Focused Plenoptic Camera and Rendering

Todor Georgiev[1, a)] and Andrew Lumsdaine[2, b)]

[1)] *Adobe Systems Inc.*

[2)] *Indiana University*

(Dated: 3 March 2010)

Plenoptic cameras, constructed with internal microlens arrays, capture both spatial and angular information, i.e., the full four-dimensional radiance, of a scene. The design of traditional plenoptic cameras assumes that each microlens image is completely defocused with respect to the image created by the main camera lens. As a result, only a single pixel in the final image is rendered from each microlens image, resulting in disappointingly low resolution. A recently developed alternative approach based on the focused plenoptic camera uses the microlens array as an imaging system focused on the image plane of the main camera lens. The flexible spatio-angular tradeoff that becomes available with this design enables rendering of final images with significantly higher resolution than those from traditional plenoptic cameras. In this paper we analyze the focused plenoptic camera in optical phase space and present a series of rendering algorithms for producing high-quality high-resolution images. We also present our GPU-based implementations of these algorithms, which are able to render full screen refocused images in real time.

[a)]Electronic mail: tgeorgie@adobe.com

[b)]Electronic mail: lums@cs.indiana.edu

I. INTRODUCTION

Integral photography, introduced by Ives and Lippmann over 100 years ago[1,2] has more recently re-emerged with the introduction of the plenoptic camera. Originally presented as a technique for capturing 3D data and solving computer-vision problems[3,4], the plenoptic camera was designed as a device for recording the distribution of light rays in space, i.e., the 4D *plenoptic function* or radiance. The *light field* and *lumigraph*, introduced to the computer graphics community respectively in[5] and[6], established a framework for analyzing and processing these data. In 2005, Ng improved the plenoptic camera and introduced new methods of digital processing, including refocusing[7,8].

Because it captured the full 4D radiance, Ng's handheld plenoptic camera could produce effects well beyond the capabilities of traditional cameras. Image properties such as focus and depth of field could be adjusted *after* an image had been captured. Unfortunately, traditional plenoptic cameras suffer from a significant drawback; they render images at disappointingly low resolution. For example, images rendered from Ng's camera data have a final resolution of 300×300 pixels.

A different approach, called "full resolution lightfield rendering"[9], can produce final images at much higher resolution based on a modified plenoptic camera (the "focused plenoptic camera"[10]). This modified camera is structurally different from the earlier plenoptic camera with respect to microlens placement, microlens focus, and the following from that assumptions made about the sampling of the 4D radiance. The traditional plenoptic camera focuses the main lens on the microlenses and focuses the microlenses at infinity. The focused plenoptic camera instead focuses the main camera lens well in front of the microlenses and focuses the microlenses on the image formed inside the camera—i.e., each microlens forms a relay system with the main camera lens. This configuration produces a flexible trade-off in the sampling of spatial and angular dimensions and allows positional information in the radiance to be sampled more effectively. As a result, the focused plenoptic camera can produce images of much higher resolution than can traditional plenoptic cameras.

Other radiance capturing cameras similar to the focused plenoptic camera include the following: Microlens approach of Lippmann[2]; Thin Observation Model by Bound Optics (TOMBO)[11]; Handheld plenoptic camera[7]; Fife's Multi-Aperture Image Sensor Architecture[12]; Panoptes sensor[13].

This paper is a comprehensive presentation of the focused plenoptic camera (plenoptic camera 2.0) and image rendering with it. In particular, it describes:

- Background of the plenoptic camera 2.0, including basic radiance theory and modeling, the plenoptic camera 1.0, and other radiance capturing cameras.
- A complete development of the focused plenoptic camera, including derivation of its sampling in optical phase space, basic rendering algorithms, and a detailed description of the hardware.
- A development of high-quality rendering algorithms that provide high-resolution realistic camera effects such as depth of field and refocusing.
- A presentation of a computational framework for working with focused plenoptic camera data, including implementations of rendering algorithms and their efficient realization on modern GPU hardware.

II. RADIANCE THEORY AND MODELING

Following the development in[9,10], we denote the radiance at a given plane perpendicular to the optical axis by $r(q,p)$, where q and p represent position and direction in ray space, respectively. Compactly, a coordinate in ray space is represented by $x = (q,p)^T$.

Rays are transformed by the application of optical elements. An arbitrary ray transfer matrix, A, transforms each ray according to

$$x' = Ax. \tag{1}$$

Refraction by a lens and travel of rays in free space are respectively described by the matrix transforms: L and T:

$$L = \begin{bmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}. \tag{2}$$

Optical transforms of rays induce corresponding transforms of functions (such as radiance) defined on ray space. Let A be an optical transform of (1), and consider the induced transformation of $r(x)$ to $r'(x)$. Since all optical transfer matrices satisfy $\det A = 1$, and assuming conservation of energy, we have the radiance conservation property of all optical

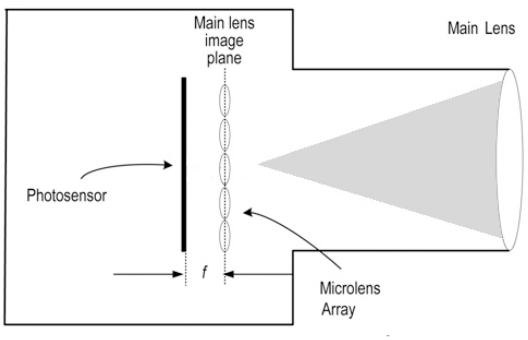

FIG. 1. The conventional plenoptic camera.

systems, i.e., we must have $r'(x') = r(x)$. Taken with $x' = Ax$, we must also have $r'(Ax) = r(x)$. Considering a ray $y = Ax$, we thus obtain $r'(y) = r(A^{-1}y)$. Since y is an arbitrary ray, we obtain the radiance transformation formula

$$r'(x) = r(A^{-1}x). \tag{3}$$

The intensity of an image at a given spatial point, denoted $I(q)$ is the integral of the radiance over all of the rays incident at that point, i.e.,

$$I(q) = \int_p r(q,p)dp. \tag{4}$$

III. PLENOPTIC CAMERAS

A. The Plenoptic Camera 1.0

The traditional plenoptic camera is based on an array of microlenses at the image plane of the main camera lens, with the sensor placed one focal length behind the microlenses (see Figure 1). The camera samples the radiance in front of the microlenses with a kernel as shown in Figure 2. Each microlens image is a vertical stack of samples in the (q,p) plane, capturing strictly the angular distribution of the radiance at the image plane.

The main camera lens is focused one focal length in front of the microlenses. Consider one microlens. We will show that each pixel under it measures the energy coming to a plane one focal length in front of that microlens as rays at a specific for that pixel angle (See Figure 3).

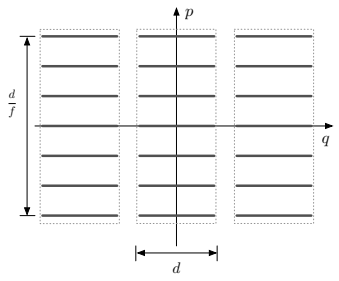

FIG. 2. Sampling of the radiance $r(q, p)$ by the microlens array represented in the two-dimensional (q, p) plane. Each pixel samples a single direction in the directional coordinate and samples a span of d (the microlens and microimage size) in the positional coordinate.

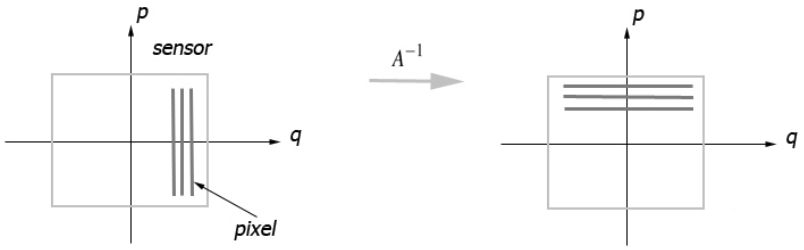

FIG. 3. Sampling pattern of one microlens in Plenoptic 1.0 camera.

To see this we compute the matrix A and A^{-1} for rays incident to a plane one focal length in front of a given microlens.

$$A = \begin{bmatrix} 1 & f \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{bmatrix} \begin{bmatrix} 1 & f \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & f \\ -\frac{1}{f} & 0 \end{bmatrix}, \; A^{-1} = \begin{bmatrix} 0 & -f \\ \frac{1}{f} & 0 \end{bmatrix} \tag{5}$$

.

Consider equation (5). A pixel on the sensor responds approximately equally to rays from all angles. Therefore its sampling kernel in ray space is represented as a vertical line as thick as the pixel in Fig. 3. A^{-1} maps this vertical line to a horizontal line because due to the bottom right zero matrix element, input p doesn't influence output p. Moreover, the spatial size of that horizontal line (the amount sampled in the spatial domain) is limited only by the microlens' diameter. This large size of the sampling kernel is the reason for the low resolution of the plenoptic camera (1.0).

Images are rendered from the radiance captured by the traditional plenoptic camera by integrating all angular samples at a particular spatial point. However, each spatial point is sampled by a single microlens, so rendering involves integrating all of the pixels in each

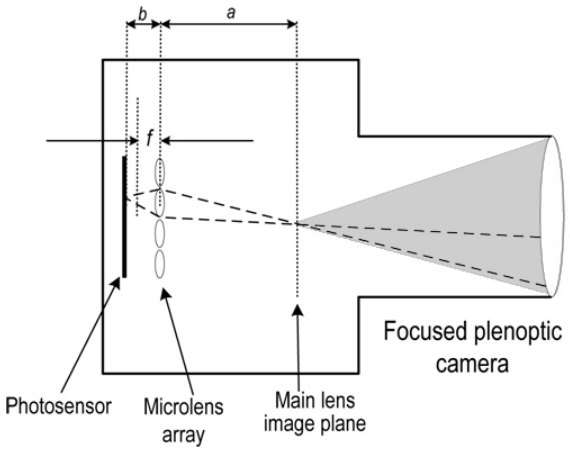

FIG. 4. The focused plenoptic camera.

microimage. As designed, rendering from the plenoptic camera produces only one pixel per microlens, resulting in a rendered image with very low resolution. Even with 100,000 microlenses, the handheld plenoptic camera reported in[7] produces a final image of only 300×300 pixels.

B. The Plenoptic Camera 2.0

As shown in Figure 4, the focused plenoptic camera is based on an array of microlenses *focused on the image plane* of the main lens. Thus, each microlens captures a portion of the image formed by the main lens. We can think of the sensor as moved back, away from the main lens, so the image is formed some distance a in front of the microlenses. The microlenses serve as an array of real cameras, re-imaging parts of that image onto the sensor.

In playing this role, each microlens forms a relay imaging system with the main camera lens. The position of each microlens satisfies the lens equation, $1/a + 1/b = 1/f$, where a, b, and f are respectively the distance from the microlens to the main lens image plane, the distance from the microlens to the sensor, and the focal length of the microlens. In our setting, b is greater than f. A different setting is possible, where the main lens image is a virtual image formed *behind* the sensor. In this case a would be negative, and b would be less than f. We do not discuss this setting of the camera in the current paper, but the treatment of such a case would be similar.

Next we will show that the focused plenoptic camera samples the radiance as in Figure 5.

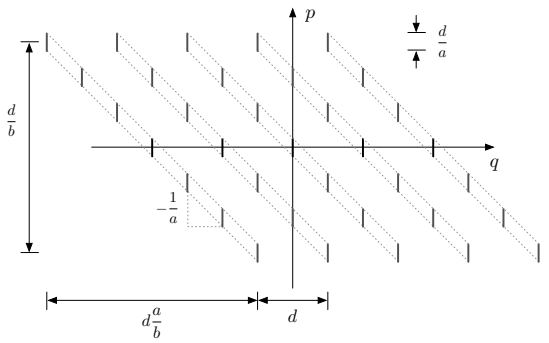

FIG. 5. Sampling of the radiance $r(q,p)$ by the microlens array of the focused plenoptic camera, represented in the two-dimensional (q,p) plane. The microlens aperture is given by d and a and b are the spacing from the microlens plane to the image plane and from the microlens plane to the sensor, respectively.

Each microlens image is a slanted stack of samples in the (q,p) plane, capturing both angular and positional distribution of the radiance at the image plane.

The total transfer matrix from that plane to the sensor is

$$A = \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -\frac{b}{a} & 0 \\ -\frac{1}{f} & -\frac{a}{b} \end{bmatrix}. \tag{6}$$

The last equality holds due to focusing. Computing the inverse,

$$A^{-1} = \begin{bmatrix} -\frac{a}{b} & 0 \\ \frac{1}{f} & -\frac{b}{a} \end{bmatrix}. \tag{7}$$

The important observation here is that due to the zero top right element, the sampling kernel for each pixel remains vertical in optical phase space after inverse mapping. As a result, sampling is done by a dense set of thin vertical kernels, and is decoupled from microlens size. See Fig. 6. Considering that minification for each microcamera is a/b, the high spatial resolution achieved is b/a times the sensor resolution, as shown in Figure 5

An important result is that the spatio-angular tradeoff for the focused plenoptic camera is not fixed by the number of microlenses. Rather, the spatio-angular tradeoffs are determined by the optical geometry (a and b). To counter edge effects in the microimages, relatively large microlenses can be used.

As with the traditional plenoptic camera, images are rendered from radiance captured with the focused plenoptic camera by integrating the angular samples at every spatial point

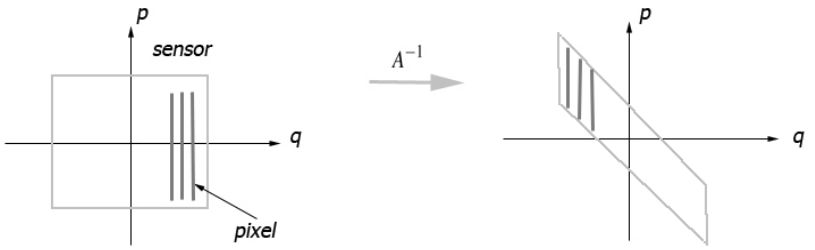

FIG. 6. Sampling pattern of one microlens in the focused plenoptic camera.

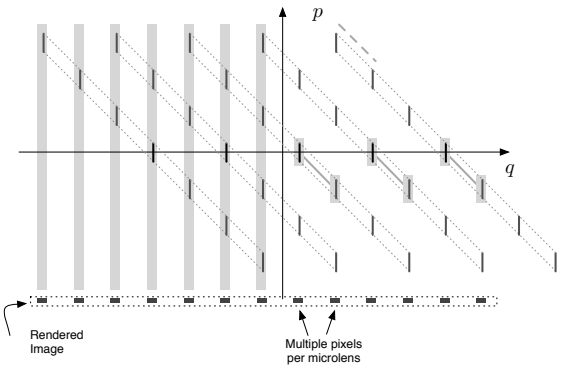

FIG. 7. Image rendering with the focused plenoptic camera. The left half of the figure shows rendering that integrates all directions associated with a given position. Note that integration takes place *across* microlens images. The right half of the figure shows rendering that only uses a single direction at each position. For the configuration shown, the rendered image has two pixels from each microimage.

(see Figure 7). Unlike the traditional plenoptic camera, however, the angular samples for a given spatial point are sampled by *different* microlenses. Therefore, rendering with plenoptic 2.0 camera data involves integrating *across* microlens images rather than within microlens images.

IV. RENDERING WITH THE FOCUSED PLENOPTIC CAMERA

A. Basic Rendering

The basic rendering process is shown in Figure 7 (right part). If we render with one angular sample for each location, we obtain M samples from each microlens image. In this example $M = 2$ and the rendered image has twice as many pixels as there are microlenses—

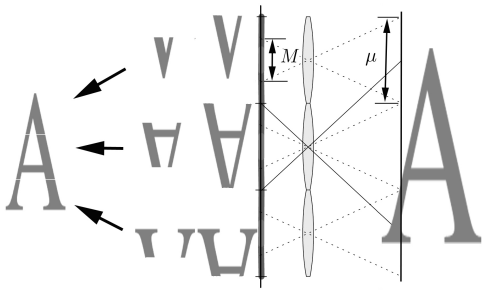

FIG. 8. Image capture geometry. Proper focusing is achieved when $\mu = M\frac{a}{b}$.

twice the resolution of the traditional plenoptic camera. In general, the attainable resolution of a full resolution rendered image depends on the depth of the scene. As derived in[10], the spatial resolution of a full-resolution image is b/a times the spatial resolution of the sensor. Resolution increases for image planes closer to the microlens plane (where b/a approaches unity), or, equivalently, for planes in the scene that are closer to the main lens (in the foreground). Thus, image planes in the foreground can be rendered with a higher resolution (larger number of pixels per microlens) than image planes in the background.

In the conventional plenoptic camera, all of the directions for a given spatial sample are contained within a single microimage and all of the spatial samples for a given direction are spread across microimages. In the focused plenoptic camera, the different views for a given spatial sample are spread across microimages.

As with the conventional plenoptic camera, an image is rendered from the focused plenoptic camera according to (4). And, as with the conventional plenoptic camera, we could implement single viewpoint rendering of (4) by evaluating $r(q, p)$ at some particular value of $p = p_0$, i.e., let $I(q) = r(q, p_0)$. In this case, however, we need to account for the fact that a single microimage samples over a range of q and a range of p. In particular, rather than selecting one spatial sample per microlens that corresponds to a single value of p to render the final image, at each microlens we extract a range of spatial samples corresponding to a range of directions (see Figure 7 where $M = 2$).

An output image corresponding to a given view (a small range of angles) can be rendered from focused plenoptic radiance data by selecting a contiguous set of pixels (a patch) from each microimage and tiling all such patches together into the final image. The important parameter in this process is the pixel size of the patch to select from each microimage.

Consider the image capture geometry in Figure 8 where we wish to reconstruct the image on the main lens image plane with pieces taken from each microlens image. The distance between microlenses (the pitch of the microlens array) is μ. We divide the main lens image plane into $\mu \times \mu$ sections such that each such section maps to an $M \times M$ portion of a microlens image. The main lens image can be reconstructed by putting together those $M \times M$ portions ("patches"). Strictly speaking, we require a negative value of M to "flip" the patches to their correct orientation before assembling them.

However, there is an alternative interpretation of the image capture geometry. Namely, for a given rendering pitch (the patch size M with which we render), there is an image plane at distance a in front of the microlenses that will satisfy $\mu = M\frac{a}{b}$. That plane is "in focus" in the sense that an image picked up from it will be rendered with no artifacts. The patches of that exact size tile together perfectly. In other words, the rendered image is "focused" only for that plane by the choice of the pitch i.e. the patch size M.

The pseudocode for the rendering algorithm, along with a graphical depiction of it, are shown in Appendix A. Intuitively, the algorithm operates as follows. We specify the pitch (defined by the number of pixels per microimage) and select squares of that size from each microlens image. The final image is rendered by tiling the selected squares together. Choosing one pitch or another puts different world planes "in focus." In other words, patches match each other perfectly only for one image plane behind the main lens. By the lens equation, this corresponds to a given depth in the real world. Thus, a different patch size would correspond to a different depth. In other words, the equivalent of refocusing is accomplished in the plenoptic 2.0 camera data through the choice of the patch size (the pitch) in the full-resolution rendering algorithm. This could be called the "full resolution" rendering principle, and it is underlying idea in all 2.0 rendering methods. The GLSL implementation is discussed in Appendix C.

Artifacts in basic full resolution rendering. Unfortunately, the basic full resolution rendering process can produce strong artifacts, as we can see in the background in Figure 10.

These artifacts result because the pitch necessary to produce artifact-free full resolution rendering is dependent on the depth in the scene. That is, different parts of a scene will require different patch sizes to be properly rendered.

The relationship between focusing and patch size also explains the artifacts that can arise when rendering a scene that has different depths. In particular, if we use a fixed patch

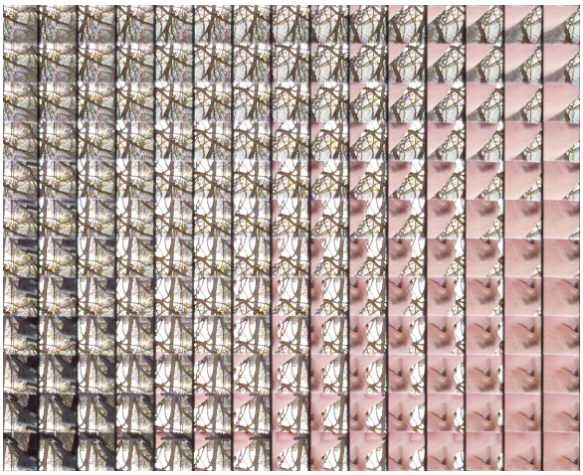

FIG. 9. A small crop from the input radiance data for our photographer rendering examples.

FIG. 10. Out of focus rendering of our photographer image: The pitch for the background is too large, resulting in artifacts.

size for a scene that has differing depths, there will be parts of the scene where the patch size is not the correct one to bring that part of the scene into focus. In those parts of the scene, the patches will not match at their boundaries. Unless that region of the scene is smooth, obvious artifacts at the microimage boundaries will be apparent. Figures 11 and 12

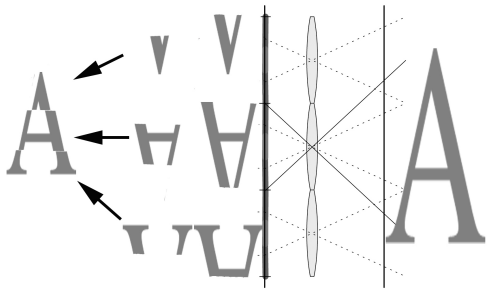

FIG. 11. Out of focus rendering regions of the scene. Here, our chosen pitch is too small for the scene being rendered, resulting in pixellation artifacts.

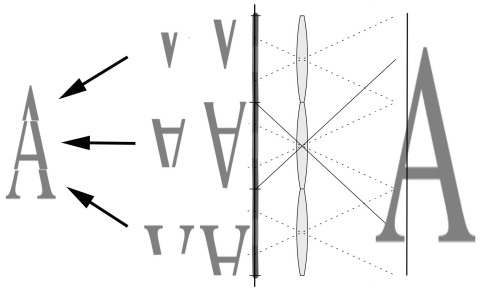

FIG. 12. Out of focus rendering regions of the scene. Here, our chosen pitch is too large for the scene being rendered, resulting in "screen door" artifacts.

illustrate the artifacts that arise when the choice of M is respectively too small or too large for rendering a scene.

These artifacts are indications that the simple rendering approach described above is not well-matched to the particular task of rendering a focused plenoptic data. Modifying the approach to produce artifact-free images depends on the particular task to be accomplished. Next we will consider depth based rendering and blensed rendering as basic approaches to reduce artifacts.

B. Depth-Based Rendering

Depth-dependence of focus for the plenoptic 2.0 camera suggests one approach for artifact-free rendering—namely, to use depth information from the scene so that different (and correct) patch sizes can be used for rendering different parts of the scene. We develop an algorithm for extracting the depth information of a scene directly from a plenoptic image

and then using this depth information to render an artifact-free image.

To determine the correct patch sizes across the rendered image, we take advantage of another important property of the focused plenoptic data, namely, that microimages capture overlapping regions of the scene. The patch size also determines the spacing between patches—the correct patch size for focusing a region of a scene will also be the spacing at which neighboring microimages will overlap. That is, the same portion of the scene that is captured by different microlenses must be rendered to the same position in the output image. This matching condition suggests the following two-pass algorithm for rendering.

1. For each microlens, determine the patch size that results in the best match with all of its neighbors.

2. Render the final image with the saved pitch value for each microlens.

Determining the minification that provides the best match between two microlens images is essentially an image registration problem. We can exploit several aspects of our system to streamline this process. First, the microimages in the captured radiance are precisely determined by the microlens geometry, and precisely aligned. Thus, the difference between neighboring microimages along the horizontal and vertical axes of the lens array will only be horizontal and vertical translations, respectively. Moreover, based on the optical design of the camera, there are bounds on how large the shift between microlens images can be. These characteristics of the captured radiance greatly simplify the depth estimation algorithm. The pseudocode for the depth estimation algorithm, along with a graphical depiction of it, are given in Appendix D.

The depth estimation algorithm produces an array of patch size values that we can subsequently use in rendering the final image. To render the final image, we modify the full resolution rendering algorithm, Appendix C, so that rather than using a fixed pitch value, we look up the precomputed value for the given microlens.

The GLSL implementation for the rendering algorithm is shown in Appendix E.

By extracting depth information as described above and then rendering the image with different magnification at each microlens, we produce the image in Figure 13. Note that regions of the image at all depths are rendered essentially artifact-free.

(a)Rendered image.

(b)Estimated depth. Lighter regions correspond to foreground (and larger pitch values).

FIG. 13. Image rendered using depth correction.

C. Rendering with Blending

Rendering for finite size apertures involves integration in the angular dimensions. Intuitively, this integration process means we must average together ("blend") the same spatial point across different microlens images (See Figure 7, left). A direct comparison of basic

rendering and rendering with blending is shown in Figure 14. Note the reduction of artifacts and the effect of "focusing" in the blended rendering version of the image.

(a)Image rendered using basic algorithm. Note the presence of artifacts due to non-matching patches at large depth.

(b)Image rendered using our blending algorithm. Note that the artifacts are now suppressed and that the out-of-focus regions appear properly blurred.

(c)A small crop from the 39 megapixel raw image that was used to render (a) and (b).

FIG. 14. Comparison of basic rendering (a) and rendering with blending (b)

The GLSL implementation for the rendering algorithm in this section is shown in Appendix F.

Although rendering with depth information enables artifact-free rendering of a single view of the scene, the result is an "all in-focus" image which precludes depth-of-field and refocusing effects. Obtaining these effects requires combining multiple views of a scene, that is, integrating over a range of views as specified in equation 4.

To accomplish the integration over p in equation (4) for the focused plenoptic camera, we must average together ("blend") the same spatial point across microlens images. (Averaging across microlens images is in contrast to the conventional plenoptic camera that averages within microlens images). For microlenses spaced μ apart and a patch size of M, the pixels that need to be averaged together to a given output pixel will be separated by distance $(\mu - M)$ in the captured raw image.

A comparison of two blended rendering results is shown in Figures 15(a) and 15(b). From the phase space diagram Figure 7 of the process we can see that for small mismatches of the slope of the integration direction, blending should be sufficient to produce a smooth blur. (Note that the slope is vertical in the figure, but since different depths are related by shear in ray space, in general slope is nonvertical, defined by M). For larger mismatches, we might not have enough views and we may instead see ghost-like artifacts due to features being repeated across multiple patches.

(a)Image rendered using blending algorithm with small pitch (7 pixels). Note that the background is in focus, the foreground is out of focus.

(b)Image rendered using blending algorithm with large pitch (10 pixels). Note that the background is out of focus, the foreground is in focus.

FIG. 15. Comparison of rendering with blending: Small pitch (a) Large pitch (b).

Focusing. An important characteristic of the plenoptic 2.0 camera was mentioned above: refocusing is accomplished through choice of the pitch size in the full-resolution rendering algorithm. That pitch is related to shear in ray space. Conversely, the value of the pitch determines the plane (the depth) in the image that is in focus. Images captured with the focused plenoptic camera can therefore refocus rendered images by choosing different values

of the pitch size. When multiple microlens images are integrated (blended), the out-of-focus regions appear blurred, as would be expected. Because the microlenses have a very small aperture, there is a significant depth of field, i.e., portions of the scene that are in focus for a given value of the pitch size will extend for a large depth.

V. CAMERA PROTOTYPE

In this section we present details of our current physical camera along with rationale for its design. The camera is medium format with an 80-mm lens and a 39-megapixel P45 digital back from Phase One. In order to produce square microimages that tile together with little loss of sensor space we have introduced a square aperture at the original aperture location. Also, we have mounted the lens on the camera with a 13-mm extension tube, which provides the needed spacing a. The microlens array is custom made by Leister Axetris. The microlenses have focal length of 1.5 mm so that they can be placed directly on the cover glass of the sensor. We are able to provide additional spacing of up to 0.5 mm to enable fine tuning of the microlens focus. The pitch of the microlenses is 500 μm with precision 1 μm. The sensor pixels are 6.8 μm. The value of $b \approx 1.6$ mm was estimated with precision 0.1 mm from known sensor parameters and independently from the microlens images at different F/numbers. Captured plenoptic images are 7308×5494 pixels.

Our microlens apertures are circular, with diameters 100 μm, and are formed by the black chromium mask deposited on the microlenses. While the small apertures extend depth of field and make microlenses diffraction limited, they also introduce high F/number and associated with it diffraction blur and longer exposure times.

The relatively large pitch of the microlenses is chosen in order to match the F-number of the main camera lens, which can be as low as $F/3$. This large pitch is needed because our microlenses are at large distance ($1.6mm$) from the sensor, defined by the cover glass.

VI. CONCLUSION

REFERENCES

[1]F. Ives, "Patent US 725,567," (1903).

[2]G Lippmann, "Epreuves reversibles. Photographies integrales.." Academie des sciences, 446–451(March 1908).

[3]T. Adelson and J. Bergen, "The plenoptic function and the elements of early vision," in *Computational models of visual processing* (MIT Press, 1991).

[4]T. Adelson and J. Wang, "Single lens stereo with a plenoptic camera," IEEE Transactions on Pattern Analysis and Machine Intelligence, 99–106(1992).

[5]Marc Levoy and Pat Hanrahan, "Light field rendering," ACM Trans. Graph., 31–42(1996).

[6]Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen, "The lumigraph," ACM Trans. Graph., 43–54(1996).

[7]R Ng, M Levoy, M Bredif, G Duval, M Horowitz, *et al.*, "Light field photography with a hand-held plenoptic camera," Computer Science Technical Report CSTR(Jan 2005).

[8]R. Ng, "Fourier slice photography," ACM Trans. Graph., 735–744(2005).

[9]Andrew Lumsdaine and Todor Georgiev, "Full resolution lightfield rendering," Tech. Rep. (Adobe Systems, 2008).

[10]Andrew Lumsdaine and Todor Georgiev, "The focused plenoptic camera," in *International Conference on Computational Photography* (2009).

[11]J Tanida, T Kumagai, K Yamada, and S Miyatake, "Thin observation module by bound optics (tombo): Concept and experimental verification," Appl. Opt(Jan 2001).

[12]K. Fife, A. El Gamal, and H.-S. P. Wong, "A 3mpixel multi-aperture image sensor with 0.7um pixels in 0.11um cmos," in *IEEE ISSCC Digest of Technical Papers* (2008) pp. 48–49.

[13]M Christensen, M Haney, D Rajan, S Douglas, and S Wood, "Panoptes: A thin agile multi-resolution imaging sensor," Government Microcuircuit Applications and Critical Technology Conference (GOMACTech-05)(Jan 2005).

[14]E Antunez, A Barth, A Adams, M Horowitz, and M Levoy, "High performance imaging using large camera arrays," International Conference on Computer Graphics and Interactive Techniques(Jan 2005).

[15]D Capel and A Zisserman, "Computer vision applied to super resolution," Signal Processing Magazine(Jan 2003).

[16]J. Chai, S. Chan, H. Shum, and X. Tong, "Plenoptic sampling," ACM Trans. Graph., 307–318(2000).

[17]F. Durand, N. Holzschuch, C. Soler, E. Chan, and F. Sillion, "A frequency analysis of light

transport," ACM Trans. Graph., 1115–1126(2005).

[18]T Georgiev, K Zheng, B Curless, D Salesin, and et al., "Spatio-angular resolution tradeoff in integral photography," Proc. Eurographics Symposium on Rendering(Jan 2006).

[19]A. Gerrard and J. M. Burch, *Introduction to Matrix Methods in Optics* (Dover Publications, 1994).

[20]B Hunt, "Super-resolution of images: algorithms, principles, performance," International Journal of Imaging Systems and Technology(Jan 1995).

[21]Aaron Isaksen, Leonard McMillan, and Steven J. Gortler, "Dynamically reparameterized light fields," ACM Trans. Graph., 297–306(2000).

[22]M Levoy, "Light fields and computational imaging," Computer(Jan 2006).

[23]T. Naemura, T. Yoshida, and H. Harashima, "3d computer graphics based on integral photography," Optics Express, Vol. 8, 2(2001).

[24]F. Okano, H. Hoshino, J. Arai, and I Yuyama, "Real-time pickup method for a three-dimensional image based on integral photography," Applied Optics, Vol. 36, 7, 1598–1603(1997).

[25]F. Okano, J. Arai, H. Hoshino, and I. Yuyama, "Three-dimensional video system based on integral photography," Optical Engineering, Vol. 38, 6(1999).

[26]P Sloan, M Cohen, and S Gortler, "Time critical lumigraph rendering," Proceedings of the 1997 symposium on Interactive 3D graphics(Jan 1997).

[27]S Todt, C Rezk-Salama, A Kolb, and K.-D Kuhnert, "Gpu-based spherical light field rendering with per-fragment depth correction," Computer Graphics Forum **27**, 2081–2095 (Dec 2008).

[28]A. Veeraraghavan, A. Mohan, A. Agrawal, R. Raskar, and J. Tumblin, "Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing," ACM Trans. Graph. **26**, 69 (2007).

[29]Kurt Bernardo Wolf, *Geometric Optics on Phase Space* (Springer, 2004).

[30]Randi J. Rost, *OpenGL(R) Shading Language (2nd Edition)* (Addison-Wesley Professional, 2006).

[31]William R. Mark, R. Steven, Glanville Kurt, Akeley Mark, and J. Kilgard, "Cg: A system for programming graphics hardware in a C-like language," ACM Transactions on Graphics **22**, 896–907 (2003).

[32]John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, "Scalable parallel program-

ming with cuda," Queue **6**, 40–53 (2008).

[33]Khronos Group, "OpenCL - The open standard for parallel programming of heterogeneous systems," Http://www.khronos.org/opencl/.

[34]"PyOpenGL," http://pyopengl.sourceforge.net/, `http://pyopengl.sourceforge.net/`.

Appendix A: Full Resolution Rendering Algorithm

Computationally, we represent the 4D radiance as a 4D array (or, equivalently, as a 2D array of 2D arrays). Assume each microlens image has dimension $n_x \times n_y$ and that the total number of microlens images is $N_x \times N_y$. We represent the radiance as `r[i,j,k,l]`, a 4D array of pixels with dimension $N_x \times N_y \times n_x \times n_y$.

Given patch size P, pseudocode for the full resolution rendering algorithm to produce a $P * N_x \times P * N_y$ output image is given below. A graphical depiction of the algorithm is shown in Figure 16.

Full Resolution Rendering Algorithm.

Given: Discrete $N_x \times N_y \times n_x \times n_y$ radiance `r[i,j,k,l]`, pitch size P.

Output: Rendered $P * N_x \times P * N_y$ image `I[s,t]`.

For (`s,t`) in (`P*Nx, P*Ny`)

`I[s,t]` = `r[i,j,k,l]`,
where `i=(s/Nx)*P`, `j=(t/Ny)*P`,
`k=(s%Nx)*P`, and `l=(t%Ny)*P`.

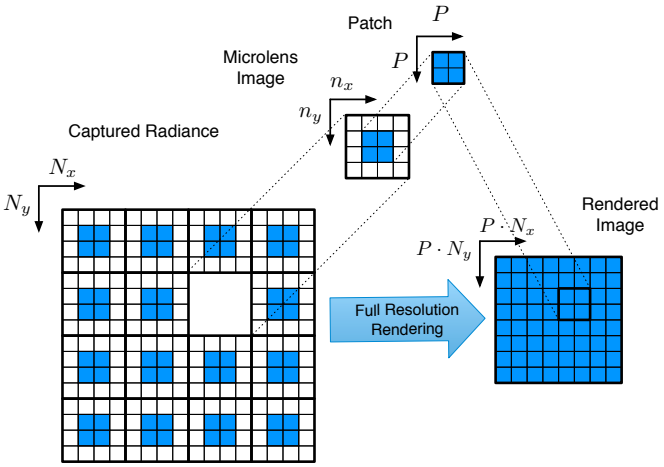

FIG. 16. The full resolution rendering algorithm creates a final rendered image from $P \times P$ patches of each $n_x \times n_y$ microimage. With $N_x \times N_y$ microimages in the captured radiance, the final rendered image is $P \cdot N_x \times P \cdot N_y$.

Appendix B: GPU Implementation

The structure of lightfield processing and rendering maps well to the massively-parallel high-performance architectures of modern GPU platforms. The results in this paper were obtained with a GPU-based approach for lightfield processing and rendering, with which we are able to achieve real-time performance.

A number of programming languages and tools have emerged for GPU programming: the OpenGL Shading Language (GLSL)[30], Cg[31], CUDA[32], and OpenCL[33]. Whereas GLSL and Cg are aimed at rendering and related tasks in mind, CUDA and OpenCL are aimed directly at general purpose programming tasks. Because lightfield rendering is similar to traditional rendering, and because a key part of interactively working with lightfields is to display them, GLSL is well-suited to our needs.

Our GLSL implementations of lightfield rendering are carried out completely in OpenGL fragment shaders. Although our examples were implemented in Python (via the PyOpenGL library[34]),interfaces to OpenGL are similar across languages. Other than providing support for creating and installing the shader, the main functionality provided by OpenGL were the following:

1. Read in the plenoptic image data (from a stored image),
2. Serialize the lightfield data to a format suitable for OpenGL,
3. Create a 2D OpenGL texture object for the plenoptic image data, and
4. Define the texture in OpenGL, using the serialized image data.

Rendering the plenoptic image data is then accomplished by rendering the installed texture, using our custom shader.

Appendix C: Basic Rendering

The shaders themselves turn out to be fairly straightforward to implement. To explain their operation, we first discuss some of the details of the optical image capture geometry as interpreted by OpenGL.

Consider the rendering geometry shown in Figure 17. Let μ be the size of one microlens image, measured in pixels. In this example, $\mu = 7$. For a given point x in the output image,

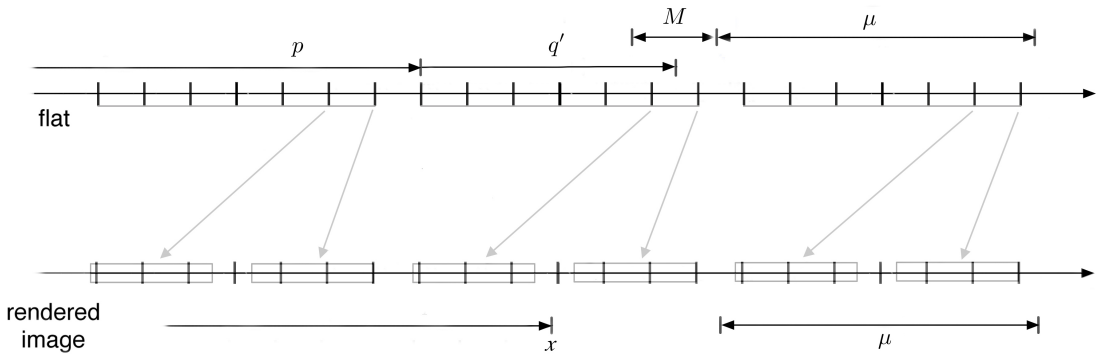

FIG. 17. Basic rendering directly from the plenoptic image, no blending. Note the magnification a/b.

we need to find the corresponding sampling point on the flat. To accomplish this process, we need to perform two computations. First, given x, we need to determine from which microlens we will render x. Second, we need to determine where in the region of size M the point x lies.

To compute which microlens corresponds to x, we take the integer part of x divided by μ, which gives us the index of the microlens. In other words, this number (let us call it p) is given by

$$p = \lfloor \frac{x}{\mu} \rfloor. \tag{C1}$$

The pixel location of the beginning of that microlens in the flat is then given by multiplying the microlens number by the size of one microlens image, i.e., $p\mu$.

Next, we need to compute the offset within the region of size M corresponding to x. To do this, we compute the difference between x and the start of microlens (p). This will give the offset in the rendered image, but we need the offset in the flat. Since we are scaling a region of size M in the flat to a region of size μ in the final image, the offset needs to be scaled by $\frac{M}{\mu}$. That is, the offset q is given by

$$q = \left(x - \lfloor \frac{x}{\mu} \rfloor \mu \right) \frac{M}{\mu} = \left(\frac{x}{\mu} - p \right) M. \tag{C2}$$

Finally, we need to make one more adjustment. We want the center of the $M \times M$ region of the flat to render to the center of the corresponding region of the final image. The formulas above will map the left edge of the microlens image to the left edge of the corresponding

region in the rendered image. To accomplish this centering, we add an offset of $\frac{\mu - M}{2}$ to q:

$$q' = q + \frac{\mu - M}{2} = \left(\frac{x}{\mu} - p\right) M + \frac{\mu - M}{2}. \tag{C3}$$

Combining (C1) and (C3), the corresponding point in the flat for a given point x in the output image is given by $f(x)$ where

$$f(x) = p\mu + q'. \tag{C4}$$

The GLSL fragment shader code to carry out this algorithm is obtained in a straightforward fashion from this formula is as follows:

```
uniform sampler2DRect flat; // Plenoptic image

uniform float M, mu;
uniform float XOffset;
uniform float YOffset;

void main()
{
  vec2 x_mu = gl_TexCoord[0].st/mu;  // x/μ
  vec2 p = floor(x_mu);                          //p = ⌊x/μ⌋
  vec2 q = (x_mu − p) ∗ M;                       //(x/μ − p)M
  vec2 qp = q + 0.5∗(mu − M);                    //q′ = q + (μ − M)/2

  vec2 offset = vec2(XOffset, YOffset)∗(mu − M);
  vec2 fx = p ∗ mu + qp + offset; // f(x) = pμ + q′

  gl_FragColor = texture2DRect(flat, fx);
}
```

The plenoptic image, as well as the values for μ and M, are provided by the user program via ***uniform*** variables. The shader program computes q, q', and $f(x)$ as q, qp, and fx, respectively. Changing the viewpoint of a synthesized image is enabled by adding user-specified offsets, *XOffset* and *XOffset* (both equal to 0 by default), to the coordinates fx.

Finally, we look up the value of the pixel in the flat and assign that value to the requested fragment color.

Appendix D: Depth Estimation

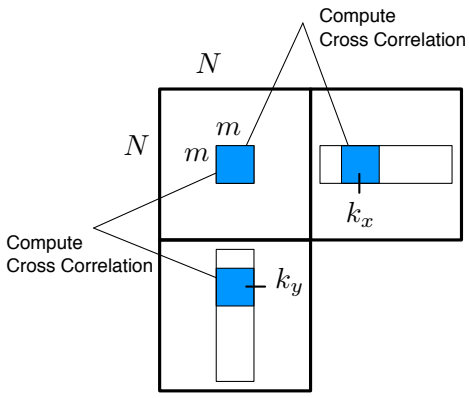

FIG. 18. Depth estimation algorithm.

An algorithm for estimating depth is given below. The algorithm produces an array of pitch values that we can subsequently use in rendering the final image. The operation of the algorithm is shown graphically in Figure 18.

Depth Estimation Algorithm.

1. For each $N \times N$ microlens image

   (a) Select a $m \times m$ window from the center of that microlens image.

   (b) For $k = -N + m/2$ to $k = N - m/2$

      i. Compute the cross correlation between the $m \times m$ window and a corresponding window centered at k_x in the neighboring microlens image along the x axis.

      ii. Record the value of k_x with the best correlation.

      iii. Compute the cross correlation between the $m \times m$ window and a corresponding window centered at k_y in the neighboring microlens image along the y axis.

iv. Record the value of k_y with the best correlation.

(c) Record value of k equal to average of k_x on left and right boundaries and k_y on top and bottom boundaries

2. Return the array of recorded values of k.

Appendix E: Depth-Based Rendering

To render the final image using depth information, we modify the rendering algorithm in Appendix C so that rather than using a fixed pitch value, we look up the value for the given microlens from the pitch array.

```
uniform sampler2DRect Flat;
uniform sampler2DRect DepthMask;

uniform float MicroimageSize;
uniform vec2 Offset;

void main()
{
  vec2 offset = Offset*MicroimageSize;
  float M2 = -25.6*texture2DRect(DepthMask,
                                 gl_TexCoord[0].st + offset).r;

  vec2 p = floor(gl_TexCoord[0].st / MicroimageSize);
  vec2 Qp = (gl_TexCoord[0].st / MicroimageSize - p) * M2 + 0.5*(MicroimageSize - M2);
  vec2 R = Qp + offset;

  vec2 vPosXY = p * MicroimageSize + R;

  vec4 colXY = texture2DRect(Flat, vPosXY);
  gl_FragColor = colXY;
}
```

Appendix F: Rendering with Blending

To realize the integration over p in equation (4), i.e., over multiple views, we must average together ("blend") the same spatial point across microlens images. For microlenses spaced μ apart and a patch size of M, the pixels that need to be averaged together to a given pixel in the rendered image will be distance $(\mu - M)$ apart. That is, we average all pixels at position $f_i(x)$ where

$$f_i(x) = p_i + q' \tag{F1}$$

where

$$p_i = \lfloor \frac{x}{\mu} \rfloor + i(\mu - M) \tag{F2}$$

for $i = \ldots, -2, -1, 0, 1, 2, \ldots$ and q' still given by (C3) above. Since μ is constant this means that for images generated with a given sampling pitch M there is a fixed upper bound, lim, for the absolute value of i, namely

$$lim = (\frac{\lfloor \frac{\mu}{M} \rfloor - 1}{2}) \tag{F3}$$

In addition, we can weight the contribution from different views to perform weighted blending. To accomplish weighted blending within GLSL, we use another texture (single-component in this case) as a lookup table to specify the weight of each pixel in the microlens as a function of position. If fragment coordinates fall outside the weight mask then the texture wrapping mode would determine what happens with the lookup. This situation occurs when the weight mask is smaller than one microlens image, or when the chosen lim value is larger than the one obtained from equation (F3). We suggest the use of a $\mu \times \mu$ Gaussian, or squared Gaussian mask with GL_CLAMP set as the wrapping mode. Also, note that when using Gaussian-like masks an alternative way of changing the viewpoint of the synthesized image is by adding an $offset$ vector to the weight coordinates. When using varying weights we need to keep track of the net weight, so that we can normalize the output color components. The GLSL implementation is shown below.

uniform sampler2DRect *weight, flat;*
uniform float *M, mu;*
uniform int *lim;*

```
void main()
{
  vec2 p = floor(gl_TexCoord[0].st / mu);
  vec2 qp = (gl_TexCoord[0].st / mu − p) ∗ M + 0.5∗(mu − M);

  vec4 colXY = vec4(0.0);
  float total_weight = 0.0;

  for (int i = −lim; i <= lim; ++i) {
    for (int j = −lim; j <= lim; ++j) {

      vec2 ij = vec2(float(i), float(j));
      vec2 dq = qp − ij ∗ M;

      float weight = texture2DRect(weight, dq).r;
      vec2 vPosXY = (p + ij)∗mu + dq;

      colXY += texture2DRect(flat, vPosXY); ∗ weight;
       total_weight += weight;
    }
  }
  gl_FragColor = colXY / total_weight;
}
```