Plenoptic Rendering with Interactive Performance using GPUs

Georgi N. Chunev[a], Andrew Lumsdaine[a], Todor G. Georgiev[b]

[a]Indiana University, Lindley Hall, Room 215, 150 S. Woodlawn Ave., Bloomington, IN 47405-7104, USA;

[b]Adobe Systems Inc., 345 Park Avenue, San Jose, CA 95110-2704, USA

ABSTRACT

Processing and rendering of plenoptic camera data requires significant computational power and memory bandwidth. At the same time, interactive rendering performance is highly desirable so that users can explore the infinite variety of images that can be rendered from a single plenoptic image. In this paper we describe a GPU-based approach for lightfield processing and rendering, with which we are able to achieve interactive performance for focused plenoptic rendering tasks such as refocusing and novel-view generation. We present a progression of rendering approaches for focused plenoptic camera data and analyze their performance on popular GPU-based systems. Our analyses are validated with experimental results on commercially available GPU hardware. Even for complicated rendering algorithms, we are able to render 39Mpixel plenoptic data to 2Mpixel images with frame rates in excess of 500 frames per second.

Keywords: Computational Photography, Plenoptic Capture, Plenoptic Rendering, GPU Algorithms

1. INTRODUCTION

Even though plenoptic photography can be dated back to the work of Gabriel Lippmann[1] and Frederick Ives,[2] the technology to make it practical has only recently become available. Microlens technology, large sensor CCDs, and highly parallel computers for the first time enable digital capture and processing of plenoptic images. While Levoy's work on lightfield rendering[3] and the Stanford Multi-Camera Array and Adelson and Wang's work on plenoptic capture[6] revived the field, Ng's handheld plenoptic camera[4] has opened the possibility of commercializing plenoptic photography. The resolution limitations of Ng's camera were addressed by Lumsdaine and Georgiev,[5] who developed the focused plenoptic camera, which generalizes the capture process and allows for a tradeoff between the spatial and angular resolution of the captured lightfield. Algorithms based on this camera are a topic of ongoing research. The goal is to achieve the highest quality output images at the highest rendering speeds.

In this work, we further Lumsdaine and Georgiev's analysis of the focused plenoptic camera in order to discuss more explicitly how it captures different perspectives of the scene. We derive a patch-based approach to synthesizing images from focused plenoptic data and demonstrate a GPU implementation that runs at interactive frame rates.

2. PLENOPTIC CAPTURE

For all discussions in this work, we choose to parameterize radiance using the intersection position, q, and the direction, p, of each ray passing through a given plane orthogonal to the viewing direction. Even though, in reality both q and p are 2D vectors, we will perform our analysis assuming that they are 1D quantities, for simplicity and without losing generality.

Further author information: (Send correspondence to G. N. Chunev)
G. N. Chunev: E-mail: gnchunev@indiana.edu
A. Lumsdaine: E-mail: lums@cs.indiana.edu
T. G. Georgiev: E-mail: tgeorgie@adobe.com

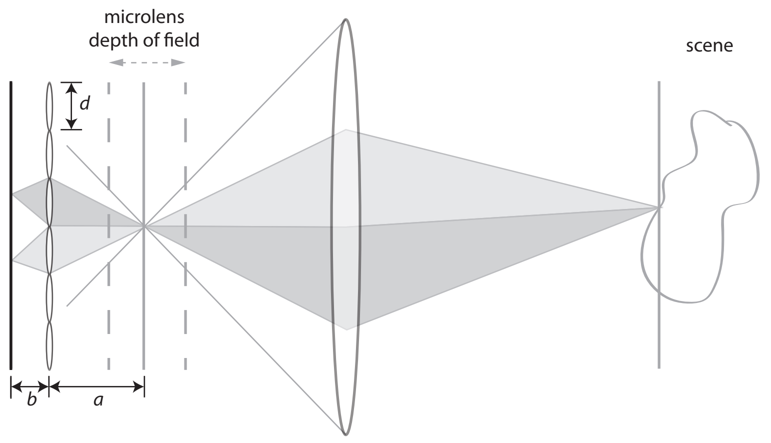


Figure 1. The focused plenoptic camera as a relay system of lenses. The microlens array is focused on images formed by the main lens and multiplexes different viewpoints of each scene location. The apertures of the microlenses are normally matched with the main lens apperture to maximize CCD utilization and avoid overlap.

2.1 Multiplexing Directional Components

The focused plenoptic camera[5] can be thought of as a lens relay system, see 1. The microlens array is kept focused on a range of image planes created by the main lens system, effectively sampling the radiance at these planes. One approach to determine the radiance capture process is to treat the microlenses as independent microcameras, directly viewing the scene in front of them, regardless of how it was transformed by the main lens. This analysis approach was taken by Lumsdaine and Georgiev,[5] who, using Gaussian optics and a Lambertian scene assumption, derived a relationship between the radiance captured by a single microimage, $r_b(q,p)$, and the radiance, $r_a(q,p)$, at a plane a distance a in front of the given microlens. See figure 2. In particular, they showed that the image captured by a microlens can be expressed as:

$$I_b(q) = \int_p r_b(q,p)dp \tag{1}$$

$$= \frac{d}{b} r_a(-\frac{a}{b}q, \frac{1}{b}q) \tag{2}$$

Rendering can be achieved by tracing back the above described capture process. When focusing on an object, and under a Labbertion assumption, to synthesize the output image I_a we can relate the pixel $I_a(q)$ to the microimage pixel $I_b(-\frac{b}{a}q)$. This is the rendering algorithm suggested by Lumsdaine and Georgiev,[5] and is the general approach that we take, but only after considering how the microimages created by off-axis microlenses relate to each other and to the perspective created by the main lens.

2.2 Multiplexing Perspectives

The microcamera based analysis approach described in the previous section does not take into account the perspective transformation introduced by the main lens and does not explicitly relate the radiance captured by neighboring microimages. An easy way to extend the analysis is to keep using Gaussian optics to describe the radiance captured behind a microlens that is coaxial to the main lens, but treat all other microlenses as pinhole cameras in order to obtain positional and directional component shifts for the radiance that they capture relative a coaxial microlens. These shifts can be inferred figure 3, which shows only rays passing through microlens centers. In order to maximize CCD utilization, the microlens apertures are usually matched to the main lens aperture. This creates microimages of size $M = \frac{d(b+m)}{m}$, where b and m give the distance from the microlens array to the

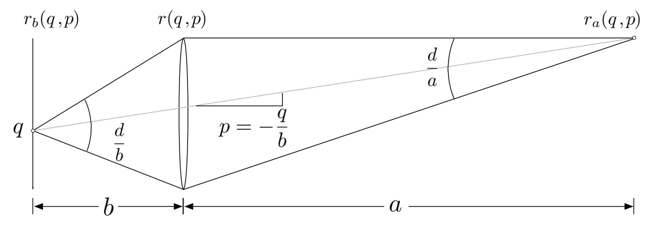


Figure 2. A diagram of an isolated microlens. The sensor is placed a distance $b > f$ behind the microlens, with f being the microlens focal length. A pixel at sensor location q integrates all rays within a $\frac{d}{a}$ range of directions, passing through point $-\frac{b}{a}q$ on the image plane. Under a Lambertian scene assumption, we can express the captured radiance in terms of the intensities of light rays passing through the center of the microlens.

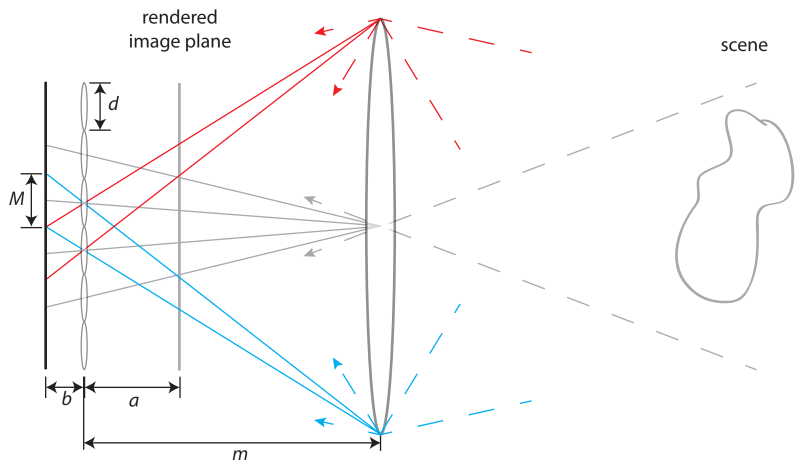


Figure 3. For any image plane that is rendered in focus, we can think of having captured the microimages with pinhole microcameras. Pixels from corresponding locations in the microimages will form distinct perspectives. The microimages are of equal sizes, $M = \frac{d(b+m)}{m}$, and contain one pixel per perspective. Because of aperture matching, the microimages are contiguous on the CCD.

sensor and to the main lens, respectively. Pixels from corresponding locations within each microlimage differ from each other in steps of $\Delta p = \frac{d}{m}$ and $\Delta q = \frac{d(m-a)}{m}$ in qp space, where a is the distance to the plane that is put in focus. They also capture distinct perspectives of the scene. Note the possibility for stereoscopy illustrated in figure 3. The distance between stereo viewpoints is limited by the main lens aperture, and the angle of the views is determined by the refractive properties of the main lens. We use all of these results to derive our rendering algorithms in the next section.

3. PLENOPTIC RENDERING

Our rendering algorithms are based on interpreting the captured microimages as radiance samples and constructing a final image by combining some subset of the available directional data for each position in that image. Thus, rendering depends on making assumptions on where the microimages fall in qp space. Figure 4 shows a diagram of qp space with plots of the pixels from two neighboring microimages. We choose the origin of the coordinate system to correspond to the axis of the main lens aperture, which we virtually move when synthesizing images with different perspectives. As can be seen in figure 3, the range of directions, p samples, that is covered by a microimage is $\frac{M}{b}$ and the covered range of positions, q samples, is $\frac{aM}{b}$. While vignetting can affect the outermost pixels, the central pixels from each microimage integrate a range of $\frac{d}{b}$ in p, as shown in figure 2. The

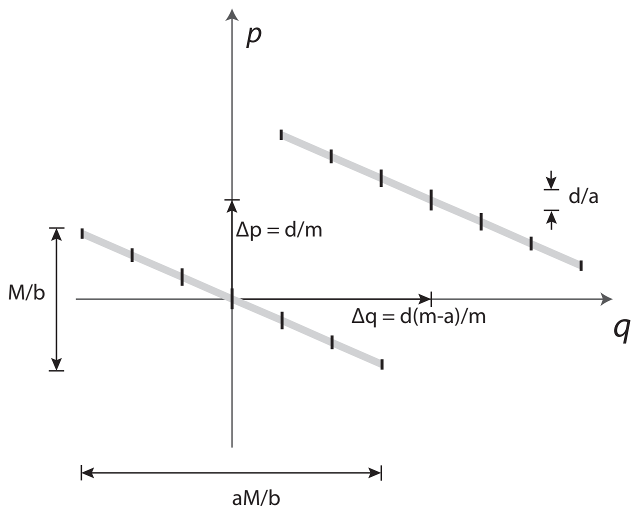


Figure 4. A diagram of qp space with plots of two neighboring microimages, one of which is concentric to the main lens. To do the plots, we interpret the microimages as capturing radiance from objects on an image plane a distance a in front of the microlens array.

size of these ranges stays the same across microimages, but there are relative shifts of $\Delta p = \frac{d}{m}$ and $\Delta q = \frac{d(m-a)}{m}$ between neighbors. Note that the q range and Δq shift depend on the plane that we choose to focus on, while the pixels that we chose to align with the perspective passing through the origin define the virtual center of each microimage and depend on the viewing axis for the image being synthesized. Thus, by interpreting the captured data differently, we can synthesize images that have different focus and show different viewpoints of the scene. The resulting perspective cannot be inferred by the qp diagram, but we know that it depends on the refractive properties of the main lens, as illustrated in 3.

3.1 Basic Patch-Based Rendring

Note that we do not render images using only one pixel from every microimage, which would be the case if we were to take only pixels that correspond to a single captured perspective. Instead, we combine information from neighboring perspectives to render neighboring pixels in the output image. This is possible, because the focused plenoptic camera allows for different perspectives to capture different, non-overlapping, portions of the scene. The advantage of doing this is an increased output resolution, as derived by Lumsdaine and Georgiev.[5] Final images necessarily have some directional discontinuities, but these are not noticeable.

The most basic rendering approach consists of taking sections from each microimage that are contiguous in q space and stitching them together at the output. For our implementations, we choose to use equally sized patches centered on some fixed location within each micorimage, which means that we restrict ourselves to synthesizing image planes that are orthogonal to the main lens axis. This is sufficient to demonstrate refocusing and the possibility for change of viewpoint, as discussed by Adelson and Wang.[6] Extensions for handling tilted image planes can be considered further.

Figure 5 shows the motivation for our basic rendering approach. We take equally sized patches from the same area of each microimage and generate an output image. While the size of the patches determines what plane we are focused on, their position determines the viewing direction. The latter can be seen in figure 6, which shows where the microimages would be located in qp space if we consider placing a different, off-axis, perspective at the origin of the coordinate system. If we want to keep rendering the same area of q space after changing the perspective, then we need to shift the input patches accordingly. Again, have in mind that the different views that we synthesize are actually co-planer, and that the resulting viewing direction is defined by the main lens.

With this basic patch-based rendering approach, out of focus regions of the image suffer from pixelation artifacts, see figure 7. These can be blurred out through blending, as described next.

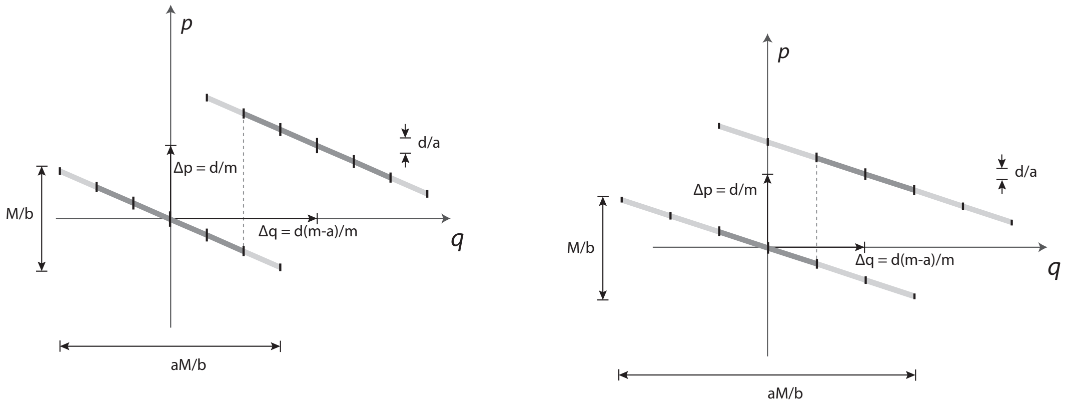


Figure 5. *qp* diagrams of two neighboring microimages, highlighting the size of the patches that need to be extracted from each microimage to synthesize an artifact free image of an object on an image plane a distance a in front of the microlens array.

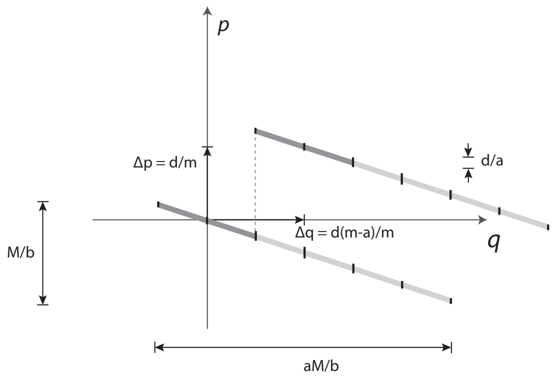


Figure 6. A *qp* diagram of two neighboring microimages, plotted as though they were captured with a shifted main lens aperture. The input patches used for basic rendering are highlighted.

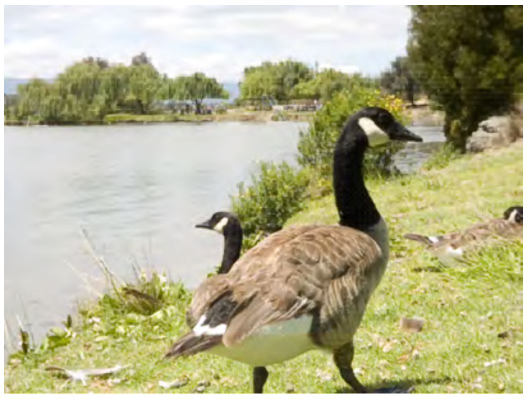
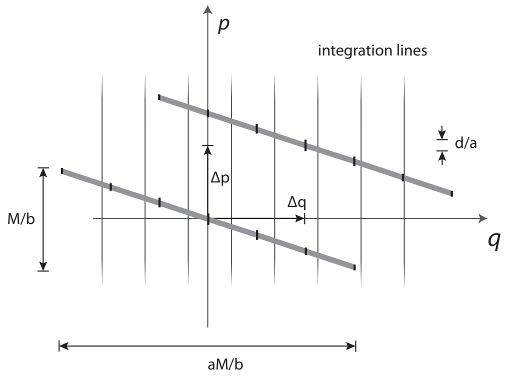


Figure 7. An example image rendered using the basic patch-based approach.



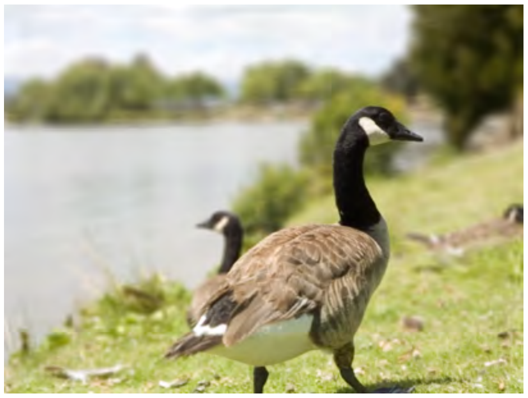
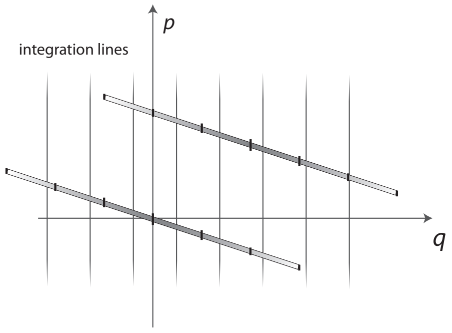


Figure 8. An example of full refocusing. The *qp* diagram shows integration lines that indicate which samples need to be blended to fully refocus the image at the plane indicated by a.



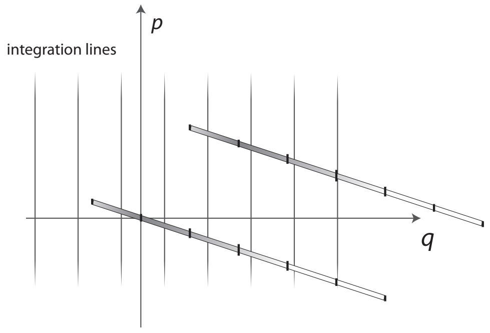


Figure 9. *qp* diagrams of two neighboring microimages. The integration lines indicate which samples need to be blended, while the gray gradient illustrates the Gaussian blending weights we use. When changing the rendered perspective, we interpret a different set of pixels as the centers of the microimages, which changes the weight assignments.

3.2 Full Refocusing

According to the image formation integral, given in equation 2, we would like to sum, or blend, all of the available directional components for a given output image position to carry out the full refocusing process. This is illustrated in figure 8, where, for each output location q, we obtain a color by blending all of the available data along the p dimension. Rendering can still be implemented using the patches from the basic algorithm, but blended with their corresponding patches in q from neighboring microimages. Refocusing can be done by changing the size of the patches, but the viewpoint is fixed, because all of the available directional information is in use. To allow for viewpoint selection, we make one last modification of the rendering algorithm that assigns weights to the blended directional samples based on the perspective that is being constructed.

3.3 Weighted Refocusing

The final algorithm that we suggest is a weighted version of the blending algorithm used for full refocusing. Since the weights are used to define an aperture for the perspective that is being rendered, a Gaussian kernel with standard deviation of roughly 0.25 of the microimage size, or less, could be appropriate. Rendering is illustrated in figure 9, and a stereo pair of synthesized images is given in figure 10. In our implementation, focusing and perspective selection are again done through manipulations of input patches. The pixels of each patch are weighted by a Gaussian blending kernel centered on the samples corresponding to the desired perspective.

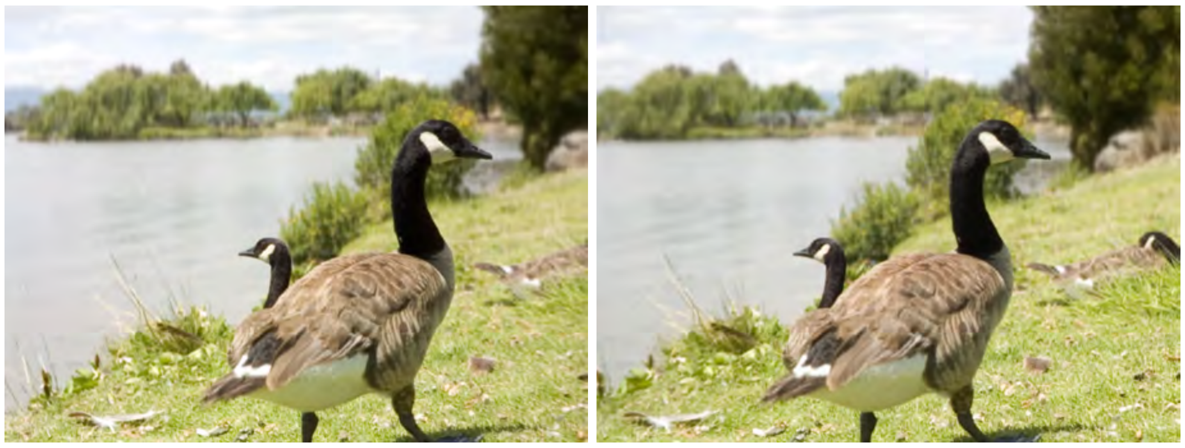


Figure 10. A stereo pair rendered using the weighted refocusing algorithm.

4. EXPERIMENTS

The images shown in previous sections come from data captured with a medium format camera with an 80-mm lens and a 39-megapixel digital back from Phase One. We used a square aperture for the main lens, so that we could match the boundaries of neighboring micorimages. We allowed objects at infinity to register in roughly 10x10 microimages. Closer objects were captured in about half as many microimages, which was still sufficient for us to be able to vary the focus and perspective of the images we synthesized.

4.1 Implementation

For our implementations we used fragment shaders, so that we can take full advantage of the GPU's parallel capabilities and specialized texture lookup hardware. In the code listing below, we provide an example OpenGL Shading Language implementation of one version of our weighted refocusing algorithm. The input flat radiance image and the Gaussian weights are provided as textures. The size of the weight mask is equal to the microimage size, mu, which is also provided to the shader as a uniform. The parameters that the user can change are the input patch size M, used for focusing; a coordinate offset vector, used for perspective selection; and a limit value, used for selecting the amount of blending in terms of the number of visited neighboring mircoimages. We make use of the OpenGL rectangular texture extension and map the full flat radiance image onto a single quad. The size of the drawn quad relative to the viewport determines the number of output pixels. We usually let the user specify the size of a patch on the output and compute a size for the quad based on the number of captured microimages. This allows for changing the focus of the final image without having to change its size. Note that, while the full output resolution for basic rendering is limited by the input patch size, weighted blending can result in some super-resolution effects, so it is fine to slightly scale up the input patches. To avoid exceeding the texture limits when blending, we suggest setting the boundary of the weight mask to zero. The same can be done say for the boundary alpha channel of the flat and the current weight can be multiplied by it.

```
#extension GL_ARB_texture_rectangle : enable

uniform sampler2DRect flat; //radiance image
uniform sampler2DRect weights; //weigth mask
uniform float mu; //microimage size
uniform vec2 offset; //coord. offset used for perspective selection
uniform float M; //patch size used for focusing
uniform int lim; //determines the amount of blending

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);
    int limX = lim;
    int limY = lim;
    float total_weight = 0.0;

    for (int i = -limX; i <= limX; ++i) {
      for (int j = -limY; j <= limY; ++j) {
        vec2 ij = vec2(float(i), float(j));

        vec2 dq = qp - ij * M;
        float weight = texture2DRect(weights, dq).r;

        vec2 vPosXY = (p + ij)*mu + dq - offset;
        vec4 coltmp = texture2DRect(flat, vPosXY);

        total_weight += weight;
        colXY += coltmp * weight;
      }
    }

    gl_FragColor = colXY / total_weight;
    gl_FragColor.a = 1.0;
}
```

4.2 Performance Analysis

We ran tests of our weighted refocusing shader on a Fermi architecture GeForce 480M GTX, which has 2GB of GDDR5 RAM running at 1200MHz and connected to a 256bit bus, resulting in a peak bandwidth of 71.5GB/s. We assumed that our shader is memory bound, but that weight lookups might be negligible, because they are shared among many threads and could be broadcast under favorable scheduling. Therefore, we simply counted the number of fetches from the flat texture (each fetch being 16 bytes large) performed by each instance of the shader and computed the expected FPS throughput for a chosen output image size. Plots of the modeled and observed performance can be seen in figure 11. We observed higher performance than expected, which is most likely because of additional data sharing. In all cases, the implementation ran interactively, with FPS counts ranging from around 50 to several hundred. Performance changed with the number of lookups for blending in accordance to the model, which indicates that the implementation is indeed memory bound. Considering that desktop GPUs can have more than double the bandwidth of the GeForce 480M GTX, we can expect at least double frame rates on standard GeForce 400 series cards.

5. CONCLUSIONS

The focused plenoptic camera multiplexes different perspectives of the scene within each microimage. Therefore, rendering different views of the scene consists of taking samples from different areas of each microimage. The range of viewing directions is limited by the main lens aperture and refractive properties. Pixels that correspond to neighboring perspectives capture non-overlapping parts of the scene, which allows for rendering images at a higher resolution by combining information from different views. This can be done by extracting patches from the microimages and stitching them together at the output. If we consider refocusing image planes that are orthogonal to the main lens axis, then the size of these patches will determine the depth of the focused plane, while their location will determine the synthesized perspective. To fully refocus a given image plane, all directional samples need to be integrated. This can be done by blending each input patch with the patches

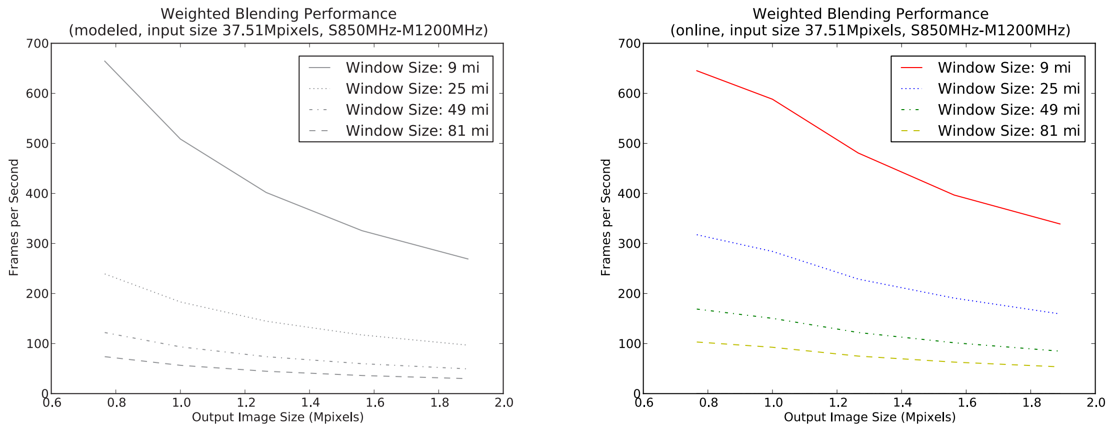


Figure 11. Plots of the performance of our weighted refocusing algorithm. Models are shown to the left, while actual experimental results are shown to the right. We ran tests with four different sizes for the neighborhood of microimages used for blending (9, 25, 49, and 81 microimages). For the actual tests we used an input patch size of 10 pixels, which corresponded to focusing on a midrange object and resulted in an output resolution of about 0.7MPixels. Due to super-resolution effects from the weighted sampling, larger images can be synthesized at the same focus. In such cases, more of the input data is shared, which could account for the higher observed performance.

from neighboring microimages that correspond to the same region of the scene. To allow for view selection while blending, a Gaussian weight can be applied to the samples of each micorimage. The mean and standard deviation of the Gaussian would define the position and the size of the aperture of the synthesized view, respectively. The resulting algorithm is very well suited for GPU implementations, as it is highly parallel and can make use of texture mapping hardware. A fragment shader implementation can run at hundreds of frames per second on modern GPUs, even for input images as large as 39MPixels.

REFERENCES

1. G. Lippmann, “Épreuves réversibles. photographies intégrales.,” *Académie des sciences* , pp. 446–451, March 1908.
2. F. Ives, “Patent US 725,567,” 1903.
3. M. Levoy and P. Hanrahan, “Light field rendering,” *ACM Trans. Graph.* , pp. 31–42, 1996.
4. R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, “Light field photography with a hand-held plenoptic camera,” Tech. Rep. CSTR 2005-02, Stanford University Computer Science, Apr. 2005.
5. A. Lumsdaine and T. Georgiev, “The focused plenoptic camera,” in *In Proc. IEEE ICCP*, pp. 1–8, 2009.
6. T. Adelson and J. Wang, “Single lens stereo with a plenoptic camera,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* , pp. 99–106, 1992.