Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2010, Article ID 942037, 12 pages
doi:10.1155/2010/942037

Research Article

Near Real-Time Estimation of Super-Resolved Depth and All-In-Focus Images from a Plenoptic Camera Using Graphics Processing Units

J. P. Lüke,[1] F. Pérez Nava,[2] J. G. Marichal-Hernández,[1] J. M. Rodríguez-Ramos,[1] and F. Rosa[1]

[1] *Departamento de Física Fundamental y Experimental, Electrónica y Sistemas, Universidad de La Laguna, 38203 Canary Islands, Spain*
[2] *Departamento de Estadística, Investigación Operativa y Computación, Universidad de La Laguna, 38203 Canary Islands, Spain*

Correspondence should be addressed to J. P. Lüke, jpluke@ull.es

Received 30 April 2009; Revised 4 September 2009; Accepted 14 September 2009

Academic Editor: Xenophon Zabulis

Depth range cameras are a promising solution for the 3DTV production chain. The generation of color images with their accompanying depth value simplifies the transmission bandwidth problem in 3DTV and yields a direct input for autostereoscopic displays. Recent developments in plenoptic video-cameras make it possible to introduce 3D cameras that operate similarly to traditional cameras. The use of plenoptic cameras for 3DTV has some benefits with respect to 3D capture systems based on dual stereo cameras since there is no need for geometric and color calibration or frame synchronization. This paper presents a method for simultaneously recovering depth and all-in-focus images from a plenoptic camera in near real time using graphics processing units (GPUs). Previous methods for 3D reconstruction using plenoptic images suffered from the drawback of low spatial resolution. A method that overcomes this deficiency is developed on parallel hardware to obtain near real-time 3D reconstruction with a final spatial resolution of 800×600 pixels. This resolution is suitable as an input to some autostereoscopic displays currently on the market and shows that real-time 3DTV based on plenoptic video-cameras is technologically feasible.

1. Introduction

Since the introduction of television, a great effort has been made to improve the overall experience for viewers. Improvements in color, picture quality, and sound quality have contributed to an enhanced viewing experience. It is believed that three-dimensional television (3DTV) that enables people to watch content in three dimensions is the next step in the evolution of television. Proponents of 3DTV have argued that it will bring the viewer a more natural and life-like visual home entertainment experience [1]. Although there have been several attempts to initiate 3DTV [2] over the last decades, its introduction is considered as becoming increasingly feasible thanks to recent technologies and advances in the fields of camera development, image processing, and display design.

A complete technological solution for 3DTV must consider the whole production chain involving content creation, coding, transmission, and display [3]. In contrast to traditional television, 3DTV has to capture multiview information about the scene in order to reconstruct 3D information. Most 3D material is shot using a dual-camera configuration or using an array of cameras. The storage and transmission of this 3D material involves a large amount of data because one stereoscopic image consists of multiple views. Therefore, a considerable effort is involved in compressing the digital images in order to obtain savings in bandwidth and storage capacity. This is of particular relevance in the case of 3D HDTV, where a single uncompressed HDTV channel may consume up to one Gbit/s transmission bandwidth, or in the case of 3D video transmission over low-bandwidth transmission channels, such as the Internet [4, 5]. In terms of compatibility with existing broadcast systems, double the bandwidth would be needed for transmitting the left- and right-eye views of a dual camera. The use of a depth range camera recording the RGB image and accompanying

depth value per pixel (2D plus depth format) overcomes this bandwidth problem. There are several developments in this area: active cameras like Axi-Vision by NHK [6] or Z-cam by 3DV [7] or passive cameras like the CAFADIS plenoptic video-camera [8, 9]. Although this is a promising area, there are still some challenges to accurately recovering depth with enough spatial and range resolution such that the viewing comfort is at least comparable to that of conventional 2D television. Transmitted images are shown on stereoscopic or autostereoscopic displays [10]. Stereoscopic displays require the viewer to wear an optical device (e.g., polarized glasses, shutter glasses) to direct the left and right eye images to the appropriate eye, while the separation technique used in autostereoscopic displays is integrated into the display screen.

The development of autostereoscopic displays that accept the standard 2D plus depth format [11] makes possible a simple solution for the 3DTV production chain based on the recently developed depth range cameras. This solution is backward compatible with conventional broadcast television and ensures a gradual transition from one system to the other.

This paper addresses the creation of 3DTV content using a depth range plenoptic video-camera. It is clear that the introduction of 3DTV is conditioned on the existence of enough 3D content that can be displayed to viewers. The content creation process requires the capacity to capture scenes realistically or to recreate them using computer generated graphics. The capture of real scenes in 3DTV requires a significant technological effort. Multiview information has to be obtained with an array of two or more cameras and has to be processed in order to obtain the necessary 3D information. The use of camera arrays is not a portable solution and can thus hinder the filming process. Another disadvantage of camera arrays is the need for geometric and color calibration or frame synchronization between the cameras. It would be desirable for 3D capture to be done with a device of dimensions similar to those of traditional TV cameras. One such device for obtaining multiview information and which does not require calibration is a plenoptic video-camera [8, 9]. Plenoptic cameras use a microlens array behind the main lens to capture multiple views of the scene, generating a lightfield image that records the radiance and direction of all light rays in the scene. Using the lightfield image, it is possible to generate the focal stack using several approaches [12–15]. The focal stack serves as input for multiview stereo depth estimation algorithms. Using the estimated depth and the focal stack, it is straightforward to obtain the all-in-focus image and depth map of the scene.

An important drawback of plenoptic cameras is the generation of low-resolution images due to their spatioangular lightfield sampling [16]. The angular information recorded results in a small output image being produced in comparison with sensor size. Methods based on superresolution techniques that overcome this spatial resolution limitation have recently been presented [17, 18]; however, such methods are much too time intensive to be executed in real time on an ordinary CPU. A solution for achieving the real-time requirements of 3DTV is the use of parallelization techniques

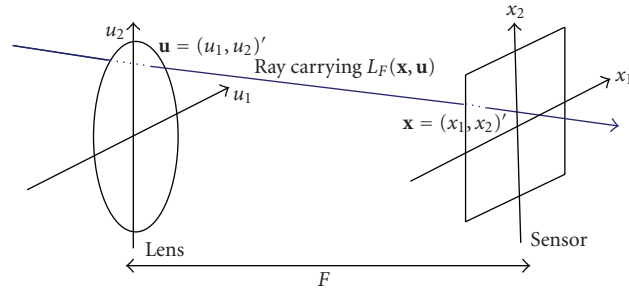


Figure 1: Two-plane parameterization of the lightfield.

that increase the computational processing power. In recent years, graphics processing units (GPU) have experienced enormous change, becoming parallel coprocessors that can be used to perform general purpose computations with a significant speedup [19]. In this article we describe an implementation on GPUs of the super-resolution and distance extraction algorithm presented in [18].

The rest of this paper is divided as follows: in Section 2 we present the super-resolution technique that obtains all-in-focus and depth images. Details of multi-GPU implementation are explained in Section 3. Section 4 shows some experimental results and finally Section 5 contains some concluding remarks and future work.

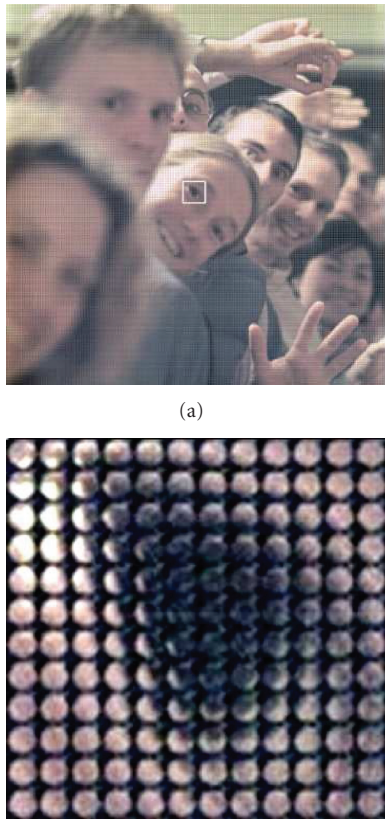
2. Simultaneous Superresolution Depth and All-In-Focus Images from Plenoptic Cameras

In this section we present a super-resolution technique [18] to produce depth and all-in-focus images from plenoptic cameras. This technique is based on the super-resolution discrete focal stack transform (SDFST) that generates a focal stack that is processed to estimate 3D depth. This estimated depth is then used to obtain the all-in-focus image. We present the entire process in this section.

2.1. The Focal Stack Transform (FST). The SDFST is an extension of the Focal Stack Transform [12, 13] that processes a lightfield image from a plenoptic camera and generates images focused on a desired frontal plane. The FST is based on the Photography transform [12] that generates conventional 2D photographs from a lightfield. To introduce the Photography transform it is necessary to parameterize the lightfield defined by all the light rays inside a camera. Using the two-plane parameterization we write $L_F(\mathbf{x}, \mathbf{u})$ as the radiance traveling from position $\mathbf{u} = (u_1, u_2)'$ (apostrophe means transpose) on the lens plane to position $\mathbf{x} = (x_1, x_2)'$ on the sensor plane. F is the distance between the lens and the sensor (see Figure 1 adapted from [12]).

The lightfield L_F can be used to compute conventional 2D photographs at any depth αF. Let $\mathcal{P}_\alpha$ be the operator that transforms a lightfield L_F at a sensor depth F into a photograph at sensor depth αF, then the Photography operator can be written as [12]

$$\mathcal{P}_\alpha[L_F](\mathbf{x}) = \frac{1}{\alpha^2 F^2} \int L_F\left(\mathbf{u}\left(1 - \frac{1}{\alpha}\right) + \frac{\mathbf{x}}{\alpha}, \mathbf{u}\right) d\mathbf{u}. \quad (1)$$



(a)

(b)

Figure 2: (a) Lightfield captured by a plenoptic camera. (b) Detail from the white square in the left image.

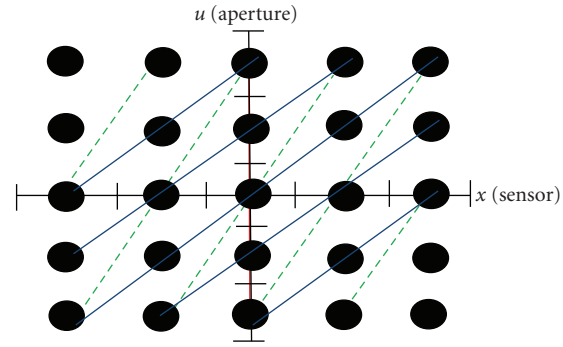


Figure 3: Photography transform for two particular α values.

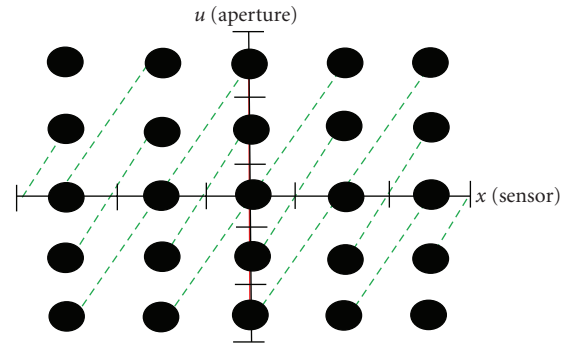


Figure 4: The lines used in the SDFST for a particular distance.

This equation shows how to compute $\mathcal{P}_\alpha[L_F]$ at different depths from the lightfield L_F. When photographs are computed for every sensor depth αF, we obtain the focal stack transform $\mathcal{I}$ of the lightfield defined as [13]

$$\mathcal{I}[L_F](\mathbf{x}, \alpha) = \mathcal{P}_\alpha[L_F](\alpha\mathbf{x}). \tag{2}$$

A plenoptic camera only captures discrete samples of the lightfield. We will assume that the plenoptic camera is composed of $n \times n$ microlenses and each of them generates an $m \times m$ image. A discrete lightfield captured from a plenoptic camera is shown in Figure 2 [12]. To obtain a discretized version of the focal transform it is necessary to interpolate the lightfield L_F and to discretize the integral in (1). There are several techniques for performing this interpolation [12–15]. A common result of these techniques is that the focal stack is composed of images with $n \times n$ pixels. Several focal stacks can be built depending on the application. We can use the radiance in the lightfield L_F to obtain the usual focal stack, measures of focus to obtain the Laplacian focal stack [13, 15], or measures of photoconsistency to obtain the variance focal stack.

To introduce the super-resolution focal stack it is necessary to show how to compute the Photography transform on a discretized lightfield. To simplify the presentation, the super-resolution technique is shown in 2D instead of 3D. The extension of the results to 3D is straightforward. In the 2D case, to compute the Photography transform (1) at sensor depth αF it is necessary to compute a line integral on the lightfield. For each of the n pixels in the x axis, a sensor depth αF generates a line through x whose slope is determined by α (see Figure 3). Since the line is continuous, to approximate the integral we have to sum all the values on the line and interpolate where pixels are not available.

2.2. The Superresolution Discrete Focal Stack Transform . The super-resolution extension to the Focal Transform (SDFST) is based on performing the Focal Stack Transform for a different set of lines. The selection of this new set of lines is explained in Figure 4.

The new set of lines for each slope is an extension of the initial set with lines of the same slope. This new set passes through every point in the discretized lightfield, not only through points on the x-axis. For the example above, intersecting these lines with the x-axis we see that we now have twice the resolution as before (see the dashed lines of Figures 3 and 4). The super-resolution algorithm only uses the samples that lie exactly on each line and obtains the extended resolution using the following theorem and corollary [18].

Theorem 1. *Given a lightfield* $L_F(x, u)$, $|x| \leq r|u| \leq s$, $n = 2r + 1$, $m = 2s + 1$, $0 < |\Delta x| \leq r$, $0 < |\Delta u| \leq s$, Δu, $\Delta x \in Z$, *if we pad* $L_F(x, u)$ *with* $|\Delta x_g|s$ *zero-columns on each side of the x-axis, the super-resolution focal stack image for slope* $\Delta u/\Delta x$

Input: Lightfield $L_F(\mathbf{x}, \mathbf{u})$, $\mathbf{x} = (x_1, x_2)'$, $\mathbf{u} = (u_1, u_2)'$, $|x| \leq r$ $|u| \leq s$.
Slope $\Delta u/\Delta x$ with $\Delta u, \Delta x \in Z$ as in the previous theorem
Output: Super-resolution focal stack image $F_{\Delta u/\Delta x}(\mathbf{k})$
Compute g =g.c.d.$(\Delta u, \Delta x)$, $\Delta u_g = \Delta u/g$ and $\Delta x_g = \Delta x/g$
For each microlens $\mathbf{x} = (x_1, x_2)'$
  For each pixel in that microlens $\mathbf{u} = (u_1, u_2)'$
    Compute $\mathbf{k} = (k_1, k_2)'$, $\mathbf{k} = \Delta u_g \mathbf{x} - \Delta x_g \mathbf{u}$
    Update the mean value in $F_{\Delta u/\Delta x}(\mathbf{k})$ using $L_F(\mathbf{x}, \mathbf{u})$

Algorithm 1

has $|\Delta u_g| n + |\Delta x_g| m - |\Delta u_g| - |\Delta x_g| + 1$ *points with* $\Delta u_g = \Delta u/g$ *and* $\Delta x_g = \Delta x/g$, *where* g *is the greatest common divisor (g.c.d.) of* $|\Delta u|$ *and* $|\Delta x|$.

Corollary 2. *Under the conditions of the preceding theorem the collection of slopes* $\{\Delta u/\Delta x \; 0 < |\Delta x| < s, \Delta u = s, \Delta u, \Delta x \in Z$ *with* s *a prime number}* *can generate a super-resolution focal stack with* $\Delta u n - \Delta u + 1$ *pixels in each image.*

Therefore, the final resolution is approximately $\Delta u n \approx mn/2$, that is, half the full resolution of the sensor. The extension of the above results to 3D using squared aperture and microlenses is trivial. The final resolution in that case is approximately the full resolution divided by 4. In actual plenoptic images the resolution is somewhat lower due to the circular shape of microlenses and their edge effects. The algorithm for computing an image of the super-resolution focal stack is shown in Algorithm 1 [18]. In Algorithm 1, $\mathbf{x} = (x_1, x_2)'$ and $\mathbf{u} = (u_1, u_2)'$ are the coordinates of the microlens and the coordinates of the pixel within its microlens, respectively. g is the g.c.d.$(\Delta u, \Delta x)$. As shown in the previous theorem, in order to maximize the spatial resolution the value of g must be equal to 1. Each plane of the focal stack has associated with it a slope $\Delta u/\Delta x$ of integration so that Algorithm 1 has to be executed for each plane. $\Delta u, \Delta x \in Z$ are the discrete points in the sampled lightfield. $\mathbf{k} = (k_1, k_2)'$ is the coordinate of a pixel in the focal stack plane $F_{\Delta u/\Delta x}$ where lightfield data is accumulated to finally give the photograph focused at the depth associated with the slope $\Delta u/\Delta x$.

The computational complexity for the complete super-resolution focal-stack in the 3D case is $O(n^2 \times m^3)$.

The result of the algorithm for a region of the lightfield in Figure 2 is shown in Figure 5. The algorithm computes the super-resolved focal stack, generating several photographs of the same object focused at different depths. The resulting images have a 25× spatial resolution magnification with respect to previous approaches based on plenoptic cameras [12–15].

Figure 5: Results of the super-resolution focal stack with a 25× resolution magnification.

2.3. Multiview Stereo over the Superresolution Focal Stack. The multiview stereo algorithm operates on the variance focal stack and is able to obtain a $t \times t$ estimation of depth ($t = \Delta u n - \Delta u + 1$) with $m - 3$ different depth values and a $t \times t$ all-in-focus image. The variance focal stack is based on the photoconsistency assumption (PA) which states that the radiance of rays from a 3D point over all directions is the same (the Lambertian model). To test the PA assumption the variance of all points along every line is measured (see Figure 4). The multiview stereo algorithm uses this variance measure to choose the optimal depth. A straightforward approach for solving the multiview stereo problem would be to choose the line with less variance for each pixel. However it is well known that more assumptions are necessary if good results are to be obtained. The assumption that surfaces are smooth makes it possible to use the variance focal stack and the Markov Random Field (MRF) approach to obtain the optimal depth by minimizing an energy function [20]. The energy function is composed of a data term E_d and a smoothness term E_s, $E = E_d + \lambda E_s$, where the parameter λ measures the relative importance of each term. The data term is simply the sum of the per-pixel data costs, $E_d = \sum_{\mathbf{p}} c_{\mathbf{p}}(d)$, where $c_{\mathbf{p}}(d)$ is the variance measure for pixel $\mathbf{p}$ in the superresolved image and $d = \Delta u/\Delta x$ is a specific slope (see Figure 4). The smoothness term is based on the 4-connected neighbors of each pixel and can be written as $E_s = \sum_{\mathbf{p},\mathbf{q}} V_{\mathbf{pq}}(d_{\mathbf{p}}, d_{\mathbf{q}})$ where $\mathbf{p}$ and $\mathbf{q}$ are two neighboring pixels. $V_{\mathbf{pq}}(d_{\mathbf{p}}, d_{\mathbf{q}})$ is defined as

$$V_{\mathbf{pq}}\left(d_{\mathbf{p}}, d_{\mathbf{q}}\right) = \begin{cases} 0, & \text{if } d_{\mathbf{p}} = d_{\mathbf{q}}, \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

The hierarchical belief propagation (BP) approach [20] is used for optimizing the energy function E. This algorithm passes messages around the 4-connected image grid in an

iterative process using the following message updating rule:

$$M^{i}_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{q}}\right) = \min_{d_{\mathbf{p}}}\left(h_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{p}}\right) + \lambda \cdot V_{\mathbf{pq}}\left(d_{\mathbf{p}}, d_{\mathbf{q}}\right)\right), \quad (4)$$

where $M^{i}_{\mathbf{p}\to\mathbf{q}}(d_{\mathbf{q}})$ is the message passed from pixel $\mathbf{p}$ to pixel $\mathbf{q}$ for disparity $d_{\mathbf{q}}$ at iteration i and

$$h_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{p}}\right) = c_{\mathbf{p}}\left(d_{\mathbf{p}}\right) + \mu \sum_{\mathbf{s}\in(\mathbf{p})} M^{i-1}_{\mathbf{s}\to\mathbf{p}}\left(d_{\mathbf{p}}\right) - M^{i-1}_{\mathbf{q}\to\mathbf{p}}\left(d_{\mathbf{p}}\right), \quad (5)$$

where $N(\mathbf{p})$ is the four-connected neighborhood of pixel $\mathbf{p}$ and $\mu \in (0, 1]$. Once the iterative process has finished after a fixed number of I iterations, the belief vector for every pixel is computed as

$$b_{\mathbf{q}}\left(d_{\mathbf{q}}\right) = c_{\mathbf{q}}\left(d_{\mathbf{q}}\right) + \mu \sum_{\mathbf{p}\in N(\mathbf{q})} M^{I}_{\mathbf{q}\to\mathbf{p}}\left(d_{\mathbf{q}}\right), \quad (6)$$

and the value of $d_{\mathbf{q}}$ that leads to the minimum of $b_{\mathbf{q}}(d_{\mathbf{q}})$ is selected as the optimal depth.

The general belief propagation algorithm needs $O(k^2)$ computing time per message, where k is the number of labels in each pixel ($m - 3$ depth levels in our case). Several techniques can be used to decrease the computing and memory requirements of the algorithm [21]. Messages can be computed in $O(k)$ time by taking into account the particular structure of $V_{\mathbf{pq}}(d_{\mathbf{p}}, d_{\mathbf{q}})$ function and rewriting the message update rule as

$$M^{i}_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{q}}\right) = \min\left(h_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{q}}\right), \min_{d_{\mathbf{p}}}\left(h_{\mathbf{p}\to\mathbf{q}}\left(d_{\mathbf{p}}\right) + \lambda\right)\right). \quad (7)$$

The computational complexity for one iteration of the belief propagation algorithm in the 3D case is $O(n^2 \times m^2 \times m)$ using this linear update rule.

A bipartite graph approach can also be used to compute messages faster and reduce the memory requirements in half. Finally a hierarchical approach is also used for computing message updates since it runs much faster than the general BP while yielding comparable accuracy. The main difference between the hierarchical BP and general BP is that hierarchical BP works in a coarse-to-fine manner, first performing BP at the coarsest scale and then using the output from the coarser scale to initialize the input for the next scale.

3. Implementation on Multi-GPU

In recent years, graphics hardware has evolved toward massively parallel coprocessors that can be used to perform general purpose computation. GPUs are specially suited to problems requiring data-parallel computations with high arithmetic intensity [19]. This means that the same program can be executed in parallel on different data elements using an SIMD (single instruction multiple data) programming model, speeding up computations. Also, multiple GPUs can be used to increase the benefits of parallel computing. Many algorithms have been implemented on the GPU, and the results often yield a significant speed-up over the sequential CPU implementation of the same algorithm. On the GPU each data element has to be mapped to parallel processing threads that execute on an SIMD architecture. This requires a different approach than for sequential implementations.

Until now, general purpose processing on GPU was done through the graphics API using shader languages. This made programming very tricky since graphic primitives were used to implement nongraphic algorithms, resulting in overhead. Recently, nVidia released CUDA (Compute Unified Device Architecture) [22], which is a general purpose parallel computing architecture with a scalable programming model that is available in the latest generations of nVidia's graphics hardware. This programming model is structured in such a way that each GPU processes a kernel function that executes multiple threads in parallel. CUDA kernels have many similarities with fragment shaders, which enable stream processing. However, CUDA is conceived for general purpose parallel processing, eliminating the overhead of graphics API and also including some features of parallel processing like thread synchronization and random-access write to memory, also known as scatter. To execute kernel functions, threads are grouped into a grid of thread blocks. Each thread is identified by a block identifier and a thread identifier within the block. Identifiers can be multidimensional, up to three dimensions. Inside the kernel function, the thread identifier is used to determine which input data portion will be processed.

Memory accesses are possible in multiple memory spaces during kernel execution. Each thread has a private local memory. Each block has a shared memory for all its threads within the lifetime of the block. Finally, threads can access a global memory shared between all threads. Global memory is not cached, but by using convenient memory access patterns to coalesce various memory accesses into one and bringing data to faster shared memory, performance can be improved significantly.

Using multiple GPUs to improve timing results is also possible with CUDA. However, it is the responsibility of the programmer to design how tasks are scheduled to GPUs. In the ideal case, the problem can be divided into separate parts that can be processed in parallel, thus eliminating the need for communication between GPUs. If data has to be transferred between GPUs, it causes overhead. It is important to minimize data transfers because they are very time consuming. As in traditional parallel programming, adding more processors (GPUs) also leads to more communications overhead.

We have used CUDA to implement the algorithm described in Section 2 on a multiGPU personal computer equipped with two nVidia 9800GX2 cards, each with two GPUs. The four GPUs make for a total of 512 processing cores.

In a multi-GPU implementation there are two levels of parallel execution: multithread executions within a GPU and parallelism between GPUs. Different parts of the input data have to be scheduled over the available GPUs so as to minimize data transfers between GPUs.

We decided to divide the input into parts with some overlap between areas. The scheduling scheme is depicted in

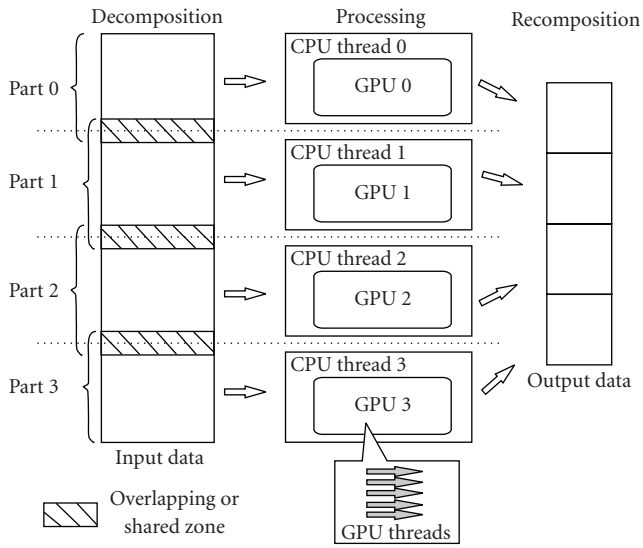


Figure 6: Data assignment scheme for multiGPU processing.

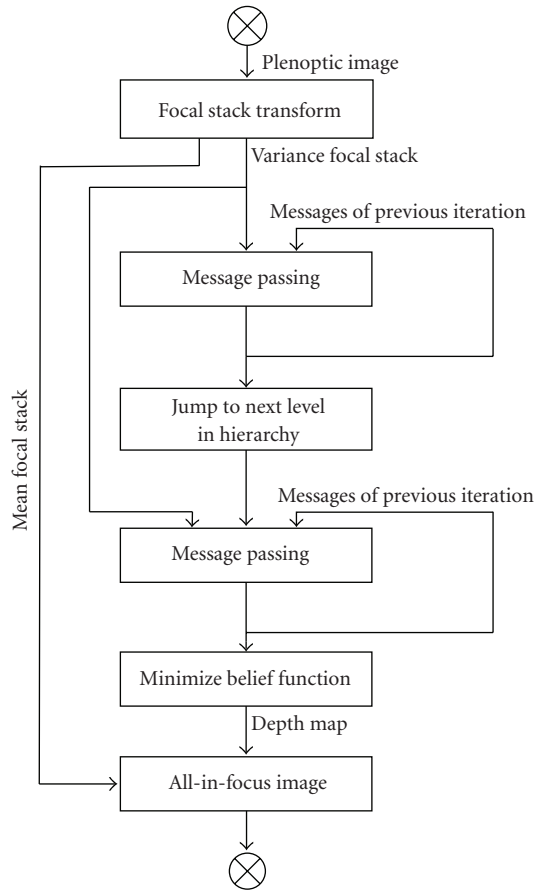


Figure 7: General flow of algorithm. Boxes represent kernel functions executing on the GPU.

Figure 6. It consists of a decomposition of the input data, a processing step that is mostly independent on each GPU, and a recomposition of the partial solution to obtain the global solution.

First, the lightfield input data is transferred from host to GPU with some overlap. This overlap allows us to compute the means and variance focal stacks without any data transfers between GPUs by performing some redundant computations. The result is a region of the focal stack with overlap on each GPU.

The same approach can now be followed by the belief propagation step, but the transfer of messages between GPUs is unavoidable. In each iteration, some messages corresponding to the edges of divisions have to be exchanged between GPUs. Once the iterative process is finished, a section of the depth map and all-in-focus image is obtained by each GPU. Finally the partial results are transferred back to the adequate location in host memory to yield the whole result.

Data processing on each GPU consists of several kernel executions as shown in Figure 7. The first step computes the super-resolved mean and variance focal stacks. Then a two-level hierarchical belief propagation is executed using the variance focal stack. Belief propagation has an iterative part consisting of the execution of the message passing kernel. After a certain number of iterations it jumps to the next level and an analogous iterative process takes place again. Then the belief function is minimized to obtain a super-resolved depth map. Finally, the all-in-focus image is generated using the depth map and focal stack.

3.1. Parallelization of SDFST. A close examination of the algorithm presented in Section 2.2 shows that the computation of each focal stack data element only requires a set of input pixels from the discrete lightfield captured with the plenoptic camera. This means that each focal stack data element can be computed in parallel if it is possible to simultaneously read the lightfield data.

To develop an efficient access scheme, it is necessary to find a mapping between the input and the output elements. The forward mapping relates the input elements from lightfield to output pixels of the focal stack, where accumulations for means and variances have to be done. Such an approach is used in the pseudocode presented in Section 2.2. The backward mapping, on the other hand, relates output memory positions to input memory positions of the lightfield, so that the necessary input pixels can be determined for a given output pixel position.

Global memory bandwidth is used more efficiently in CUDA when the thread memory accesses can be coalesced in a single memory access. nVidia GPUs with computing capability 1.1 have to access the memory addresses in sequence, so that the jth thread accesses memory position j to achieve coalescence. Examining the mappings, it can be seen that sequential access is only possible for one side of the mapping. For the forward mapping, the input can be accessed with such a sequential pattern and output accumulations are updated with an uncoalesced read/write access. If backward mapping is used, the reading of input data is uncoalesced and one coalesced output write access can

be performed. Each thread can compute the accumulations in a register and then write the result to global memory. So backward mapping is more adequate because it needs fewer memory accesses at the uncoalesced side.

Given the forward mapping $\mathbf{k} = \Delta u_g \mathbf{x} - \Delta x_g \mathbf{u}$, the backward mapping is

$$\mathbf{x} = \frac{\mathbf{k} + \Delta x_g \mathbf{u}}{\Delta u_{\mathrm{g}}}, \quad x_i \in Z. \tag{8}$$

Once the backward mapping is determined, each CUDA thread will be responsible for computing the accumulations for one pixel of the mean and variance focal stacks. Each thread is identified by $\mathbf{k} = (k_1, k_2)$. Using the inverse mapping in (8), a set of $(\mathbf{x}, \mathbf{u})$ pairs, with $\mathbf{x} = (x_1, x_2)$ and $\mathbf{u} = (u_1, u_2)$, is determined from $\mathbf{k}$. Lightfield values for each $(\mathbf{x}, \mathbf{u})$ pair are loaded and accumulated. Squared values are also accumulated. Then the variance and mean focal stack is obtained.

3.2. Parallelization of Belief Propagation. The belief propagation algorithm is well suited for parallel implementation on a GPU. Several implementations exist using shader languages [23, 24]. CUDA implementations can also be found [25]. These GPU implementations show a significant speed up over CPU implementations.

Examining the linear time message updating rule (7), it can be seen that each output message only depends on the input messages of neighboring pixels in the previous iteration and on the data term for the current pixel. There is no dependence on other partial results. This means that message updating within the same iteration can be mapped easily on the CUDA execution model.

Special care has to be taken in the implementation of iterative message passing since the process is very time consuming.

The bipartite graph approach is used to increase speed up of the message passing step [21]. The image pixel set is divided into two disjoint subsets, A and B (checkerboard pattern), so that messages of A only depend on messages sent from B in the previous iteration and vice versa. This allows computing the messages of set A at odd iterations and messages of set B at even iterations. When the solution converges it can be assumed that $M^i_{p\to q}(d_q) = M^{i-1}_{p\to q}(d_q)$ so that a final solution can be constructed from both sets. This has the advantage that at each iteration only half of the messages have to be computed, which allows either doubling the speed or executing double the number of iterations in the same time. The memory requirements are also reduced by half since messages can be computed in place.

Messages have to be suitably organized so as to favor coalescent memory accesses. It is also convenient to operate in shared memory and once a result is obtained to write it back to global memory. To explain memory organization it is useful to describe memory addressing in multidimensional matrices. Incoming messages from the previous iteration are stored in a `float4` structure. Different memory organizations can be used to obtain linear addresses from a three-dimensional coordinate (x, y, z), where x and y are spatial coordinates of the focal stack and z represents the depth level. The differences in memory organization are essentially due to the order of coordinates when they are transformed into linear addresses.

During the iterative process, messages have to be exchanged between GPUs. This implies a data transfer from one GPU to host memory and from host memory to another GPU. It is convenient to reduce the number of calls to the `cudaMemcpy` function that performs the data transfers. This can be done by properly organizing messages in GPU memory. An adequate organization is the use of (y, x, z) ordering of coordinates, because the partition between GPUs is done by rows and messages for all depth levels have to be transferred for each row. Any other ordering requires more memory copies because depth levels are not stored in adjacent positions.

Next, two possible implementation approaches of the message passing rule are discussed. The first one uses a parallel reduction scheme to compute the minimization of the messages during the update rule (4). The second approach is similar to that presented in [25] and performs minimization sequentially within the message passing kernel. However, in [25] a different expression for V_{pq} is used.

In the first approach, each thread of the block computes (5) for its depth level and stores it in shared memory. After that, all threads in the block collaborate in a parallel reduction scheme to compute $\min_{d_p}(h_{p\to q}(d_p)) + \lambda$, which is a subexpression of (7) common to all depth levels. Then, each thread has to compute (7) for its depth level using the previously obtained global minimum. This implementation is limited by the amount of available shared memory, but it has the advantage that sequential execution of thread blocks in the grid is scheduled by CUDA, not by the programmer. From the view point of memory organization the (y, x, z) ordering seems to be the most adequate since the messages for different depth levels are stored in adjacent memory positions and can be loaded with coalescent accesses. This memory organization also reduces the number of `cudaMemcpy` calls, reducing the overhead of data transfers between GPUs.

The second approach to implement the message updating rule is performing the global minimum sequentially [25]. On the GPU each thread is responsible for computing one message of the current iteration using messages from the previous one. After that, for each depth level d_p the minimum between the global minimum and $h_{p\to q}(d_p)$ is computed. It offers the advantage of having an easier extension because the programming of the kernel function is more similar to traditional sequential programming. However, since each thread computes outgoing messages for each pixel, it does not make sense to share data between threads. Although shared memory can still be used to store intermediate results during message computation, this forces a reduction in the number of threads per block since the amount of memory required increases. This, along with the sequential coding style, has an adverse impact on performance. For this approach (z, y, x) coordinate ordering is more suitable. However, the checkerboard pattern results in having only odd or even pixels used at each iteration for every row. This

does not favor coalescent memory accesses because the pixels to be read are not in adjacent memory positions.

For both approaches the results are written to the corresponding memory positions. This cannot be done with coalescent memory accesses because outgoing messages for each pixel have to be distributed to neighboring pixels accessing only one member of the `float4` structure. This problem could be solved by dividing the `float4` structure into individual memory zones. However, the checkerboard pattern used to implement the bipartite graph again breaks the coalescence.

After a certain number of iterations the algorithm jumps to the next hierarchy level and repeats the iterative message passing rule for the next level. When the iterative process is finished, the belief function has to be minimized to obtain the depth map.

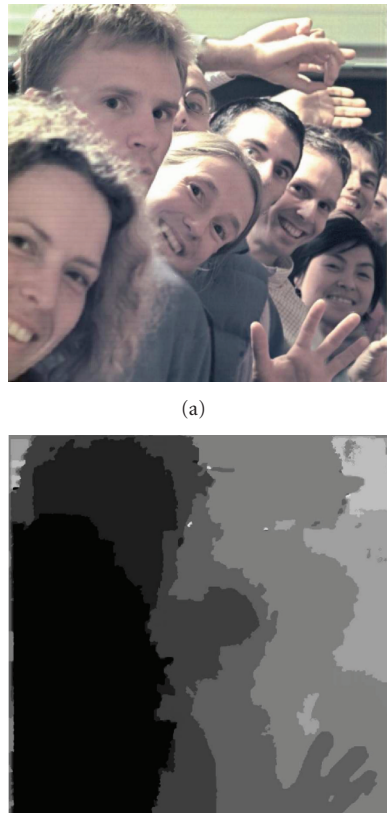
In the end it was decided to implement the first approach that consists in a message passing rule with a parallel reduction as this is expected to yield better performance.

4. Results

This section discusses some of the results obtained with the technique described in the preceding sections. First, some images and distance maps obtained from different plenoptic images are presented to show the effectiveness of the methods for high resolution images suitable for 3D HDTV. Since the objective was to generate images suitable for input to a commercial 3DTV autostereoscopic display, a more reduced lightfield of $160 \times 120 \times 15 \times 15$ was used to conduct performance tests of the GPU implementation. For this lightfield an 800×600 all-in-focus image and depth map for a Philips WOVvx 20″ display was generated [11]. Finally, the sensitivity of the parameter value for the belief propagation algorithm on the 3D reconstruction is studied.

4.1. Experimental Results. In Figure 8 we can see the all-in-focus image and depth estimation from the super-resolution focal stack and multiview stereo algorithms using the lightfield in Figure 2. Figure 9 shows an area from the plenoptic image [12] to compare the magnification results. The top image shows a nearest neighbor interpolation and the center image shows a bicubic interpolation of the 292×292 all-in focus image computed with previous approaches. Finally, the bottom image presents the results of the super-resolution technique. The results show an improvement in detail and noise reduction. It should also be noticed that from a theoretical point of view, super-resolution focal stack effectively increases resolution in lambertian scenes because each pixel in the final image is formed with data recorded directly from the scene using the plenoptic camera. In the case of interpolation techniques new pixel values are created from neighboring values without having direct information about them.

More results from other lightfield data [12] are shown in Figure 11 for the $256 \times 256 \times 15 \times 15$ lightfield of Figure 10. This shows that this method can also be used in wide range scenes.



(a)

(b)

Figure 8: Super-resolved depth and all-in-focus image.

The main drawback that can be observed in the results is the low depth resolution. This is due to the low number of pixels behind each microlens ($m = 15$). This could be solved by using a different plenoptic camera configuration with a higher value of m.

4.2. Performance Tests. To study the feasibility of real-time 3D reconstruction with a plenoptic video-camera, some performance measures were obtained for the multiGPU implementation that uses the parallel reduction technique during the message passing step of the belief propagation algorithm.

Four GPUs, each with 128 computing cores, were used to perform computations on the $160 \times 120 \times 15 \times 15$ lightfield shown in Figure 12, which is a portion of the lightfield in Figure 2 that guarantees an 800×600 pixel output. Table 1 shows speed up results obtained with up to 4 GPUs with respect to a sequential CPU implementation executed on an Intel Core 2 Quad at 2.5 GHz.

This sequential implementation executes on just one core and does not make use of special instruction set extensions, specific code optimizations, or external libraries. The program was compiled with GCC 4.1 setting the –O3 switch for code optimization. The multiGPU implementation uses a core with each GPU.

(a)

(b)

(c)

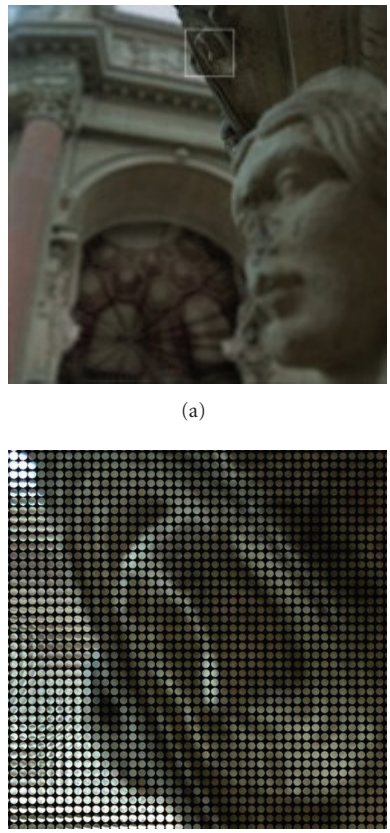
Figure 9: Nearest neighbor interpolation (a), bicubic interpolation (b), and super-resolved all-in-focus image (c).

Table 1: Frame rate and speedup over sequential implementation for up to 4 GPUs.

Execution time on CPU (ms)			4313
Number of GPUs	Time (ms)	Frame rate (fps)	Speed up
1	194.6	5.1	22.2
2	104.4	9.6	41.3
3	73.0	13.7	59.1
4	58.5	17.1	73.8

The results show that with four GPUs, near real-time performance can be achieved if we consider that the video frame rate is 24 frames per second.

Another feature to consider in parallel implementations is scalability. In the ideal case, the performance would increase linearly with the number of processors. However, in practice there is some overhead caused by data communication between processors (GPUs). It is important to study the effect caused by this overhead as the number of GPUs increases so as to decide on the optimum amount of GPUs. To show the scalability of the implementation, the frame rate with different numbers of GPUs was measured, as depicted in Figure 13. It shows that frame rate growth is quasilinear with respect to the amount of processors for up to four GPUs. However, as expected, the growth rate is smaller than in the ideal case. To increase the frame rate it would be more adequate to use GPUs with more processing cores and memory like nVidia Tesla because this requires less communications for the same processing power.



(a)

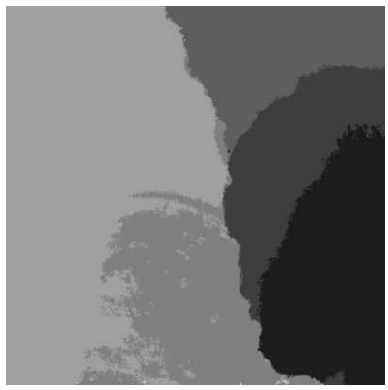
(b)

Figure 10: (a) Lightfield captured from a plenoptic camera. (b) Detail from the white square on the left image.

Performance also depends on the input size; so the frame rates necessary to generate output images with different spatial resolutions were measured and depicted in Figure 14 to show the achievable speed.

The theoretical complexity of the algorithm is $O(n^2 \times m^3 + n^2 \times m^3 \times BP\ iterations)$. Note that when m and the number of *BP iterations* are fixed, the complexity is $O(n^2)$. In this case the algorithm is linear on the output size $n^2 \times m^2$. This result also shows the feasibility of achieving satisfactory results for higher resolutions in the near future using more computing power.

(a)



(b)

Figure 11: Super-resolved all-in-focus image and depth map of lightfield in Figure 10.

Figure 12: Lightfield of $160 \times 120 \times 15 \times 15$ used for performance testing.

4.3. Sensitivity to Parameter Values of the BP Algorithm. The parameters of the belief propagation algorithm can have a significant influence on the final results. The number of iterations is imposed by real-time requirements and the value of λ has to be adjusted to obtain satisfactory results. Smaller values of λ lead to rough depth maps while higher values of λ lead to smooth solutions.

Let us now consider the sensitivity of the algorithm to changes in λ. Since the real ground truth is unknown, the proportion of equal pixels between a chosen good quality reference depth map ($\lambda = 0.05$) and the depth maps obtained with different values of λ is computed. Figure 15 shows how the parameter value influences the results obtained. Parameter changes in a small range around the selected optimal parameter do not generate significantly different results.

Figure 16 indicates the amount of depth values different from the chosen reference map for every depth map pixel. This allows us to see that border pixels are more influenced by the parameter.

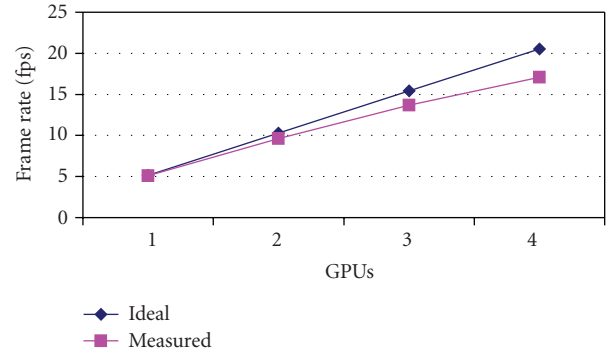


Figure 13: Ideal and measured speed up for up to 4 GPUs.

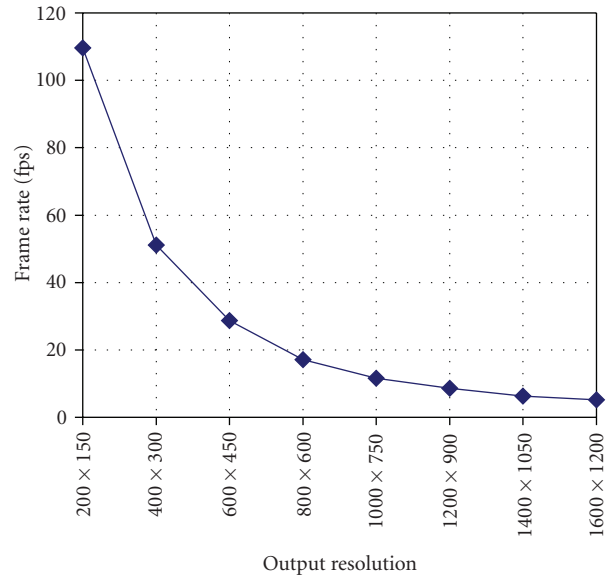


Figure 14: Frame rates obtained for different output resolutions.

5. Conclusions and Future Work

This paper presents a novel approach for obtaining a 3D reconstruction from a scene using a plenoptic video-camera. Simultaneous super-resolved depth maps and all-in-focus image estimation solve the spatial resolution drawback of previous techniques based on plenoptic cameras. Results for several lightfields are shown.

The problem of real-time execution is also addressed with the use of multiple GPUs. Near real time for a lightfield

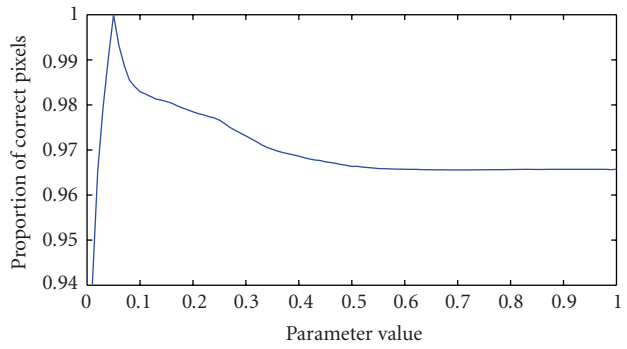


Figure 15: Proportion of correct pixels for different values of λ. Correctness was measured with respect to a reference depth map with good quality.

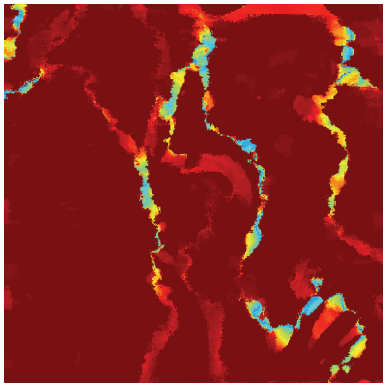


Figure 16: Number of depth values that are different from the chosen reference depth map when varying λ. Warm colors indicate an estimation independent of λ. Cold colors indicate more dependence.

of $160 \times 120 \times 15 \times 15$ is achieved. The resulting depth map and all-in-focus image are suitable as the input to a Philips WOVvx 20″ autostereoscopic display with a spatial resolution of 800×600. This shows that an adequate combination of processing algorithms with fast parallel hardware, like GPUs, makes it possible to develop a portable plenoptic video-camera with 2D color+depth output.

The use of plenoptic cameras for 3DTV has some benefits with respect to 3D capture systems based on dual stereo cameras since there is no need for geometric and color calibration or frame synchronization. Plenoptic cameras can also improve the postproduction process since special effects and synthetic content can be added easily to scenes because depths are known. This will make it easier to combine real and synthetic scenes. It should result in lower postproduction costs and time as well. There are also other features that make this format interesting for postproduction processes. Since the color image is fully focused, special effects may include defocusing on demand by blurring pixels with particular depth values. Other color effects can also be applied to particular depths in the scene (Figure 17).

Another advantage of depth range cameras is that their output can be seen as a compressed coding of 3D

Figure 17: Depth-based color manipulation of all-in-focus image.

information of a scene. Compared to other formats like dual stereoscopic video, where the amount of information with respect to conventional uncompressed 2D formats is doubled, the 2D color+depth format needs only 10%–20% more bit rate [4]. Notice that using a depth map and color image a stereo pair can be recomposed.

The use of compact designs like plenoptic video-cameras and adequate processing hardware enables the creation of passive portable 3D video capture devices. The creation of such devices will facilitate 3D content creation. Currently the lack of content is one of the main obstacles to the introduction of 3DTV in the home. Devices that generate contents for autostereoscopic displays in a simple way will help to mitigate this shortage.

Future extensions to our work will try to improve the processing speed and the super-resolution technique.

In order to increase the frame rate the use of CUDA 2.2 or higher will be considered. This new version of CUDA has some interesting features, like portable pinned memory and mapped memory. These can help to improve communication between GPUs. Also, for hardware with computing capability 1.2 the conditions for coalescent memory access are more flexible [22]. In addition, the OpenCL 1.0 framework will be an interesting candidate in the future, since it allows parallel computing on heterogeneous systems. Other improvement related with processing time will consist of porting this technique to FPGA (Field Programmable Gate Arrays) to obtain a low-cost processing device whose dimensions will allow it to be integrated as a image processing chip inside plenoptic cameras to encode the raw plenoptic image into a 2D color+depth format.

Improvements in the super-resolution algorithms will consist in the use of motion-based super-resolution techniques to achieve better resolution and to increase the robustness of depth maps and all-in-focus images. Finally, the depth resolution drawback will be addressed by using joint spatial-angular super-resolution techniques.

Acknowledgments

This work was funded by the national R&D Program (Project DPI 2006-07906) of the Ministry of Science and Technology and by the European Regional Development Fund (ERDF).

References

[1] L. Onural, "Television in 3-D: what are the prospects?" *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1143–1145, 2007.

[2] C. Fehn, "3D TV broadcasting," in *3D Videocommunication*, O. Schreer, P. Kauff, and T. Sikora, Eds., John Wiley & Sons, Chichester, UK, 2005.

[3] L. M. J. Meesters, W. A. IJsselsteijn, and P. J. H. Seuntiens, "A survey of perceptual evaluations and requirements of three-dimensional TV," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 3, pp. 381–391, 2004.

[4] A. Smolic, K. Mueller, N. Stefanoski, et al., "Coding algorithms for 3DTV—a survey," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1606–1620, 2007.

[5] G. B. Akar, A. M. Tekalp, C. Fehn, and M. R. Civanlar, "Transport methods in 3DTV—a survey," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1622–1630, 2007.

[6] M. Kawakita, K. Iizuka, T. Aida, et al., "Axi-vision camera: a three-dimensional camera," in *Three-Dimensional Image Capture and Applications III*, vol. 3958 of *Proceedings of SPIE*, pp. 61–70, San Jose, Calif, USA, January 2000.

[7] Z-cam, "3dv systems," http://www.3dvsystems.com/.

[8] F. P. Nava, J. P. Lüke, J. G. Marichal-Hernández, F. Rosa, and J. M. Rodríguez-Ramos, "A simulator for the cafadis real time 3DTV camera," in *Proceedings of 3DTV-Conference: The True Vision—Capture, Transmission and Display of 3D Video (3DTV-CON '08)*, pp. 321–324, Istanbul, Turkey, May 2008.

[9] The CAFADIS camera: international patent numbers, PCT/ES2007/000046 and ES2008/000126.

[10] P. Benzie, J. Watson, P. Surman, et al., "A survey of 3DTV displays: techniques and technologies," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1647–1657, 2007.

[11] Philips Wowx Display, http://www.businesssites.philips.com/3dsolutions/servicesupport/docs/docs.page.

[12] R. Ng, "Fourier slice photography," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '05)*, pp. 735–744, Los Angeles, Calif, USA, 2005.

[13] F. Pérez, J. G. Marichal, and J. M. Rodríguez-Ramos, "The discrete focal stack transform," in *Proceedings of the 16th European Signal Processing Conference (EUSIPCO '08)*, Lausanne, Switzerland, August 2008.

[14] J. G. Marichal-Hernández, J. P. Lüke, F. Rosa, F. Pérez, and J. M. Rodríguez-Ramos, "Fast approximate focal stack transform," in *Proceedings of 3DTV-Conference: The True Vision—Capture, Transmission and Display of 3D Video (3DTV-CON '09)*, Potsdam, Germany, May 2009.

[15] F. Pérez and J. P. Lüke, "An $O(n^2 \log n)$ per plane fast discrete focal stack transform," in *Proceedings of the Optical 3D Measurement Techniques*, 2009.

[16] T. Georgiev, K. C. Zheng, B. Curless, D. Salesin, S. Nayar, and C. Intwala, "Spatio-angular resolution tradeoff in integral photography," in *Proceedings of the Eurographics Symposium on Rendering*, 2006.

[17] A. Lumsdaine and T. Georgiev, "Full resolution lightfield rendering," Tech. Rep., Adobe Systems, Inc., January 2008.

[18] F. Pérez and J. P. Lüke, "Simultaneous estimation of superresolved depth and all-in-focus images from a plenoptic camera," in *Proceedings of 3DTV-Conference: The True Vision—Capture, Transmission and Display of 3D Video (3DTV-CON '09)*, Potsdam, Germany, May 2009.

[19] J. D. Owens, D. Luebke, N. Govindaraju, et al., "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[20] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002.

[21] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '04)*, vol. 1, pp. 261–268, Washington, DC, USA, July 2004.

[22] nVidia Corporation, "nVidia CUDA Programming Guide".

[23] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér, "Real-time global stereo matching using hierarchical belief propagation," in *Proceedings of the British Machine Vision Conference (BMVC '06)*, pp. 989–998, September 2006.

[24] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision," in *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV '06)*, p. 76, June 2006.

[25] S. Grauer-Gray, C. Kambhamettu, and K. Palaniappan, "GPU implementation of belief propagation using CUDA for cloud tracking and reconstruction," in *Proceedings of IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS '08)*, pp. 1–4, Tampa, Fla, USA, 2008.