point F_1 (resp. F_2) is located on the optical axis in front of Pl_1 (resp. behind Pl_2) at a distance f_1 (resp. f_2), which corresponds to the front effective focal length (resp. rear effective focal length) of a thick lens. Finally, the nodal point N_1 (resp. N_2) is placed on the optical axis at a distance f_2 (resp. f_1) behind F_1 (resp. in front of F_2). We assume that the aperture plane Ap and Pl_1 are superimposed. Light rays, which are parallel to the optical axis and intersect Pl_1 (resp. Pl_2), pass through the lens, emerge from it in a new direction, and cross the focal point F_2 (resp. F_1). Moreover, a light ray intersecting a nodal point N_1 (resp. N_2) emerges from the lens with the same direction than the incident light ray, as if it was passing through N_2 (resp. N_1). By using these properties and basic geometry principles, one finds that, given the effective focal lengths f_1 and f_2, light rays originating from a scene point P_f, at a distance z_f from Ap, intersect an image plane Ip, at a distance d_1 from Ap, on a point p_f at coordinates (i_x, i_y), as shown in Fig. 2. Note that z_f is the depth of focus. The relationship between these distances is given by:

$$1 = \frac{f_2}{d_1-\rho} + \frac{f_1}{z_f} \tag{13}$$

In the scene configuration, illustrated in Fig. 2, the point P_f is located at coordinates (x_f, y_f) on a plane called plane of focus PF. The light ray passing through P_f originates from a light field parametrised by two planes ST and UV, spaced by a distance d_0 and placed at a distance z_l from the Pl_1. The light ray coordinates in the light field space are (s, t, u, v). In this configuration, the planes ST, UV, PF, Ap, Pl_1, and Ip are parallels and their coordinate system is the same. To find the intersection coordinates of a light ray with Ip, Thales's theorem is used on triangles, which are formed by the light ray, PF, and a straight line parallel to the optical axis, which passes through a point at coordinates (s, t) on the plane ST. These triangles are illustrated by green dotted lines in Fig. 2. Using this theorem, the following equation is determined:

$$\begin{cases} x_f - s = \frac{z_l-z_f+d_0}{d_0}(u-s) \\ y_f - t = \frac{z_l-z_f+d_0}{d_0}(v-t) \end{cases} \tag{14}$$

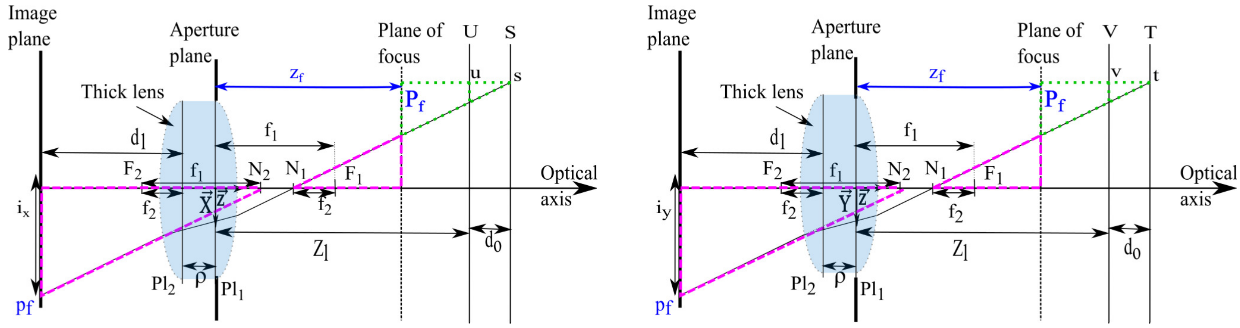

Fig. 2 Light ray projection inside a thick lens camera. The image (a) is a top view and (b) is a side view of the scene and the camera

Due to the thick lens properties, the coordinates of p_f are the projection coordinates on Ip of a light ray passing through P_f and the nodal point N_1. Using the property of light rays passing through a nodal point, we form two similar triangles. One of them is made of a light ray passing through P_f and N_1, the optical axis, and the plane of focus. The second one is composed of a part of the image plane, the optical axis, and a line passing through N_2 in the same direction than the light ray emitted by P_f and passing through N_1. These triangles are illustrated by magenta dotted lines in Fig. 2. By using Thales's theorem on these similar triangles, the projection coordinates of p_f are determined by:

$$\begin{cases} i_x = -\frac{d_1-\rho-f_2+f_1}{z_f-f_1+f_2} x_f \\ i_y = -\frac{d_1-\rho-f_2+f_1}{z_f-f_1+f_2} y_f \end{cases} \tag{15}$$

By combining (15) with (13) and (14), the light ray projection coordinates on Ip are described by:

$$\begin{cases} i_x = Ps - Qu \\ i_y = Pt - Qv \end{cases} \tag{16}$$

with

$$P = \frac{z_l\,(d1-\rho-f_2) - f_1\,(d_1-\rho)}{d_0 f_2} \tag{17}$$

and

$$Q = \frac{(z_l+d_0)\,(d1-\rho-f_2) - f_1\,(d_1-\rho)}{d_0 f_2} \tag{18}$$

3 Invertibility of $(\mathbf{H}^t\mathbf{H} - \gamma\mathbf{G})$: proof

For this demonstration, we first prove that $\mathbf{H}^t\mathbf{H}$ is invertible. Then, the invertibility of $(\mathbf{H}^t\mathbf{H} - \gamma\mathbf{G})$ is demonstrated. In the paper, $\mathbf{H}$ is a projection matrix of dimension $XY \times STUV$ and $\mathbf{H}^t$ is its transpose. In the thick lens projection geometry defined in (16), a light ray cannot be projected over two different points of the image plane. Moreover, the light ray sampling, modelled by $g()$ and specified by a unit impulse in (16), does not overlap several points of Ip. Thus, each point of Ip is struck by a unique set of light rays. Hence, each row of $\mathbf{H}$ contains a unique combination of positions and values of $\eta_{x,y,s,t,u,v}$. Therefore, each row of $\mathbf{H}$ is unique, so that the only vector solution to $\mathbf{H}\overrightarrow{\mathbf{x}} = \overrightarrow{\mathbf{0}}$ is $\overrightarrow{\mathbf{0}}$ and $\mathbf{H}$ has linearly independent columns. Let us look at the product $\mathbf{H}^t\mathbf{H}$, which forms a square matrix of dimensions $STUV \times STUV$. If the matrix $\mathbf{H}^t\mathbf{H}$ has linearly independent columns, then it is invertible. Let $\overrightarrow{\mathbf{v}}$ be a vector solution of:

$$\mathbf{H}^t\mathbf{H}\overrightarrow{\mathbf{v}} = \overrightarrow{\mathbf{0}} \tag{19}$$

If this equation is multiplied by $\overrightarrow{\mathbf{v}}^t$, we obtain:

$$\overrightarrow{\mathbf{v}}^t\mathbf{H}^t\mathbf{H}\overrightarrow{\mathbf{v}} = 0 \tag{20}$$

which is also equal to:

$$\left(\mathbf{H}\overrightarrow{\mathbf{v}}\right)^t\mathbf{H}\overrightarrow{\mathbf{v}} = \left\|\mathbf{H}\overrightarrow{\mathbf{v}}\right\|^2 = 0 \tag{21}$$

In (21), the vector $\overrightarrow{\mathbf{v}}$ must be the solution of $\left\|\mathbf{H}\overrightarrow{\mathbf{v}}\right\|^2 = 0$, where $\|.\|$ is the $L2$ vector norm. From the previous statements, we know that the only solution to $\left\|\mathbf{H}\overrightarrow{\mathbf{v}}\right\|^2 = 0$ is $\overrightarrow{\mathbf{0}}$. Hence, the only vector solution to $\mathbf{H}^t\mathbf{H}\overrightarrow{\mathbf{v}} = \overrightarrow{\mathbf{0}}$ is $\overrightarrow{\mathbf{0}}$. This solution means that the columns of $\mathbf{H}^t\mathbf{H}$ are linearly independent, so that $\mathbf{H}^t\mathbf{H}$ is invertible. Miller [9] proves that if a matrix $\mathbf{B}$ has a positive rank r, it can be decomposed in a sum of rank one matrices given by:

$$\mathbf{B} = \sum_{i=1}^{r} \mathbf{D_i} \tag{22}$$

where $\mathbf{D_i}$ is a rank one matrix. Miller also demonstrates that if the matrices $\mathbf{A}$ and $\mathbf{A} + \mathbf{B}$ are non-singular, then a non-singular matrix $\mathbf{C_{k+1}}$ is formed by $\mathbf{C_{k+1}} = \mathbf{A} + \mathbf{D_1} + ... + \mathbf{D_k}$ with $k = 1, ..., r$. Moreover, Miller

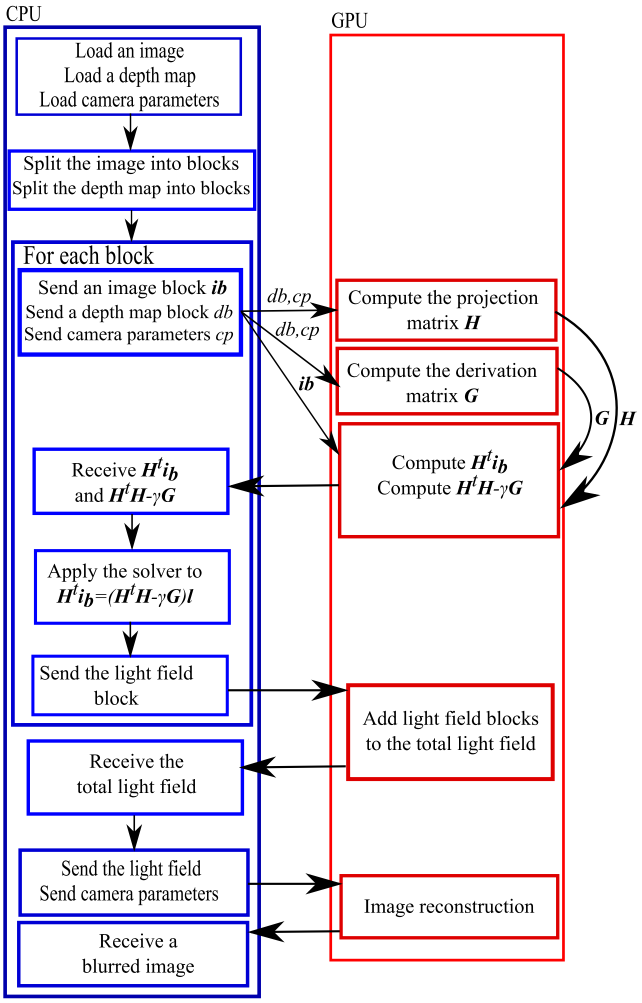

Fig. 3 Hybrid implementation architecture of the light field estimation algorithm using both *GPU* and *CPU*

shows in [9] that the inverse of the matrix $\mathbf{C_{k+1}}$ is equal to the following equation:

$$\mathbf{C_{k+1}}^{-1} = \mathbf{C_k}^{-1} + \mu_k \mathbf{C_k}^{-1}\mathbf{D_k}\mathbf{C_k}^{-1} \tag{23}$$

where $\mu_k = \frac{1}{1+tr(\mathbf{C_k}^{-1}\mathbf{D_k})}$ and $tr()$ is the trace of a matrix. This last equation is valid only if $\mathbf{C_1}$ equals $\mathbf{A}$ and $\mathbf{A}$ is invertible. In this theorem, $\mathbf{C_{r+1}}$ is equal to $\mathbf{A} + \mathbf{D_1} + ... + \mathbf{D_r}$, which is equal to $\mathbf{A}+\mathbf{B}$. Hence, $\mathbf{C_{r+1}}^{-1}$ is $(\mathbf{A}+\mathbf{B})^{-1}$ and $(\mathbf{A}+\mathbf{B})$ is invertible. Our implementation of the matrix $-\mathbf{G}$ of dimensions $STUV \times STUV$ in the paper equation (21) gives a matrix of rank six. Moreover, this matrix is non-singular. Hence, $-\mathbf{G}$ can be written as a sum of six non-singular rank one matrices $\mathbf{G_i}$. The matrix $(\mathbf{H}^t\mathbf{H} - \gamma\mathbf{G})$ can be written as:

$$\mathbf{M_7} = \mathbf{H}^t\mathbf{H} + \gamma\left(\mathbf{G_1} + ... + \mathbf{G_6}\right) \tag{24}$$

By using the theorem explained by Miller [9], $\mathbf{M_7}$ is invertible, so that $(\mathbf{H}^t\mathbf{H} - \gamma\mathbf{G})$ is invertible as well.

Table 1 Computation time (in seconds) of both *GPU/CPU* and sequential *CPU* implementations of the light field estimation for several directional resolutions $U \times V$. The speed up factor (SuF) of the *GPU/CPU* implementation with respect to the *CPU* one is also given.

Images	$U \times V$	3×3	5×5	7×7
K1	*GPU/CPU*	183.72	471.76	664.27
	CPU	27.31	353.20	855.35
	SuF	0.15	0.75	1.29
K2	*GPU/CPU*	181.71	473.28	696.37
	CPU	25.83	350.48	856.86
	SuF	0.14	0.74	1.23
K3	*GPU/CPU*	213.50	462.92	661.74
	CPU	26.64	351.90	856.85
	SuF	0.13	0.76	1.29
K4	*GPU/CPU*	184.26	452.14	680.79
	CPU	26.72	351.15	855.50
	SuF	0.15	0.78	1.26
L1	*GPU/CPU*	30.14	84.54	169.95
	CPU	5.6	74.10	182.25
	SuF	0.19	0.88	1.07
L2	*GPU/CPU*	34.94	79.45	154.11
	CPU	5.61	74.52	181.97
	SuF	0.16	0.94	1.18
L3	*GPU/CPU*	23.30	70.09	137.41
	CPU	5.70	74.73	182.51
	SuF	0.25	1.07	1.33
L4	*GPU/CPU*	40.98	88.44	166.18
	CPU	5.70	75.00	182.17
	SuF	0.14	0.85	1.10

4 Discussion on the value of the Lagrangian multiplier γ

In the paper, the choice of the Lagrangian multiplier γ is important because it determines the strength of the second requirement in paper equation (19). In the section 4.1 of the paper, we treated a non-zero value of γ. This section aims to explain why we ignore the case where γ equal to zero. If one sets γ to zero, only the reconstruction error $F_E(L)$ is considered in paper equation (19). In this case, the light field is estimated without using the scene depth information. Therefore, the estimated light field is composed of unregularised light rays which do not represent the scene geometry. By setting the value of γ above zero, the scene depth encoded in the second requirement (*i.e.* $F_C(L_s, L_t, L_u, L_v)$ in paper equation (18)) constrains the light field structure. Thanks to $F_C(L_s, L_t, L_u, L_v)$, all light rays emitted by a scene point are assumed to have a smooth radiance variation, which regularises their values and allows the light field to encode the scene geometry.

5 Intrinsic parameter calibration

Camera intrinsic parameters are determined from the manufacturers' data and calibration. The focal lengths $f1$ and $f2$, the aperture radius r, and the lens thickness ρ are found from the manufacturers' data. The length d_1 and the coefficient κ are found by using calibration methods and an optical bench. A camera is placed at a fixed position on the optical bench and a flat textured surface is placed in front of it. First, we determine the depth of focus by finding at which depth the camera captures the sharpest image of the textured surface. We assume that the sharpest image is placed at the depth of focus of the camera. The textured surface is iteratively moved of few millimeters along the optical bench and a picture of it is taken. Then the MES [15] is computed for each captured image. The MES is used as a sharpness index; the higher this index, the sharper the image is. The depth, at which the sharpest image of the textured surface is captured, is measured from the optical bench. Then equation (13) is used to calculate $d1$ from the depth of focus. The value of d_1 participates in the calculations of the irradiance model and the light ray projection in paper equations (11) and (13). Errors in the estimation of d_1 impact the light ray radiances estimated from the image and the light ray locations in the light field. Therefore, a wrong estimation of d_1 creates distortions in the estimated light field. To determine κ, several images of the textured surface are acquired. Each of these images have a different MES. In addition, their depth z_s is measured from the optical bench. For each image, we compute the standard deviation of a Gaussian point spread function, which models the image defocus blur as in [17]. We use the method presented in [16] to compute it. Moreover, the theoretical blur radius of each image is calculated from the following equation:

$$r_{defocus} = \| \frac{r}{f_2 z_s} d(z_s) \| \tag{25}$$

where

$$d(z_s) = \left(f_1 \left(d_1 - \rho\right) - \left(d_1 - f_2 - \rho\right) z_s\right) \tag{26}$$

Afterwards, the ratio between the standard deviation and the blur radius is calculated for each image. The coefficient κ, used in the paper section 5, is the mean of the ratios. Note that errors in the estimation of κ impact the defocus generated by the light field formation model in paper equation (10). Therefore, it influences the quantity of defocus blur which is left in the estimated light field.

Table 2 Computation time (in seconds) of both parallel *GPU* and sequential *CPU* implementations of the image reconstruction method for several directional resolutions $U \times V$. The speed up factor (*SuF*) of the *GPU* implementation with respect to the *CPU* one is also given.

Images	$U \times V$	3×3	5×5	7×7
K1	*GPU*	0.08	0.18	0.32
	CPU	5.50	13.47	27.40
	SuF	68.75	74.8	85.6
K2	*GPU*	0.08	0.16	0.34
	CPU	5.45	13.39	27.33
	SuF	68.1	83.7	80.4
K3	*GPU*	0.08	0.18	0.34
	CPU	5.63	13.34	27.45
	SuF	70.4	74.1	80.7
K4	*GPU*	0.08	0.18	0.33
	CPU	5.46	13.34	27.61
	SuF	68.3	74.1	83.7
L1	*GPU*	0.02	0.04	0.07
	CPU	1.16	2.78	5.69
	SuF	58.0	69.5	81.3
L2	*GPU*	0.02	0.04	0.07
	CPU	1.14	2.77	5.60
	SuF	57.0	69.3	80.0
L3	*GPU*	0.02	0.04	0.08
	CPU	1.15	2.76	5.61
	SuF	57.5	69.0	70.1
L4	*GPU*	0.02	0.04	0.08
	CPU	1.17	2.76	5.65
	SuF	58.5	69.0	70.6

6 Implementation details: Hybrid *GPU/CPU* light field estimation

In this section, we describes the implementation used to obtain the results presented in the paper. The dimensions of the sparse matrices used in the light field estimator, described in paper equation (21), are large, even if they contain a small quantity of non-zero elements. For example, for an image of 256×256 pixels and a light field of dimensions $256 \times 256 \times 11 \times 11$, the dimensions of $\mathbf{H}$ and $\mathbf{G}$ are 65536×7929856 and 7929856×7929856, while the minimum number of non-zero elements is 65536 for $\mathbf{H}$ and 23789568 for $\mathbf{G}$. Because of their large sizes, the matrices $\mathbf{H}$ and $\mathbf{G}$ cannot be stored in a personal computer memory even if a sparse matrix representation is used. To overcome this drawback, we propose to reduce their sizes by estimating light fields locally from image blocks. The resulting light fields are then combined in order to form an approximative light field of the whole scene.

The computational complexity for solving equation (21) is at least of $\mathcal{O}\left((STUV)^2\right)$. To fasten the computation, the light field estimation algorithm is implemented by using *GPU* and *CPU* together, as depicted in the implementation architecture in Fig. 3. We use the framework *Open Computing Language* (*OpenCL*) [13] for the

Table 3 Computation time (in seconds) of three sparse system solvers for different directional resolutions $U \times V$. The first solver (*CPUBICG*) is the biconjugate gradient implemented under *CPU* in *Eigen* library [6]. The second solver (*DIRECTLU*) corresponds to a *CPU* implementation of the unsymmetric-pattern multifrontal method from *Umfpack* library [4]. The third solver (*GPUBICG*) is a *GPU* implementation of the biconjugate gradient from *ViennaCL* library [11].

Images	$U \times V$	3×3	5×5	7×7
K1	*CPUBICG*	172.20	526.25	1483.76
	DIRECTLU	183.72	471.76	664.27
	GPUBICG	455.83	1514.33	10597.16
K2	*CPUBICG*	190.93	546.80	1555.10
	DIRECTLU	181.71	473.28	696.37
	GPUBICG	1277.23	3681.79	10571.52
K3	*CPUBICG*	189.32	534.43	1513.35
	DIRECTLU	213.50	462.92	661.74
	GPUBICG	1218.64	3967.59	14625
K4	*CPUBICG*	192.21	537.65	1494.62
	DIRECTLU	184.26	452.14	680.79
	GPUBICG	1214.7	4096.01	9150.23
L1	*CPUBICG*	31.50	101.31	310.96
	DIRECTLU	30.14	84.54	169.95
	GPUBICG	122.77	368.72	1438.3
L2	*CPUBICG*	32.71	99.42	293.98
	DIRECTLU	34.94	79.44	154.11
	GPUBICG	98.98	316.61	1082.43
L3	*CPUBICG*	23.65	86.19	286.95
	DIRECTLU	23.30	70.09	137.41
	GPUBICG	68.53	317.32	1071.3
L4	*CPUBICG*	35.96	104.93	311.70
	DIRECTLU	40.98	88.44	166.18
	GPUBICG	117.43	349.82	1230.38

parallel computations under the *GPU*. This framework provides a complete application programming interface, which allows parallel data processing and memory transfers between *CPU* and *GPU*. Each computation in *GPU* is encoded by using kernels written in *OpenCL language C*. The construction of the sparse matrices $\mathbf{H}$ and $\mathbf{G}$, basic sparse matrix operations, image reconstruction algorithms, and light field transformations are computed on *GPU* because they can be efficiently parallelised. In the light field estimator, the computation begins by gathering all the information needed, i.e. an image of a scene, a scene depth map, and parameters of the camera used to acquire the image. The image and the depth map are stored into matrices and read by using *openCV* library [2]. The camera parameters are all read from a file. Afterwards, the image and the depth map are split into blocks by using *OpenCV* features. We empirically found that small square blocks slow the light field estimation down because it increases the number of processed blocks and the time-consuming memory transfers between *GPU* and *CPU*. Therefore, we use the largest block size supported by the *GPU* memory. Note that the blocks used in the experiments, described in section 5 of the paper, have a size of 16×16 pixels. The projection matrix $\mathbf{H}$ is constructed by using two kernels which are executed one after the other. The first kernel looks for the light rays projected on a point p, at coordinates (x, y) on Ip, by using the projection $T(s, t, u, v)$ in (16). The second kernel reads the light ray coordinates (s, t, u, v) found by the first kernel and computes the weight $\eta_{x,y,s,t,u,v}$ associated with the light rays falling on p. Paper equation (16) describes the weight $\eta_{x,y,s,t,u,v}$. In these two kernels, the computation of each pixel of Ip is distributed in a thread. The matrix $\mathbf{G}$ is computed with one kernel, which writes a derivation mask for each light ray. The derivation masks are deduced from the explicit finite differences, which approximate the continuous derivatives in paper equation (20). In this implementation, the computation of each line of $\mathbf{G}$ is distributed in a thread. In addition, several algorithms for sparse matrix operations, among those in [12], have been implemented in kernels. After the *GPU* computations, data are transferred to the *CPU*, where the linear system in paper equation (21) is solved with the unsymmetric-pattern multifrontal method [4, 5]. This method requires a high memory load, so that it is processed on *CPU* by using *Umfpack* library [4]. Once the light field is estimated from an image block, a devoted kernel is used to add it to a total light field containing the light rays estimated from the whole image. We observe that estimation errors are high in the neighbourhood of occlusion. To remedy this, we use a light field inpainting algorithm. Iteratively, scene views are generated from the light field, holes near the occlusions are filled in by using an image inpainting method [14], and the radiance of inpainted scene points is propagated to the light rays emitted by those scene points and encoded in the light field. A scene view is generated from a light field by selecting a subspace $S \times T$ for fixed values of u and v. The light field inpainting algorithm stops once it has browsed all the scene views encoded into the light field or once no holes are left. Further below, we compare the computational speed of the proposed implementation with the one of a sequential implementation on *CPU*. In addition, we compare the computational time of three different solvers tested to obtain the solution of the linear system in paper equation (21).

7 Comparison of parallel and sequential light field estimation implementations

We propose to examine the hybrid implementation on *CPU/GPU* in contrast to a sequential implementation on *CPU*. In order to compare these implementations,

Table 4 MACADE and MΔE94 measured between ground truth images and images reconstructed from estimated light fields for two types of camera models. The images are reconstructed from light fields estimated by using either the thick lens or thin lens projection geometries. The lens thickness is set to zero for the thin lens model, while the thickness for the thick lens model is provided in the Table 1 of the paper. The ground truth images, depicted in Fig. 4 of the paper, are used as inputs of the light field estimator. The value of γ is equal to 1.

Camera types	Images	*K1*	*K2*	*K3*	*K4*	*L1*	*L2*	*L3*	*L4*
Thick lens	MACADE	0.99	0.99	0.98	0.96	0.99	0.98	0.99	1.0
	MΔE94	1.54	2.72	2.20	1.97	2.26	2.56	1.38	1.09
Thin lens	MACADE	0.98	0.98	0.98	0.96	0.99	0.99	1.0	1.0
	MΔE94	1.63	2.83	2.22	1.98	2.0	2.27	1.12	1.01

we use the computational time as a performance measure. Table 1 summarises the light field estimation computation time of both *CPU/GPU* and sequential *CPU* implementations. As a direct observation of Table 1, we notice that for most images, the *CPU* implementation is faster than the *GPU/CPU* implementation for directional resolutions of 3×3 and 5×5. This performance is due to the time-consuming memory transfers between the *CPU* and the *GPU*. However, the *GPU/CPU* implementation is faster than the *CPU* one when the directional resolution is equal to 7×7 with an average speed up factor of 1.23. When the light field size is large, the parallel calculations under *GPU* are so quick that even with slow memory transfers, the *GPU/CPU* implementation is faster than the sequential *CPU* one. In addition, we compare a sequential *CPU* and a parallel *GPU* image reconstruction implementation. The image reconstruction method corresponds to the application of paper equation (10) on an estimated light field. The computation time for both parallel *GPU* and sequential *CPU* image reconstruction implementations are summarised in Table 2. The light fields used in the image reconstruction method corresponds to the output of the light field estimation implementation on *GPU/CPU*. The measures in Table 2 show that the *GPU* implementation is about 71 times faster than the *CPU* one.

8 Impact of the linear system solver on the implementation of the light field estimation

In this section, we compare three implementations of the light field estimation on *GPU/CPU*. Each implementation applies a different sparse linear solver to the linear system in the paper equation (21). The computational time of these various implementations is used as a performance measure. The compared solvers correspond to: a biconjugate gradient method with a *CPU* implementation (*CPUBICG*) from *Eigen* library [6], a *CPU* implementation (*DIRECTLU*) of the unsymmetric-pattern multifrontal method from *Umfpack* library [4], and a *GPU* implementation (*GPUBICG*) of a biconjugate gradient method from the library *ViennaCL* [11]. The computation time of the solvers are summarised in Table 3. The *GPUBICG* solver is always slower than the two other algorithms due to intensive memory transfers between *CPU* and *GPU*, which slow the light field estimation down. For small directional resolutions, the computation times for *CPUBICG* and *DIRECTLU* are close to each other, but for higher resolutions, *DIRECTLU* is faster than *CPUBICG*. Therefore, we choose the *DIRECTLU* solver for the light field estimation implementation proposed in section 6.

9 Comparison between the thin lens and thick lens projection geometries

In section 3.3 of the paper, we propose a thick lens camera model and we use it to estimate light fields. However, in other studies [1,3], the thin lens model is usually employed. In this section, we propose to evaluate if one of these lens models is better than the other one. To achieve this, light fields are estimated from ground truth images captured with *Kinect* and *leanXcam* cameras. Then images are reconstructed from the estimated light fields and compared with their ground truths. The ground truth images are illustrated in Fig. 4 of the paper. We use the same methodology and performance measures, *i.e.* MACADE and MΔE94, than the ones employed in the sections 5.2, 5.3, and 5.4 of the paper. To simulate the thin lens, the lens thickness is set to zero and $f_1 = f_2$, while the thickness is non-zero and f_1 can be different than f_2 for the thick lens. The intrinsic parameters used to simulate these lenses are detailed in Table 1 of the paper. Table 4 summarises the measured MACADE and MΔE94 for the thick lens and thin lens camera models. From these measures, we can see that the MACADE as well as the MΔE94 determined for the thick lens have negligible differences with the ones evaluated from the thin lens. This result is explained by the lens thickness of the used camera, which is so small that we can neglect it in the thick lens projection (13) described in the paper. We cannot state that one of these models is better than the other as both produce similar results with the used cameras. One of the strengths of the thick lens model is that it models the thin lens under some special configurations, but it can also represent other types of lenses. For example, it can model lenses with a significant thickness or with different focal lengths. Therefore, the thick lens is much more flexible than a thin lens as it allows users to model many more types of lenses.

References

1. Bishop, T.E., Favaro, P.: The light field camera: Extended depth of field, aliasing, and superresolution. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(5), 972–986 (2012)
2. Bradski, G.: The openCV library. Dr. Dobb's Journal of Software Tools **25**(11), 120–126 (2000)
3. Chia-Kai, L., Yi-Chang, S., Chen, H.: Light field analysis for modeling image formation. IEEE Transactions on Image Processing **20**(2), 446–460 (2011)
4. Davis, T.A.: Algorithm 832: UMFPACK V4.3 - an unsymmetric-pattern multifrontal method. ACM Transactions on Mathematical Software **30**(2), 196–199 (2004)
5. Davis, T.A., Duff, I.S.: An unsymmetric-pattern multifrontal method for sparse LU factorization. SIAM J. Matrix Analysis and Applications **18**(1), 140–158 (1997)
6. Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)
7. Horn, B.K.: Robot Vision, 1 edn. McGraw-Hill Higher Education (1986)
8. Kolb, C., Mitchell, D., Hanrahan, P.: A realistic camera model for computer graphics. In: Proc. of the 22nd Annual Conf. on Computer Graphics and Interactive Techniques, pp. 317–324 (1995)
9. Miller, K.S.: On the inverse of the sum of matrices. Mathematics Magazine **54**(2), 67–72 (1981)
10. Ray, S.: Applied Photographic Optics, 3rd Edition, 3 edn. Focal Press (2002)
11. Rupp, K., Rudolf, F., Weinbub, J.: ViennaCL - a high level linear algebra library for GPUs and multi-core CPUs. In: International Workshop on GPUs and Scientific Applications, pp. 51–56 (2010)
12. Spagnoli, K.E., Humphrey, J.R., Price, D.K., Kelmelis, E.J.: Accelerating sparse linear algebra using graphics processing units. Proc. SPIE **8060**, 4–9 (2011)
13. Stone, J.E., Gohara, D., Shi, G.: OpenCL: A parallel programming standard for heterogeneous computing systems. IEEE Design & Test **12**(3), 6–73 (2010)
14. Telea, A.: An image inpainting technique based on the fast marching method. Journal of Graphics Tools **9**(1), 23–34 (2004)
15. Wang, X., Tian, B., Liang, C., Shi, D.: Blind image quality assessment for measuring image blur. In: Cong. on Image and Signal Processing, vol. 1, pp. 467–470 (2008)
16. Zhuo, S., Sim, T.: Defocus map estimation from a single image. Pattern Recognition **44**(9), 1852–1858 (2011)
17. Ziou, D., Deschnes, F.: Depth from defocus estimation in spatial domain. Computer Vision and Image Understanding **81**(2), 143 – 165 (2001)