Development of a 3-D Fluid Velocimetry Technique based on Light Field Imaging

by

Kyle Lynch

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 9, 2011

Keywords: 3-D imaging, light field rendering, velocimetry

Approved by

Brian Thurow, Chair, Associate Professor of Aerospace Engineering
Stanley Reeves, Professor of Electrical and Computer Engineering
Roy Hartfield, Professor of Aerospace Engineering

Abstract

A novel method for performing 3-D particle image velocimetry is developed and demonstrated. The technique is based on light field photography, which uses a dense lenslet array mounted near a camera sensor to simultaneously sample the spatial and angular distribution of light entering the camera. Computational algorithms are then used to refocus the image after it is taken and render a 3-D intensity distribution. This thesis provides an introduction to the concepts of light field photography and outlines the processing steps and algorithms required to obtain a 3-D velocity field. To support this, a ray-tracing simulator is used to simulate light field images and rendering codes are generated to form 3-D particle volumes which can be used for particle image velocimetry (PIV) interrogation. The simulation and rendering code is tested with uniform displacement fields and a spherical vortex, and measurement errors are quantified. It is shown that light field imaging is a feasible method for performing 3-D velocimetry with a single camera, and steps are outlined for further development and testing.

Acknowledgments

I would like to thank Dr. Brian Thurow for his invaluable assistance, advice, and time given in assisting with the development of this thesis. Our conversations have led to numerous ideas and thoughts regarding not only light field imaging, but also the broader field of optical diagnostics which I greatly enjoy and have established as the foundation for my future career.

I also wish to thank my parents for their continuous support throughout my education. I could not have completed this work without their encouragement and help, and their support of my future academic goals is invaluable.

Lastly, I'd like to thank my colleague Zach Reid for the hours of fruitful discussions and help regarding light field imaging and also numerous topics involving laser diagnostics.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The visualization of fluid motion has served both scientific and artistic purposes for centuries, dating back to the early sketches of Leonardo da Vinci. His work in the field included numerous illustrations of complex flow patterns, such as the turbulent flow downstream of a flat plate immersed in fluid. These observations represented a new approach to the study of fluids which emphasized the importance of visualization for understanding the complex interactions, patterns, and topology of flows. A resurgence of interest in flow visualizations occurred in the late nineteenth century due to the increased use of hydraulic systems in industry and the culmination of theoretical studies occurring in the field of hydrodynamics. A number of landmark experiments that formed the foundation of modern fluid dynamics were performed by pioneers such as Osborne Reynolds, Geoffery Taylor, and Ludwig Prandtl. In particular, the pipe flow experiments of Reynolds formed the foundation for the study of turbulence and established the Reynolds number as an important dimensionless parameter to characterize fluid behavior.

Since the time of these early experiments, the study of fluid mechanics has allowed remarkable technological progress to occur. The advances span a diverse range of fields, ranging from aeronautics and hydraulics to biology and medical research. These applications have drastically increased the need to not only visualize the fluid motion, but provide meaningful measurements which can be used to improve the design of these devices. Fortunately, a number of advances have also occurred in measurement technology which have allowed scientists to investigate a large gamut of flow regimes, characterized not only by velocity (subsonic, supersonic, and hypersonic), but also by a variety of other factors (chemically reacting flows, magnetohydrodynamic flows, etc.).

Many of the advancements in fluid mechanics have occurred not only due to the increasing accuracy and capability of measurement tools, but also from a more fundamental approach to research involving the understanding of various aspects of fluid mechanics and, in particular, turbulence. Various advancements have been made in the field over the years, notably the seminal works by Kolmogorov in 1941 [4], which outlined the range of spatial and temporal scales that universally pervade all turbulent flows, and Townsend in 1956 [5] which first brought to light the importance of coherent structures in turbulent flows. Both of these ideas have been expanded and studied in incredible detail; however, these fundamental ideas pose unique challenges to the development of diagnostics and in some ways provide a goal for all techniques to achieve. In this regard, the ultimate goal for diagnostic techniques is to resolve the motions contained at the smallest scales of the flow while also being able to measure the topology and interactions of the largest flow scales. While the focus of this thesis does not concern fundamental turbulence research, the requirements turbulent flows place on diagnostic methods provides the basic impetus for the current work.

In addition to conducting experiments, many fluid mechanical studies are performed using theoretical and computational tools. Fluid motion can be fully described by the set of nonlinear partial differential equations known as the Navier-Stokes equations. The equations were formalized in the middle of the nineteenth century; however, the complexity of the equations yielded only a small number of exact solutions that were in most cases not useful for practicing engineers. Rapid developments in computer technology and numerical solution methods have allowed these equations to be revisited and applied, establishing the field of computational fluid dynamics (CFD). Within this field the most rigorous solution method is known as direct numerical simulation (DNS), where the full range of spatial and temporal scales of the flow are computed. For specific cases (typically low Reynolds numbers), DNS is a viable option that provides highly accurate results. However, as the Reynolds number increases, the resulting number of required calculations rises exponentially and quickly exceeds the capability of computers to solve in a timely manner. Unfortunately,

many practical engineering scenarios operate in a high Reynolds number regime, and alternative solution methods are required. Typically, the approach is to use lower-order models to reduce the complexity of the computation, which rely on broad physical assumptions or tabulated experimental data.

For many scenarios, low-fidelity CFD is used as a powerful tool for engineers and scientists. The heavy use and reliance on computational solutions does not however eliminate the need for physical experiments. CFD depends on experimental data for validation purposes. Likewise, CFD is heavily used to optimize designs before testing is conducted, eliminating expensive additional testing and physical models. Thus, a primary challenge for modern measurement technology is to acquire accurate quantitative data for validation of CFD models, and to investigate flows that are still difficult to solve numerically, such as turbulent flows, separated flows, unsteady flows, and more.

The development of lasers and digital charge-coupled device (CCD) cameras within the last fifty years have transformed fluid measurement experiments. These technologies revolutionized the field by enabling non-intrusive measurements, in contrast to highly intrusive techniques utilizing hot-wires, pitot probes, tufts, and other simple devices. One of the first optical techniques to be developed was Laser Doppler Anemometry (LDA), which is capable of a point (1-D) measurement of all three velocity components using the interference of multiple laser beams to track the motion of particle tracers. As CCD technology developed further, planar 2-D techniques were developed, including particle image velocimetry (PIV) [6, 7, 3], planar Doppler velocimetry (PDV) [6, 8, 9], planar laser-induced fluorescence (PLIF) [10, 11, 12], and molecular tagging velocimetry (MTV) [13, 14]. In these techniques, lasers are used to create a high-intensity planar light source that is either scattered by particles present in the flow field or is absorbed by molecules within the flow which subsequently fluoresce. A CCD camera captures the resulting distribution of signal within the plane. A major advantage of these planar techniques is their ability to acquire data at multiple spatial locations simultaneously which enables measurements of derived quantities that were previously

inaccessible. For example, spatial derivatives of a 2-D velocity field yield a component of the vorticity and strain fields, which often are more useful quantities for fundamental fluids studies than the velocity field itself [15].

There are however substantial limitations to the utility of planar measurements. The majority of practical flows encountered are characterized by strongly three-dimensional features, making the selection of a planar measurement location difficult or the basic premise of a planar measurement inherently ill-posed. To extract meaningful 3-D information from the flow, multiple experiments are conducted at various planes of interest. This can considerably increase the duration and expense of a testing campaign. Another notable limitation is the inability to calculate the full set of 3-D spatial and temporal derivatives. These drawbacks have spurred the development of three-dimensional, three-component measurement techniques such as tomographic PIV, holographic PIV, and others. Each of these techniques, while powerful, have unique experimental restrictions and difficulties which prevent their more widespread use and will be considered herein.

The previous discussions illustrate the major challenge of modern laser diagnostics; namely, no diagnostic technique currently exists which is capable of accurately resolving the 3-D motion and properties of the entire range of scales in flows of practical interest. The techniques that have already been developed are capable of acquiring 3-D, 3-C velocity data but with multiple drawbacks and often complex setups. The subject of this thesis is the development of a novel three-dimensional measurement technique using concepts from light field rendering and photography to approach or exceed some of the capabilities of earlier 3-D, three-component (3-C) velocimetry techniques.

Chapter 2 provides additional background into existing three-dimensional measurement techniques and describes limitations of these approaches. Particular attention is given to tomographic and synthetic aperture PIV which have numerous parallels to the technique being developed. Chapter 3 introduces the concepts and ideas behind light field photography. This includes a review of earlier efforts in both photography and microscopy. Chapter 4

describes the development of both a simplified 1-D light field simulator, and a complete 2-D simulator for light fields. In this chapter, some of the basic shortcomings of the camera will be addressed and a detailed parametric study is performed to provide insight into the optimal experimental configuration. This includes variables such as particle seed density, lenslet sizing, focal plane spacing, etc. In Chapter 5, the light field rendering codes are discussed, including computational refocusing and volume reconstruction, which allow raw light field images to be converted into volumetric intensity data. Chapter 6 discusses the development of volumetric PIV algorithms which are used to perform the correlation on volumetric data. This first begins with a validation of the approach on 2-D images, then is extended to 3-D volumes. Lastly, Chapter 7 provides concluding remarks.

Chapter 2

Three-Dimensional Flow Measurement Background

In developing a new diagnostic technique, the capabilities of existing measurement techniques must be known and used for comparison throughout development. This chapter presents an overview of the state-of-the-art in *three-dimensional* (3-D), *three-component* (3-C) velocimetry techniques.

2.1 Scanning PIV

Scanning PIV is an intuitive extension of traditional PIV, which makes use of high-repetition-rate lasers and cameras to record a number of particle image pairs throughout a short duration scan, such that the motion of the flow is effectively frozen during the scan. Galvanometric scanning mirrors or rotating mirrors are used to perform the scanning by displacing the individual laser sheets so they are located at various depths with respect to the camera. Traditional PIV algorithms are then used on each particle image pair to yield a velocity field for each slice of the volume.

The technique takes advantage of the mature processing techniques developed for traditional PIV, and of all 3-D PIV techniques, has the highest in-plane accuracy in determining velocity. By using a stereoscopic imaging arrangement, the third component of velocity can be obtained for each planar slice, making the technique a true 3-D, 3-C measurement which has accuracy equal to traditional stereoscopic PIV.

The primary disadvantage of the technique is that the timescale of the scanning and imaging is often orders of magnitude longer than the characteristic timescales of most flows encountered in practice. Even with kHz-rate laser and camera systems, scanning PIV is typically only applied in water tunnels where flow velocities are on the order of centimeters

per second. Additionally, the cameras must be configured such that the focal depth extends across the entire depth of the scan, which can reduce the quality of the individual particle images and lower the signal-to-noise ratio. In 1995, Brucker [16] implemented a system based on this technique to measure the 3-D transient wake in a spherical cap wake flow in a water tunnel. In this work, a Reynolds number of 300 was studied, based on a cap radius of 15 mm and a freestream velocity of 10 cm/s. Each scan required 0.72 seconds for the acquisition of 9 individual planes (a total of 18 images). Another more recent study by Brucker [17] on an air flow with an average velocity of 5 m/s used a 2 kHz camera to acquire 9 individual planes for a total of 18 images. As a final example, Zhang et al. [18] used scanning PIV to study a laminar separation bubble on an airfoil at moderate angles of attack in a water tunnel. The Reynolds number in these experiments ranged from 20,000 to 60,000, based on a chord length of 200 mm and a maximum freestream velocity of 0.3 m/s. For all of the examples above, scanning PIV has provided useful 3-D data; however, the restrictions on flow velocity are severe enough to prevent the technique from being used for the vast majority of practical flows.

2.2 Defocusing PIV

In defocusing PIV, particle depth information is recovered by utilizing the natural blurring of the particle with increasing distance from the focal plane of the imaging system. An example of this effect is shown in Figure 2.1, where light from point A is in focus and results in a sharp point being formed on the image plane. In contrast, light from point B does not originate on the focal plane, and thus forms a diffuse point on the image plane. In theory, this information in itself could recover the particle depth and position. However, typical particle imaging scenarios are often also low-light scenarios, limiting the signal-to-noise ratio of the resulting image. Additionally, as the particle density increases, identifying individual particles becomes increasingly difficult due to substantial image overlap. Finally, even if these two issues could be resolved, the particle blur is symmetric about the focal plane, such

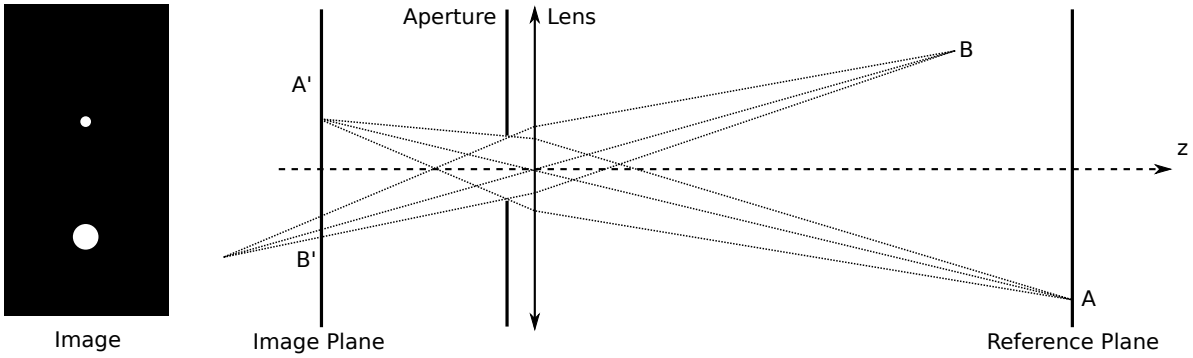


Figure 2.1: Schematic of a traditional particle imaging scenario with defocusing. Adapted from [1].

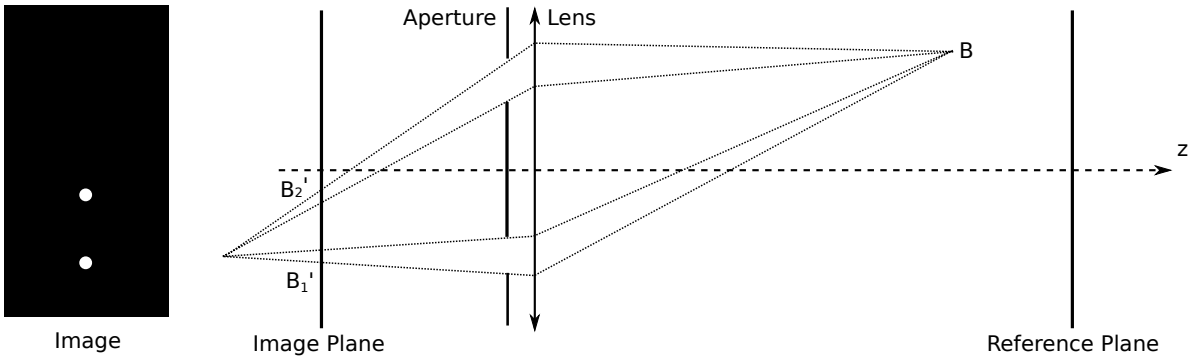


Figure 2.2: Schematic of a defocus PIV particle imaging scenario using a double aperture arrangement. Adapted from [1].

that there exists an ambiguity of whether the particle is a certain distance towards or away from the camera.

For these reasons, a specialized aperture is used to overcome each of these issues. The concept was first demonstrated by Willert and Gharib [19] in 1992 and has been continually refined since its inception. Pereira et al. [1] provides a revised description of the technique. Essentially, the system can be described through the schematic in Figure 2.2, where a dual aperture is used to encode depth information.

The defocusing PIV technique uses the image shift caused by the two apertures to perform a measurement of the depth location of the particle. Note, for a two aperture configuration, there still exists an ambiguity in depth, as a particle in opposite depth locations will form the same image on the sensor. Most defocusing PIV implementations utilize a three

aperture mask which is arranged in the form of a triangle. The ambiguity is removed by measuring the orientation of the triangle on the resulting image. Besides the triangular aperture, the concept has been extended by Lin et al. [20] using an annular aperture. The advantages of this approach are the increased optical efficiency due to the use of a larger aperture area.

Defocusing PIV has a number of attractive advantages, including a simple experimental setup and straightforward image processing. These features have led to it being used in a growing number of experiments with commercial implementations available (the TSI V3V system being an example). However, defocusing PIV has a number of drawbacks, most severely the restriction in total particle number density since the processing is more akin to individual particle tracking velocimetry rather than the image correlation techniques used in particle image velocimetry. In Pereira and Gharib [21] an error analysis is presented for determining the maximum particle density. This restriction leads to a limit in the number of recoverable velocity vectors in each image. For example, the images acquired in [20] result in approximately 450 vectors per image, in contrast to the thousands of vectors attainable from a traditional PIV recording. Further studied in [21] is the absolute error in defocusing PIV measurements, on the order of 1-2% for in-plane motion, and as much as 10% for out-of-plane motion. With these restrictions and errors, defocusing PIV is unsuitable for a large number of fundamental turbulence studies or CFD validation efforts where highly accurate data is available; however, it is finding use in general studies in a variety of fields (see Gharib et al. [22] for examples).

2.3 Tomographic PIV

Tomographic PIV is based on the illumination, recording, and reconstruction of tracer particles within a 3-D measurement volume. The technique uses several cameras to record simultaneous views of the illuminated volume which is then reconstructed using optical tomography algorithms to yield a discretized 3-D intensity field. A pair of 3-D intensity

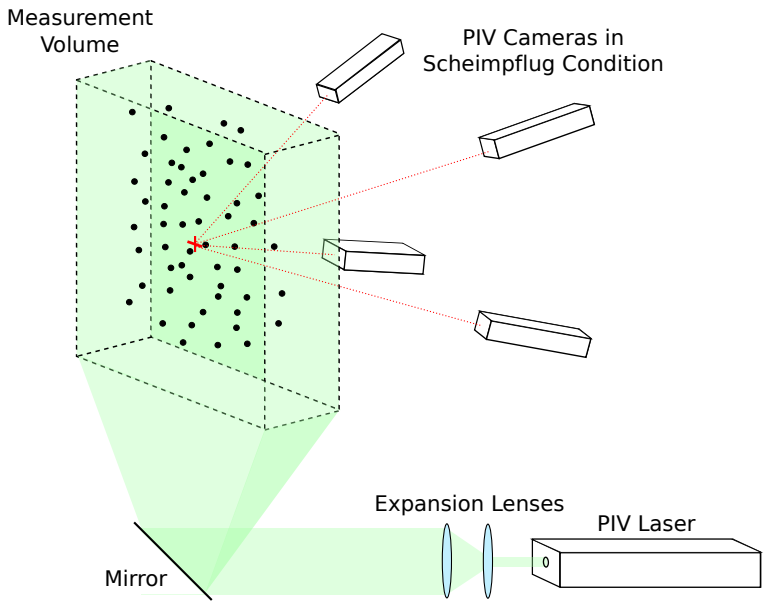


Figure 2.3: Schematic of a tomographic PIV experimental setup.

fields are analyzed using 3-D cross-correlation algorithms to calculate the 3-D, 3-C velocity field within the volume. The technique was originally developed by Elsinga et al. in 2006 [23] and has since grown and is the focus of intense research into improvements and applications of the technique.

The use of multiple cameras requires relatively complex mounting and orientation hardware to properly align each field of view. To ensure the focus is uniform across the volume, each camera lens is aligned in a Scheimpflug configuration where the lens and image planes are tilted with respect to the object plane residing in the illuminated volume. The particles are kept in the focal depth of each camera by reducing the aperture (f/#) to increase the depth of field. A calibration of the viewing angles and field of view is established using a calibration target placed at several depths throughout the measurement volume. In order for the reconstruction algorithms to correctly triangulate the particle position, the calibration must be highly accurate (typically on the order of the particle image size, under 1 mm).

The reconstruction procedure is a complex under-determined inverse problem. The primary complication is that a single set of views can result from a large number of 3-D volumes. Procedures to properly determine the *unique* volume from a set of views are the

foundation for the field of tomography. A number of procedures exist including algebraic, Fourier, and back-projection reconstruction; however, for the small number of views and high spatial frequencies present in particle images, algebraic techniques are typically used.

The first step in these algebraic algorithms is the proper conditioning of each of the images. In particular, for efficient processing, the image data must be segmented to ensure a zero-intensity background and portions of finite-intensity which define the individual particle images. This process may be quite complex due to the nature of the particle images, which themselves constitute a range of intensities. These images are used as inputs to the multiplicative algebraic reconstruction technique (MART) algorithm, which uses iterative techniques to determine the intensity of the reconstructed volumetric pixels, or voxels. The advantage of this pixel-by-pixel reconstruction technique is that it avoids the need to identify individual particles and naturally takes into account the intensity distribution of the images.

As alluded to earlier, the reconstruction is not always unambiguous, which leads to the generation of "ghost particles" which can reduce the quality of the resulting 3-D intensity field. The conceptual reasoning behind the generation of ghost particles is shown in Figure 2.4. In this case, both cameras are capturing an image consisting of two separate particle images. However, due to the ambiguity introduced along the line of sight of each camera, the reconstruction process is unable to precisely determine the actual depth of the particle, and thus accumulates intensity at the voxels corresponding to both the actual particle position and the ghost particle position. To understand why this is a problem, consider the case of a pair of particle images where the particle images have shifted between exposures. In this case, if the actual particle displacement is outward, the ghost particles will also be displaced outward, regardless of if the velocity field indicates another value. Additionally, the magnitude of the ghost particle displacement is not set by the local value of the velocity at the ghost particle location, but is a function of the velocity at the actual particle position and the viewing angles of each camera. The issue with ghost particles can be addressed by introducing additional views to the tomographic PIV setup, which substantially removes

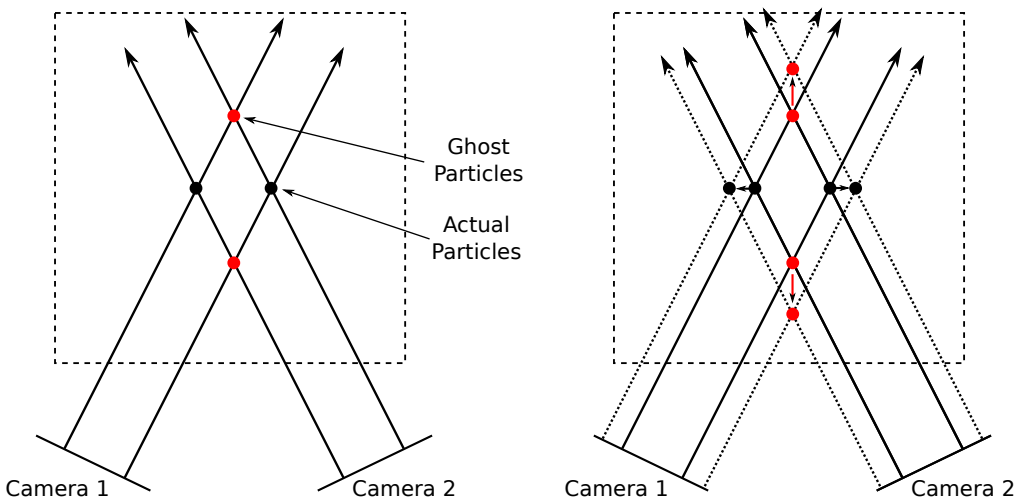


Figure 2.4: Demonstration of ghost particle formation and displacement.

the ambiguity that exists in a two or three camera setup, but increases optomechanical complexity and system cost.

In Elsinga et al. [23], a parametric study was performed using a reduced-dimensionality model (1-D slices reconstructing a 2-D volume), where parameters such as number of MART iterations, number of cameras, viewing directions, particle image density, calibration accuracy, and image noise were varied. For their assessment, the cameras were placed at optical infinity so that magnification, viewing direction, and focal depth were constant across the field of view. They concluded that accurate volumes could be reconstructed by as few as three cameras, however both the quality of the reconstruction and the achievable particle seeding density increase by adding more cameras. The viewing angles were also varied to optimize the reconstruction; an angle of 30 degrees was found optimal for reducing the number of "ghost particles," as described above. At large viewing angles, the line-of-sight from each camera increases to the point where more artifacts are present in the reconstructions. The last trade study investigated the effect of calibration inaccuracies and found that a calibration error of 1 pixel could result in as much as a 50% decrease in reconstruction accuracy. This is a significant finding that places stringent requirements on the optomechanical setup for the cameras. In practice, typical calibration errors can be reduced to around 0.2

pixels through self-calibration procedures, resulting in a negligible effect on reconstruction performance. These errors have been explored more recently in Elsinga et al. [24].

Tomographic PIV has been applied perhaps to a broader range of flows than any other 3-D velocimetry technique. In Humble et al. [25], the technique was used to elucidate the structure of a turbulent boundary layer/shock wave interaction. This study used the unique aspects of 3-D, 3-C data to draw connections from supersonic boundary layers to the more familiar structures, such as hairpin vortices and streamwise-elongated regions of low-speed flow, that are found in incompressible boundary layers. The low-frequency motion of the shock wave impinging on the boundary layer was found to have a direct connection to the motion of the elongated regions of velocity.

Another example by Scarano and Poelma [26] investigated the vorticity field in the wake of a cylinder at Reynolds numbers of 180, 360, 540, 1080, and 5540. This range of Reynolds numbers allowed the transitional behavior of the flow to be studied in great detail. In particular at $\mathrm{Re} = 180$, regular shedding occurred, which slowly began developing counter-rotative stream-wise vortex pairs at $\mathrm{Re} > 500$. Between these two Reynolds numbers, both the regular shedding and counter-rotating vortex pairs coexist. At the turbulent Reynolds number of 5540, a large increase in the range of flow scales was observed as well as the development of a distinct separated shear layer.

One of the more recent applications of the technique is for measuring the aero-acoustic field of basic shapes by using high-repetition rate lasers and cameras. Violato et al. [27] performed an experiment using a 5 kHz tomographic-PIV system to measure the 3D velocity field near the leading edge of a standard NACA 0012 airfoil which was positioned in the wake of a cylindrical rod. In this work, an analysis was performed of various pressure field integration methods to reduce errors in the calculation, which are critical to ensure the accurate sound field prediction with various acoustic analogies. It should be mentioned that this work represents a relatively new field that is receiving considerable interest for the

predicted capability of measuring the aeroacoustic properties of complex shapes and to help validate computational aeroacoustic models.

2.4 Holographic PIV

Holographic PIV (HPIV) encompasses a variety of experimental techniques which use the interference of coherent light scattered by a particle and a reference beam to encode information of the amplitude and phase of the scattered light incident on a sensor plane. This encoded information, known as a *hologram*, can then be used to reconstruct the original intensity field by illuminating the hologram with the original reference beam via optical methods or digital approximations. This intensity field is interrogated using cross-correlation techniques to yield a velocity field. By using holography instead of traditional imaging, this class of methods overcomes one of the primary limitations in most 3-D PIV schemes, namely the limited focal depth of the imaging lenses used by individual cameras. The two primary methods used in modern holographic PIV are off-axis holography and in-line holography, both of which will be discussed here.

Off-axis holography uses separate beams to provide the object and reference waves. This setup is used to avoid speckle noise from being generated from interference of the two waves within the scattering medium, which would occur if they were both propagated through the medium. A simplified schematic of this arrangement is provided in Figure 2.5. An actual off-axis experiment is a highly complex optical system comprising numerous elements, and the reader is referred to an example schematic in Sheng et al. [28] for a more complete presentation. For the recording, an object beam is directed through the particle field, which results in particle scattering normal to the beam direction. Simultaneously, a reference beam enters at an oblique angle to the setup, and the combination of scattered object light and the reference beam (i.e., the interference pattern/hologram) is recorded onto a holographic film plate. For the purposes of reconstruction, a conjugate reference beam is directed onto the developed holographic film (a transparency consisting of fine fringe patterns). The resulting

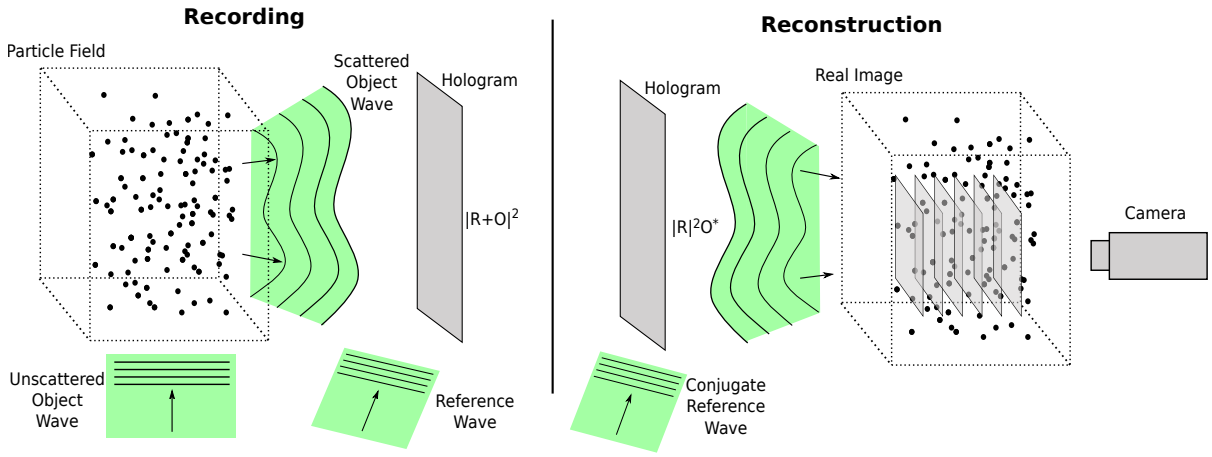


Figure 2.5: Schematic of an off-axis holographic capture and reconstruction. Adapted from [2].

wave transmitted through the film forms a stationary intensity field in real space which represents the original 3-D volume. Using a set of cameras with shallow depths of field, individual planes of this stationary volume can be captured and interrogated with typical PIV algorithms.

In-line holography is another approach that provides some unique advantages for particle imaging. Perhaps the largest of these is the use of forward scattered light, which is orders of magnitude brighter than scattering oriented normal to the beam direction. Additionally, the optical setup of such systems is much simpler because the residual light does not need to be separated and recombined at a different location. A schematic of such a setup is given in Figure 2.6. The in-line configuration also provides a relatively easy extension to apply CCD and CMOS sensors, creating a separate class of experiments known as digital in-line holography (DiH). The complexity of such setups shifts from the optical setup to image post-processing, which involves the use of simulated reference beams. Further discussion of these topics is beyond the scope of this thesis but is given an excellent treatment by Arroyo and Hinsch [29].

Critical to all holographic setups are specialized requirements on lasers and recording media. In both off-axis and in-line holographic setups, injection-seeded Nd:YAG lasers or Ruby lasers must be used due to the requirement of a long (> 1 meter) coherence length. This

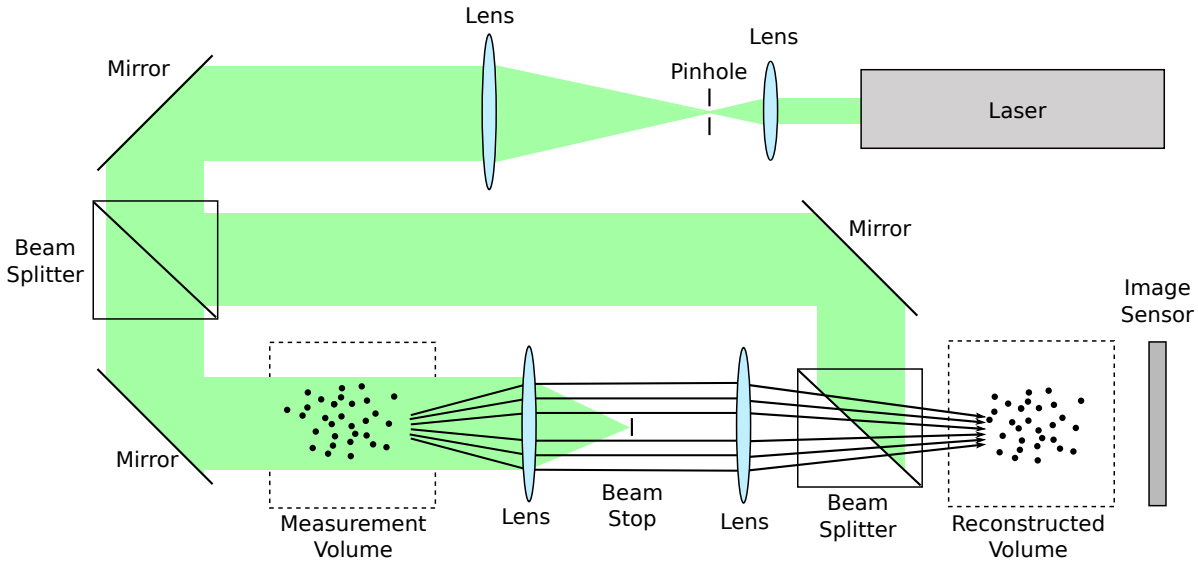


Figure 2.6: Schematic of a digital in-line holographic capture and reconstruction.

coherence requirement is necessary due to the use of interference to generate the hologram. To perform the recording, off-axis holography *requires* photographic plates to be used due to narrow holographic fringe spacing, given by Equation 2.1:

$$d_f = \frac{\lambda}{2 \sin \alpha} \tag{2.1}$$

where d_f is the fringe spacing, λ is the wavelength of laser light being used (typically 532 nm or other visible wavelengths), and α is the half-angle between the scattered object wave and the reference wave. For the case of off-axis holography, the object wave arrives at the holographic plate at a normal angle and the reference beam arrives at an angle to the plate. For example, consider the half angle α between the beams to be 22.5 degrees. The fringe spacing for this case would be approximately 0.695 microns, or equivalently 1440 lines/mm, a resolution which can only be achieved using holographic film. This precludes the use of digital cameras for direct sampling of the hologram in an off-axis setting.

A variety of issues degrade the quality of HPIV results. The first class of issues involves the reconstruction itself. In holography, the object wave of a particle is typically assumed to be spherical; however, due to Mie scattering theory, this wave is a complex shape which

can distort the reconstructed particle. Another issue is the presence of substantial speckle noise which lowers the overall signal-to-noise ratio of particle images. This effect is of greater concern for in-line holographic systems because the reference beam is propagated through the volume along with the scattered object beam. Noise can also be introduced through impurities in the scattering medium, such as temperature variations and window blemishes. Because holography requires coherent imaging, these effects are much more severe than in normal imaging conditions. The combination of these factors increases the complexity of the correlation process. In particular, the speckle noise in an HPIV recording often prevents traditional image-based correlation methods from being used. Instead, single particle identification and correlation are implemented, which set limits on particle number density. A more comprehensive outline of these error sources is given in Meng et al. [2]

In light of these issues, it may seem that Holographic PIV is too complicated and error-prone to be used for flow measurements. However, many impressive results have been obtained with all holographic approaches. Svizher and Cohen [30] used a hybrid in-line and off-axis HPIV system to study the physics of hairpin vortices in a 40 x 30 x 30 mm^3 volume with great detail. HPIV has also been useful for fundamental turbulence studies, such as in Tao et al. [31], which investigated the alignment of vorticity and strain rate tensors in high Reynolds number turbulence. In this work, a volume of 46.5 x 45.1 x 44.5 mm^3 was used. As a final example, Sheng et al. [28] used holographic microscopy to perform near-wall measurements of turbulent shear stress and velocity in boundary layers within an exceedingly small measurement volume of 1.5 mm^3. One advantage of using holography for extreme near-wall measurements is the robustness of the technique in regards to interferences (traditional PIV by comparison fares poorly in these cases, due to reflections at the interface). This listing of experiments is only a few out of numerous publications about HPIV published in recent years. For a more comprehensive listing on modern experiments, Meng et al. [2] provides a good starting point.

2.5 Synthetic Aperture PIV

Synthetic aperture PIV (SAPIV) is the most recent development in 3-D PIV techniques that makes use of similar light field rendering concepts described in this work, and thus is described in detail here. The technique uses a large camera array (eight or more cameras) to capture multiple views of the measurement volume simultaneously. Methods from light field rendering are used to generate digitally refocused planes at various depths within the measurement volume. Using reconstruction procedures, it is possible to determine a pair of 3-D intensity fields which are used as inputs for volumetric PIV through cross-correlation. The technique has just recently been developed by Belden et al. [32] and represents the newest 3-D PIV technique developed.

The design of an SAPIV experiment is in many ways similar to a tomographic PIV experiment. In each, cameras are individually positioned and adjusted to focus on a similar field of view through the use of mounting hardware and lenses operated in a Scheimpflug (tilted lens) condition. Also, the aperture of each camera must be adjusted so that the measurement volume is contained within the entire focal depth of each camera. The primary differences from a tomo-PIV setup are the larger number of cameras used and the radically different reconstruction algorithms. For SAPIV, the map-shift-average algorithm is used to construct a synthetically refocused image from the individual views by projecting each view onto a desired focal surface. In the resulting image, points that lie on the focal surface are sharp, whereas points off of the surface are blurred out. This algorithm is described for general cases by Vaish et al [33], and is briefly explained here.

The process of projecting individual views onto a desired focal surface is known as computing the *homography* of the image, which depends on the camera parameters and the focal plane (i.e., for each desired focal plane, a new homography must be computed). The homography is given as a mapping shown in Equation 2.2.

$$\begin{pmatrix} bx' \\ by' \\ b \end{pmatrix}_i = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}_i \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}_i \tag{2.2}$$

For the implementation in Belden et al [32], two assumptions were made to simplify the calculation of the homography at various focal planes. First, planar focal planes are used, and secondly, this focal plane is aligned with the center of projection of each camera. The assumption that the centers of projection of the cameras are collinear and parallel is possibly an invalid assumption for real experiments, and represents an improvement that could be made in the algorithm. Regardless, these two assumptions allow the analysis to continue, and simplifies the homography transformation for various focal planes to a coordinate shift, as shown in Equation 2.3.

$$\begin{pmatrix} x'' \\ y'' \\ 1 \end{pmatrix}_i = \begin{bmatrix} 1 & 0 & \mu_k \Delta X_{C_i} \\ 0 & 1 & \mu_k \Delta Y_{C_i} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}_i \tag{2.3}$$

In this equation, x' and y' are the coordinates of the reference focal plane (determined through a calibration procedure), and x'' and y'' are the coordinates of the new focal plane. ΔX_{C_i} and ΔY_{C_i} define the relative location of camera i, and finally μ_k is a constant that determines the amount by which the focal plane is shifted. By using this equation to transform the image from each camera and averaging the resulting transformed images, a refocused image is rendered at the specified focal plane depth. In this image, in-focus particles will appear sharp and bright, and out-of-focus particles will appear blurred and relatively dark. Through a simple thresholding procedure (considered in detail later in this thesis), the particle images can be extracted and used to build slices of a three-dimensional intensity volume. A pair of these volumes are then used in a 3-D cross-correlation to determine the 3-D, 3-C displacement.

In Belden et al. [32], a parametric study was performed to analyze the effect of the measurement volume size, particle seeding density, mapping function error, and camera arrangement. For all cases, a 5 x 5 camera array is used with an 85 mm focal-length lens, 1000 x 1000 px sensors, and pixel pitch of 10 μm. The reconstruction quality parameter Q, defined originally by Elsinga et al. [23] for tomographic PIV evaluation, is used to judge the quality of rendered volumes. In all cases, the reconstruction quality is reduced for greater particle seeding densities and camera baselines. For example, in a 100 mm^3 volume, adequate reconstructions can be obtained with particle densities up to approximately 0.15 particles/mm^3. For smaller volumes, the acceptable seeding increases dramatically; in a 50 mm^3 volume, particle densities can be as high as 0.6 particles/mm^3 to yield similar reconstruction quality. The number of cameras was also varied to determine optimal performance. In all cases, using more than 10 cameras allowed adequate reconstructions to be obtained. Diminishing returns were found for arrays consisting of more than 13 cameras. Finally, mapping errors were analyzed, and were determined to be negligible due to self-calibration procedures that can reduce the mapping error to less than 0.1 pixels.

A sample experiment was undertaken using a set of 9 computer vision cameras imaging a volume of 50 x 50 x 10 mm^3 containing a small-scale repeatable vortex ring generated by a piston apparatus. The seeding density for the experiment was approximately 0.026 particles/pixel, or equivalently 0.23 particles/mm^3. It should be noted that this density is significantly lower than simulations indicated would yield acceptable reconstruction quality. Using this setup, adequate measurements of the vortex ring were made that approximately match results obtained from 2D slices using traditional PIV.

2.6 Summary and Comparison

The previous sections presented a variety of current experimental techniques for performing 3-D, 3-C PIV. Each technique has unique strengths and weaknesses, and no single technique is capable of performing measurements for all experimental conditions. This in

some ways is a testament to the difficulty of acquiring 3-D data. In this section, comparisons will be made between the aforementioned techniques to highlight their utility and shortcomings. In this comparison, scanning PIV will not be considered due to its highly limited applicability to low-speed flows. Additionally, the processing for scanning PIV is nearly identical to planar stereoscopic PIV, for which there are numerous articles in the literature describing processing accuracy and capabilities.

The first criteria to be compared is the number of cameras used in the technique. The cost and complexity of the technique are directly tied to the number of cameras required. The fewest number of cameras needed is by holographic PIV, which acquires the entire hologram all at once using digital in-line holography or with holographic film plates. In each of these cases, only a single camera is required. In sharp contrast to this is synthetic aperture PIV, which requires upwards of 10 cameras to achieve acceptable reconstruction accuracy. However, the cameras used for SAPIV need not be high-quality cooled scientific CCD cameras, but instead can be lower-cost computer vision cameras because the averaging process used in the reconstruction procedure effectively reduces the impact of camera noise. Techniques that use a moderate number of cameras are tomo-PIV (typically using 4), and defocusing PIV (typically using 3).

The second major criteria for comparison is the size of the measurement volume each technique can handle. The largest volumes are attainable using defocusing PIV (volumes as large as 100 mm^3 have been used). In contrast, most holographic PIV systems work with very small volumes, due in large part to the limitations raised by the size of optics available and the resolution requirements for identifying individual particles. Also though, the smallest volumes are achievable using microscopic holographic PIV. In the moderate regime are tomographic and synthetic aperture PIV. Due to the sensitivity of the MART algorithm, tomo-PIV is unsuitable for depths in excess of 20 mm. Synthetic aperture PIV is capable of viewing larger depths, but with reduced seeding density.

The next comparison made is on the total number of particles which can effectively be used in each technique. This metric serves two purposes: first, it helps describe the information storage within each data set. In other words, the greater the total number of particle tracers, the more information is contained regarding the flow field. Secondly, it evaluates the robustness of the reconstruction algorithms. Currently, tomographic PIV is capable of operating with the largest number of particles, which could potentially improve as numerous researchers are exploring improvements in MART and other reconstruction algorithms. Holographic PIV is next, followed closely by defocusing PIV. Surprisingly, synthetic aperture PIV so far has been demonstrated with the least number of particles. However, SAPIV is a new technique, and with further development may equal or exceed the particle count of the other 3-D PIV methods.

Table 2.1: Comparison of Experimental Studies using 3-D PIV Techniques

Technique	Volume Size (mm^3)	No. of cameras	No. of particles	Particles/pixel
DDPIV [20]	35 x 27 x 35	1	40,000	0.038
DDPIV [21]	100 x 100 x 100	3	35,000	0.034
Tomo-PIV [23]	35 x 35 x 7	4	24,500	0.05
Tomo-PIV [25]	70 x 10 x 40	4		
Tomo-PIV [26]	100 x 100 x 20	4	100,000	0.024
HPIV [30]	40 x 30 x 30	1		
HPIV [31]	46.5 x 45.1 x 44.5	1		
HPIV [28]	1.5 x 2.5 x 1.5	1	56,250	0.014
SAPIV [32]	65 x 40 x 32	8	9860	0.015

Chapter 3

Introduction to the Concepts of Light Field Rendering and Photography

The concept of light field imaging has evolved over the past 17 years beginning with the work of Adelson and Bergen [34] and Adelson and Wang [35], and was extended by Ng et al. [36] for hand-held photography and Levoy et al. [37] for microscopy. These recent works describe the light field as the complete distribution of light in space which can be described by a 5-D function, sometimes termed the plenoptic function, where each ray is parameterized by its position (x, y, z) and angle of propagation (θ, ϕ). In a transparent medium such as air, the radiance along a light ray remains constant along all points on its path. This makes one of the spatial coordinates redundant and reduces the plenoptic function to a 4-D function which is commonly termed the light field [38]. Conventional photography only captures the 2-D spatial distribution of the 4-D light field because the angular distribution is lost through integration at the sensor surface.

Assuming the light field can be captured, there are a variety of computational methods for generating 2-D images or even 3-D models from this 4-D data. These methods fall under the general name of light field rendering, which was introduced to the computer graphics community by Levoy and Hanrahan in 1996 [38]. In general, the term refers to methods for computing new views or perspectives of a scene by extracting a proper 2-D slice from the 4-D light field. Choosing a particular slice can allow for various unique imaging situations, such as orthographic projections and crossed-slit projections. Another rendering technique that is now widely used is synthetic aperture photography, where large arrays of cameras are used to capture the light field and generate refocused images at multiple depths by extracting the proper slice from the light field.

These novel processing techniques are only available if the light field is known; therefore, any device that can record portions of or the complete 4-D light field is of tremendous value. As listed in Levoy [39], there are several ways to capture a light field including the mounting of a camera on a movable gantry and taking a large number of photos at different positions (used in the original work on light field rendering by Levoy and Hanrahan [38]), the use of a large static array of cameras (e.g. Wilburn et al. [40]), or the use of a lenslet array mounted near a camera sensor. This last device, which is termed the plenoptic camera, records the light field in a single image and is the focus of the development in this work.

3.1 Light Fields and the Plenoptic Concept

Our description of the plenoptic camera begins with the work of Adelson and Wang [35] who were the first to propose the concept of the plenoptic camera. They used the analogy of a single-lens stereo view to describe the function of the camera. Figures 3.1 through 3.3 illustrate the fundamental principles of this concept. In Figure 3.1a, a point object is focused onto an image sensor leading to a bright distinct point on the sensor surface. As the object is moved closer (Figure 3.1b) and farther away (Figure 3.1c), the image of the spot grows larger and appears blurred as it is out of focus. At this point, the position of the object is not apparent from the image. However, if an eccentric aperture is used, the distance of the object from the lens can be determined. In Figure 3.2a, when the object is in focus, the light strikes the same point as before and produces a sharp peak at the center of the sensor. When the object is moved closer to the lens (Fig. 3.2b), the eccentric aperture limits the angular range of rays traveling to the sensor and the intersection of incident rays and the image sensor leads to a blurred image of the object that is displaced to the right of center. Conversely, when the object is moved further away, the rays follow a different path through the aperture with the image formed in front of the sensor and the resulting rays traveling to the left of center. Thus, a close-up object leads to lateral displacement on the sensor to the right whereas a distant object leads to displacement to the left. As such, the precise location

and depth of the object can be determined by measuring its horizontal displacement and size on the sensor. This concept is similar to the working principles of defocusing PIV; however, by making an additional modification, the plenoptic camera is able to provide much more information regarding the light field.

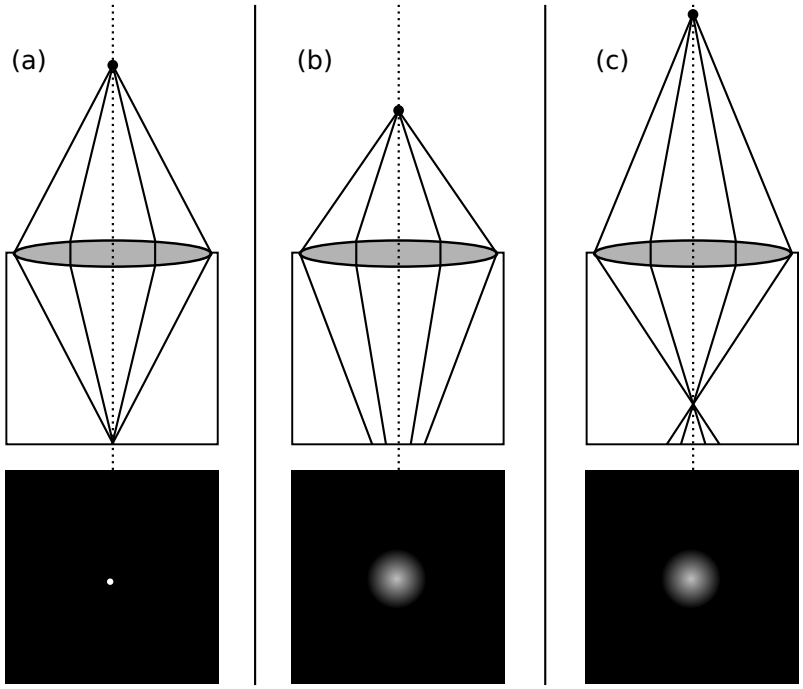


Figure 3.1: Schematic of defocusing effect with a full aperture.

The plenoptic camera does not use an eccentric aperture. Instead, a lenslet array is used to encode the angular information of incident rays onto pixels found behind each lenslet. This is illustrated in Figure 3.3 with an array of pinholes used in place of lenslets. In this case, a main lens is used to form an image on the array of pinholes with 3 x 3 pixels located behind each pinhole. As such, each pinhole represents a macropixel. When an object is perfectly focused on the center pinhole (Figure 3.3a), all of the rays converge at the pinhole illuminating all of the pixels found underneath that particular pinhole. When the object is moved closer (Figure 3.3b), however, a blurred spot is produced that spans several pinholes. As the angle of rays reaching the pinholes varies depending on the pinhole location, only certain pixels under each pinhole receive light whereas the others remain dark. This is

illustrated by the pattern of pixels found beneath each figure. Conversely, when the the object is moved further away, the angle of light rays incident on each pinhole is different, and results in a different pattern on the pixel array (Figure 3.3c). As such, by analyzing the distribution of light under each pinhole, the depth of the object can be determined. Replacing the array of pinholes with a lenslet array yields an identical result, but greatly increases the amount of light collected by the sensor.

This concept was demonstrated by Adelson and Wang [34] using a 500 x 500 pixel CCD camera with microlenses forming 5 x 5 pixel macropixels. As such, the spatial resolution of the resulting image was 100 x 100 pixels with the 25 pixels under each macropixel used to record angular information about the image. This illustrates an inherent trade-off associated with the plenoptic camera between spatial and angular resolution. Nonetheless, Adelson and Wang were able to use their prototype sensor to demonstrate the concept for rangefinding where they produced qualitatively accurate depth maps for various images.

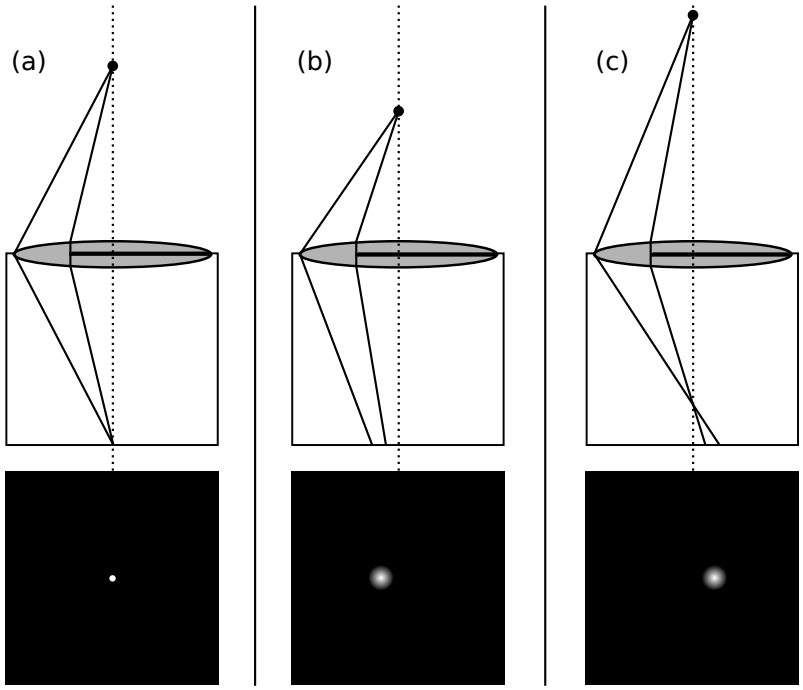


Figure 3.2: Schematic of defocusing effect with the use of an eccentric aperture.

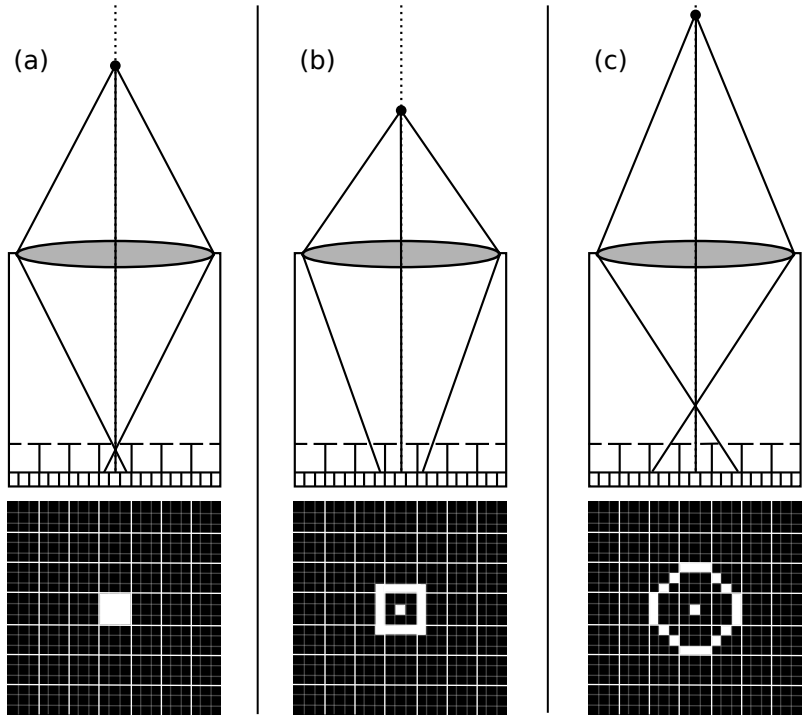


Figure 3.3: Schematic of defocusing effect with the use of an eccentric aperture.

3.2 Recent Developments

Interest in plenoptic cameras picked up recently due to the rapid increases in CCD resolution which allow both the spatial and angular information to be adequately sampled. In particular, we note the work of Ng et al. [36] who developed a hand-held version of the camera using a commercially available 16 megapixel image sensor and a lenslet array consisting of 296 x 296 lenslets. Their focus was on digital photography where the additional information made available with lightfield imaging enables computational rendering of synthetic photographs, allowing for focusing of the camera or adjustment of the aperture after the image has been taken. Also demonstrated in their work is the ability to move the observer across the aperture of the camera, which produces changes in parallax. This is particularly useful in macro (close-up) imaging as is often used in the laboratory and wind tunnel environment. The number of views available is equal to the number of pixels behind each lenslet. In their case, this corresponded to a total of 196 (14 x 14) different views of the same scene recorded on a single

sensor. This will prove to be an important point when we consider the idea of tomographic reconstruction where 196 viewpoints using separate cameras is not practical.

More recently, efforts have been underway by Levoy et al. [37, 41] to develop a lightfield microscope based on the plenoptic camera. The fundamental principle remains the same; however, their work focused on three additional challenges associated with microscopic imaging. For one, wave optics and diffraction must be considered in a microscopic environment whereas geometrical optics was sufficient before. Secondly, a typical microscope objective functions differently than a normal camera lens. Finally, most objects in microscope images are partially transparent whereas the previous effort in plenoptic photography by Ng et al. [36] had focused on scenes with opaque objects. This last point is perhaps the most relevant to this work, where illuminated particle fields are also partially transparent. This may at first appear to be a major disadvantage in reconstructing a 3-D volume, because of the contributions from scatterers located at multiple depths. Nevertheless, this imaging scenario can take advantages of developments in the field of microscopy, where there has been substantial effort at increasing the depth resolution of images acquired of transparent scenes. The technique of *deconvolution microscopy* uses knowledge of the point-spread-function (PSF) of the microscope objective to increase the depth resolution of the resulting image by removing the contributions from out-of-plane captured light. The processed images exhibit clear and defined features that are typically hidden by out-of-focus scatterers.

These prior efforts present a logical approach for determining a 3-D intensity field of a transparent scene, which will be used in this work. First, a focal stack of images is generated using the computational refocusing technique. The focal plane spacing and number of planes generated will be discussed in later chapters. Second, a thresholding, 3-D deconvolution, or limited-angle tomography algorithm can be used to modify the planes of the focal stack by removing out-of-plane contributions and yielding a volume containing sharp particle images. An analysis of these algorithms will be also be covered later in this thesis.

3.3 Plenoptic 2.0

Since the original work by Ng et al. [36] on plenoptic photography, there have been a number of additional investigators researching various extensions and modifications to the technique. Most notably, Lumsdaine and Georgiev [42, 43] have explored the tradeoff between spatial and angular resolution and developed a similar camera where the lenslet array acts as an imaging system focused on the focal plane of the main camera lens. This is in contrast to the original plenoptic cameras, which placed the lenslet array *at* the focal plane of the main lens. This change in operation can be seen in Figure 3.4. In their work, a 16 megapixel sensor was coupled with a lenslet array of 250 micron pitch and 750 micron focal length. The number of lenslets was 144 in both directions. Thus, traditional plenoptic rendering would only generate an image of 144 px x 144 px resolution. However, high resolution rendering is able to extract an image of resolution 1040 px x 976 px.

The primary drawback to this technique is that much less angular resolution is captured. At present, the amount of angular sampling required to generate clearly refocused images of *particle fields* is unknown (the high spatial frequencies in particle images may invalidate the high resolution plenoptic approach). Additionally, the rendering of high-resolution techniques is more complicated and is less intuitive. For these reasons, traditional plenoptic rendering is used throughout this work. However, as shown in [43], due to the similarities in camera operation the plenoptic 2.0 rendering algorithms can be used for traditional light field data. This presents the possibility of exploring these rendering algorithms in future studies.

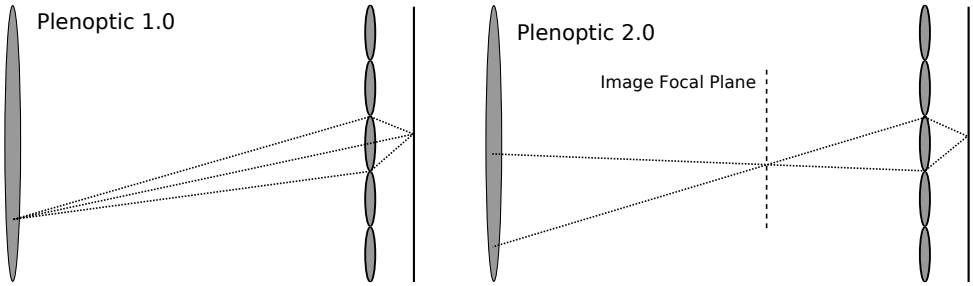


Figure 3.4: Fundamental difference in operation of original plenoptic and "focused plenoptic" cameras.

Chapter 4

Camera Simulation

To evaluate camera performance under a variety of scenarios, both a simplified 1-D and complete 2-D light field image simulator are developed. The 1-D simulator is used primarily for the purpose of understanding the basic concept and operation of the camera and identifying shortcomings or unexpected behavior. The 2-D simulator is used to render full particle fields for use in reconstruction and cross-correlation algorithms. The simulators are designed to generate a representative signal or image acquired by the camera sensor for a particle field which is approximated by a distribution of one or more point light sources. The value of this approach stems from the ability to vary parameters such as main lens focal length, lenslet pitch, lenslet focal length, and others to test and optimize camera performance. This chapter first begins with a description of the ray tracing procedure and defines the variables to be used throughout the simulations. This provides the necessary framework for constructing the 1-D simulator, which is then used to test a variety of imaging scenarios. The simulator is then extended to include diffraction effects and produce complete 2-D light field images of particle fields.

4.1 Ray Tracing Description and Optical Configuration

The use of linear (Gaussian) optics is well established for geometrically tracing the path of light through space and various optical elements through the use of matrix methods from linear algebra. An important application of Gaussian optics is ray tracing in computer graphics. Briefly, ray tracing is a rendering technique in which a large number of light rays from a scene are used to form an image at arbitrary locations or viewpoints. Rays of light are initialized at the light source by specifying an initial position and direction. Any number of

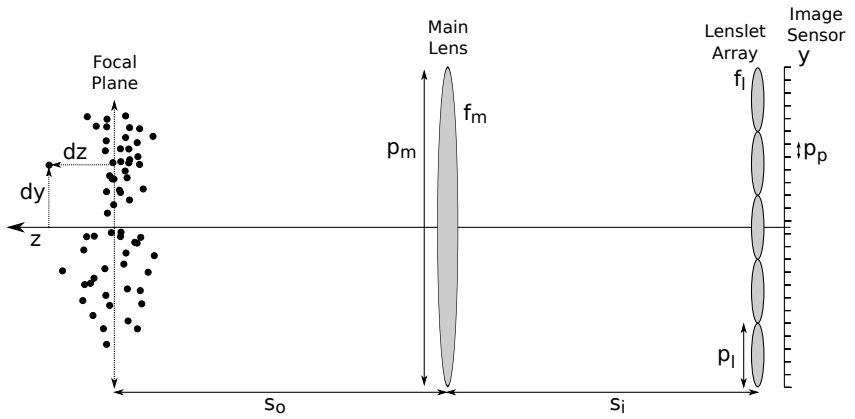


Figure 4.1: Schematic of Optical Configuration and Dimensional Definitions.

ray transfer matrices are then used to simulate optical elements and the propagation of light through free space [44]. The intersection each ray makes with a sensor plane or designated viewpoint defines the generated image. The primary limitation of traditional Gaussian optics is that ray transfer matrices assume the optical elements and coordinates are defined with respect to the optical axis. For a plenoptic camera, the lenslets are not located on the optical axis, and thus require a modified approach. Georgeiv and Intwala [45] and Ahrenberg and Magnor [46] have shown that Gaussian optics can be extended to light field imaging as well, through an extension of basic linear optics known as affine optics. The simulations constructed herein apply their work with affine optics to ray tracing, allowing lenslets to be properly simulated and light field images of entire particle fields to be generated.

In Figure 4.1, the optical elements comprising the simulation are shown (not to scale), and the corresponding dimensional variables are labeled. The origin of the optical axis is located at the center of the sensor plane, with the z-axis oriented out of the camera through the center of the lens aperture, and the x- and y-axes aligned with the sensor plane (the x-axis is projected into the page in this figure). This imaging configuration can be described using the classical thin lens equation, shown in equation 4.1. The assumption made with the thin lens equation is that the thickness of the lens is negligible with respect to the overall length of the optical system. Modern lenses with multiple optical elements are not thin by

this definition, but the concept of principle planes can reduce any combination of lenses to a single thin lens. The design of modern lenses takes this into account and allows them to be simulated using this equation.

$$\frac{1}{f_m} = \frac{1}{s_o} + \frac{1}{s_i} \tag{4.1}$$

In this equation, s_o is the distance from the object focal plane to the main lens, and s_i is the distance between the main lens and the image focal plane. The main lens of focal length f_m acts to form an image at the image focal plane, exactly like a conventional camera. However for the plenoptic camera, rather than placing the sensor at this plane, a lenslet array of focal length f_l and pitch p_l is inserted. The sensor plane, with individual pixels of pitch p_p, is then placed at the focal length of the lenslet array. For all current simulations, the fill factor of both the lenslets and the pixels are assumed to be 100%. Note again the lenslet array is located at the focal point of the main lens, and that the distance between the lenslet array and the sensor is much smaller than the distance between the main lens and the lenslet array. With the basic dimensions specified, the ray transfer matrices can now be defined.

The first and most basic ray transfer matrix is for the propagation of light through free space, or translation. This is modeled as a linear propagation of light from the original point (x, y) at an angle (θ, ϕ) over a distance t. This is expressed in matrix notation in equation 4.2. Note that only the position of the ray changes, and the direction remains the same.

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \phi' \end{pmatrix} = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ \theta \\ \phi \end{pmatrix} \tag{4.2}$$

The next basic transfer matrix used is for refraction of light rays through a thin lens, which is expressed in equation 4.3, where f is the focal length of the lens. In contrast to translation, the position of the light ray is constant, while the direction of the light ray is modified. As mentioned previously, the lens must either be thin, or capable of being reduced

to a single effective thin lens for this equation to be used. This assumption is used for all lenses used in both the 1-D and 2-D simulators.

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \phi' \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1/f & 0 & 1 & 0 \\ 0 & -1/f & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ \theta \\ \phi \end{pmatrix} \tag{4.3}$$

The previous equations for light propagation and lens transfer make the assumption that the optical element is centered with respect to the optical axis. For light field cameras using a lenslet array, a modification must be made to account for the lenslets that are shifted from the optical axis. The derivation of this approach, known as affine optics, is given in Georgeiv and Intwala [45]. The result of their derivation is to treat the propagation through a lenslet array of focal length f_l and separation from the optical axis (s_x, s_y) as the combination of a lens transfer matrix and a prism, as given below in equation 4.4. In contrast to the previous two transfer matrices, both the position and the direction of the light ray are modified.

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \phi' \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1/f_l & 0 & 1 & 0 \\ 0 & -1/f_l & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ \theta \\ \phi \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ s_x/f_l \\ s_y/f_l \end{pmatrix} \tag{4.4}$$

4.2 1-Dimensional Ray-Tracing Simulator

The 1-D simulator is used as a relatively simple means to evaluate basic camera concepts without requiring a full particle simulation and allowing the propagation of rays through the system to be easily visualized. All particles are approximated in the simulator as point sources of rays. This simplification allows the particle field to be defined as simply the coordinates of each point, which is given in terms of a shift from the optical axis dy and a shift from the object focal plane dz. A positive value of dy is oriented upward, and a positive

value of dz is oriented away and out from the camera along the optical axis as indicated in Figure 4.1.

The remaining variables are camera parameters, which can be divided into two categories: fixed parameters and variable parameters. The fixed parameters are set through the hardware design of the camera and lenslet array and include the lenslet focal length f_l, the lenslet pitch p_l, the pixel pitch p_p, and the number of pixels $n_{p,y}$. These can be modified in the simulator as needed but cannot be modified in the physical hardware once manufactured.

The other class of parameters are variable parameters, which can be changed to accommodate different imaging configurations. These include the main lens focal length f_m and the object focal plane distance s_o. These parameters could be directly varied to achieve the desired imaging conditions. However, a more pragmatic approach can be used by defining a field of view FOV_y and the main lens focal length. This allows the magnification to be calculated and enforces a unique combination of s_o and s_i in accordance with the thin-lens equation. The camera specifications can be further defined by placing a condition on the image-side or effective f-number of the main lens. As recognized by Ng et al. [36], to prevent overlap or unused pixels of the lenslet images, the image-side f-number (also termed the effective f-number) of the main lens must be equal to or larger than the f-number of the lenslets. For the main lens, the image-side f-number is calculated in terms of the magnification, as described by Smith [47] and defined in equation 4.8. This f-number is used to calculate the pitch or aperture diameter p_m of the main lens. For the lenslets, the distance s_i is much larger than the focal length of the lenslets f_l, so it can be assumed the lenslets are focused to infinity, allowing the simplified definition of f-number ($(f/\#)_l = f_l/p_l$) to be used. These quantities and relationships are defined, in order of their calculation, through the following equations.

$$M = -(n_{p,y} \times p_p)/FOV_y \tag{4.5}$$

$$s_i = f_m(1 - M) \tag{4.6}$$

$$s_o = -s_i/M \tag{4.7}$$

$$(f/\#)_m = (f/\#)_l/(1-M) \tag{4.8}$$

$$p_m = f_m/(f/\#)_m \tag{4.9}$$

The positioning of the lenslet array must now be defined. It would be computationally efficient and simple to develop an algorithm where the lenslet array begins at the exact edge of the CCD. However, in an actual camera, the lenslet array will not be perfectly aligned with the edge of the CCD and will exhibit a slight overlap over the entire sensor. To simulate this, the number of lenslets needed to completely cover the sensor is determined. The total size of the array will be slightly larger than the sensor, so an offset is calculated and applied to center the lenslet array with respect to the optical axis. The simulated lenslet array can be defined by three variables: the lenslet pitch p_l, the number of lenslets in the vertical direction $n_{l,y}$, and a vertical offset o_y. Determining the lenslet $n_{s,y}$ that a ray strikes is then a computationally efficient process of division and rounding, as given by equation 4.10. The corresponding shifts require multiplying the lenslet number by the lenslet pitch, then adding a shift to move from the corner to the center of the lenslet, as shown in equation 4.11. Note that the lenslet numbering starts at the top left corner of the sensor, which requires the rays to be shifted by an amount equal to half the sensor size. This is accounted for in the following equations.

$$n_{s,y} = \text{round}([y + n_{p,y}p_p/2 - o_y - p_l/2]/p_l) \tag{4.10}$$

$$s_y = n_{s,y}p_l + o_y + p_l/2 - n_{p,y}p_p/2 \tag{4.11}$$

With the physical parameters of the simulator defined, the actual simulation process can be presented. As mentioned previously, the simulator acts as a ray-tracing program to propagate rays of light through the main lens and lenslet array to the sensor plane. Initializing the rays for this procedure consists of two steps: the angular sweep of rays must be defined and the rays themselves must be created. The first step is important for

computational efficiency; the simulator should not allow rays to be generated which will fall outside of the camera aperture. These bounds are easily calculated through equations 4.12 and 4.13 for each particle. The computational implementation uses the `atan2` function to ensure the correct sign is returned.

$$\phi_{max} = \tan^{-1}\left(\frac{p_m/2 - dy}{s_o + dz}\right) \tag{4.12}$$

$$\phi_{min} = \tan^{-1}\left(\frac{-p_m/2 - dy}{s_o + dz}\right) \tag{4.13}$$

At this point, a correction is added to the simulator to take into account the differences in the solid collection angle of light depending on the position of the particle. The angle subtended by rays intersecting the outer circumference of the aperture is denoted as the reference angle γ_{ref}. For each particle, the subtended angles are calculated and are divided by the reference angle to determine the corrected number of rays to generate for each point, as given in equation 4.14.

$$n_{rays,corr} = \text{round}\left(\frac{n_{rays}(\phi_{max} - \phi_{min})}{\gamma_{ref}}\right) \tag{4.14}$$

A uniformly distributed random number generator is used to produce randomized angles for the ray which fall within the angular bounds defined above. Using a random distribution of angles better approximates the stochastic nature of image formation and prevents an accumulation of simulation artifacts which could occur for uniform angular distributions. With the initial position x, y, z and angle θ, ϕ of the ray defined, the tracing begins through the use of equations 4.2 and 4.3. For the case of the 1-D simulation, these are simplified to terms of y, z, and ϕ. The equations simulate propagation of the ray to the main lens, through the main lens, and to the lenslet array. At this point, the lenslet that the ray strikes must be determined using equations 4.10 and 4.11. This shift is then used in conjunction with equation 4.4 to propagate the ray through the correct lenslet and to the sensor array.

When the ray reaches the sensor, the pixel that it strikes must be determined. A similar division and rounding procedure can be used to determine the correct pixel y_{px}, as given in equation 4.15. Note that as before, the pixel coordinates begin at the top left corner of the image and are accounted for with this equation.

$$y_{px} = \text{round}\left(\frac{(p_p n_{p,y} + p_p)/2 - y}{p_p}\right) \tag{4.15}$$

This series of steps is repeated until the correct number of rays is generated for each particle. Each ray accumulates signal on the sensor to form the final image. For the 1-D simulator, this image is a 1-D array which is visualized using simple plotting.

4.2.1 Tests using the 1-D Simulator

Four distinct test cases are used to illustrate important physical concepts behind the plenoptic camera. These include a positive and negative depth change varying only dz, an in-plane displacement test varying only dy, and a case involving a blur across two lenslets due to a change in both dy and dz. These tests lead to the formation of an expression for the in-plane and out-of-plane resolution of the camera and allow the expressions to be tested.

For these test cases a simplified set of parameters is used, listed below in Table 4.1. Note that in order to fit the main lens and lenslet array into a single readable plot, the focal length of the main lens is reduced to a very small value of 1.5 mm, which reduces the values of s_o, s_i, and p_m. This does not affect the trends which will be visualized herein, and is only for display purposes. In all cases, a total of 50,000 rays were used to accumulate statistically significant signal values. For plotting purposes, only 100 of the 50,000 rays are shown in the following ray diagrams.

The first test evaluates a particle at two different depths. Figure 4.2 presents a simulation where the particle is located on the optical axis but shifted 1000 μm away from the

Table 4.1: Definition of Simulator Parameters

Lenslet Focal Length	f_l	500 μm (0.500 mm)
Lenslet Pitch	p_l	125 μm (0.125 mm)
Pixel Pitch	p_p	7.4 μm (0.0074 mm)
Number of Lenslets	$n_{l,y}$	5
Number of Pixels	$n_{p,y}$	75
Main Lens Focal Length	f_m	1500 μm (1.5 mm)
Magnification	M	-1

camera. The first vertical line from the left is the object focal plane, the second line represents the main lens, the third line is the position of the lenslet array, and the fourth line is the pixel array. To the right of the pixel array is a curve which provides a relative measure of the integrated signal at each pixel. Readily apparent in the image is the large blur spot incident on the lenslet array. Since this blur spot spans multiple lenslets, the signal is spread to multiple locations on the sensor. The incident angle of the light rays on the lenslet array determines the resulting pattern at the sensor. Figure 4.3 is an alternate case demonstrating a particle located 1000 μm closer to the camera lens. Note the different signal pattern, which is caused by the large difference in angles incident on the lenslet array. These signal patterns are a key element in plenoptic camera performance, which allows the depth of particles to be determined. Also, these two examples indicate that the depth of two in-line particles can be determined without any ambiguity.

The next test exposes a potential limitation of the plenoptic camera. Figure 4.4 presents an example where the particle is located on the focal plane with a -62 μm in-plane displacement, which forms an image at the edge of the central lenslet. Figure 4.5 locates the particle at the center of the focal plane resulting in an image being formed at the middle of the central lenslet. Finally, Figure 4.6 has a +62 μm in-plane displacement, the exact opposite of Figure 4.4. The most noticeable characteristic about these images is that the resulting signal is very nearly similar, with hardly any bias in signal level, and no signal contributions from other lenslets. This illustrates a *worst-case* scenario for the plenoptic camera; namely,

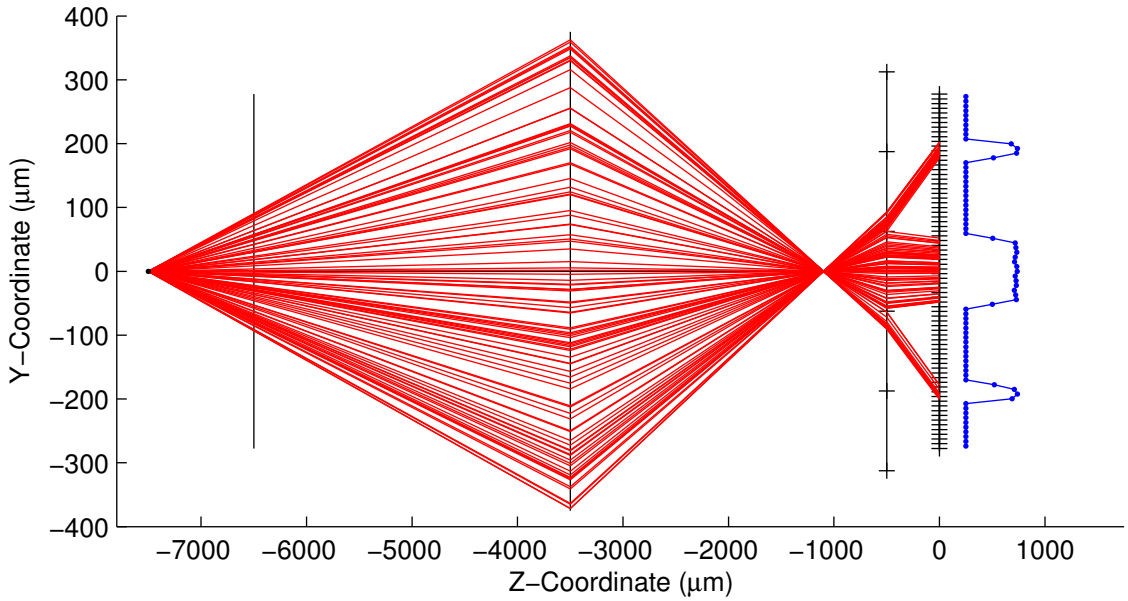


Figure 4.2: Depth testing, $dy = 0$ μm , $dz = +1000$ μm.

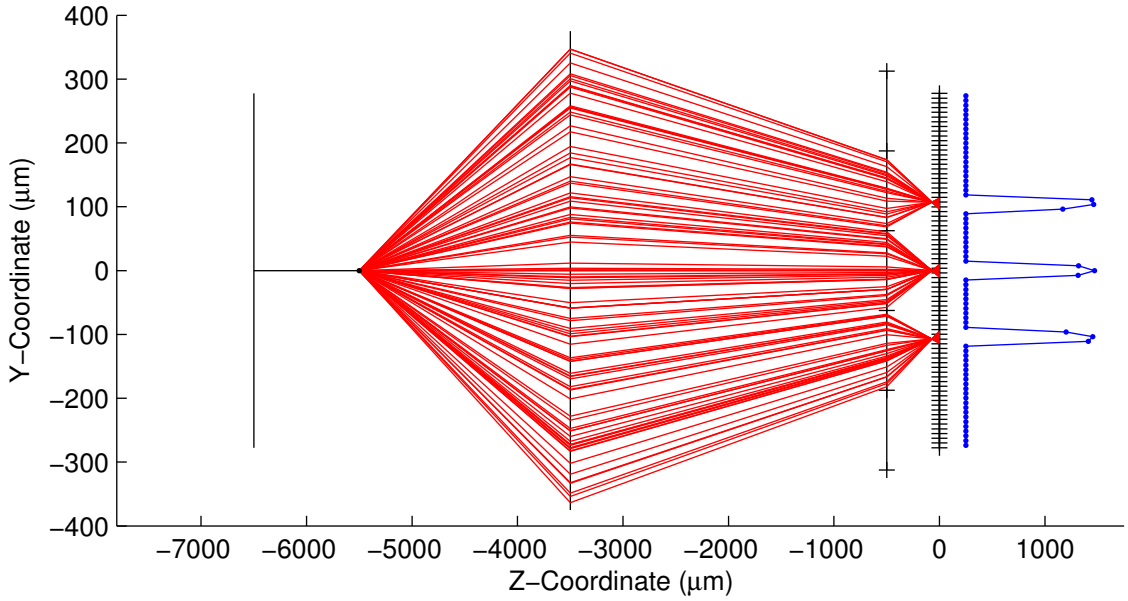


Figure 4.3: Depth testing, $dy = 0$ μm , $dz = -1000$ μm.

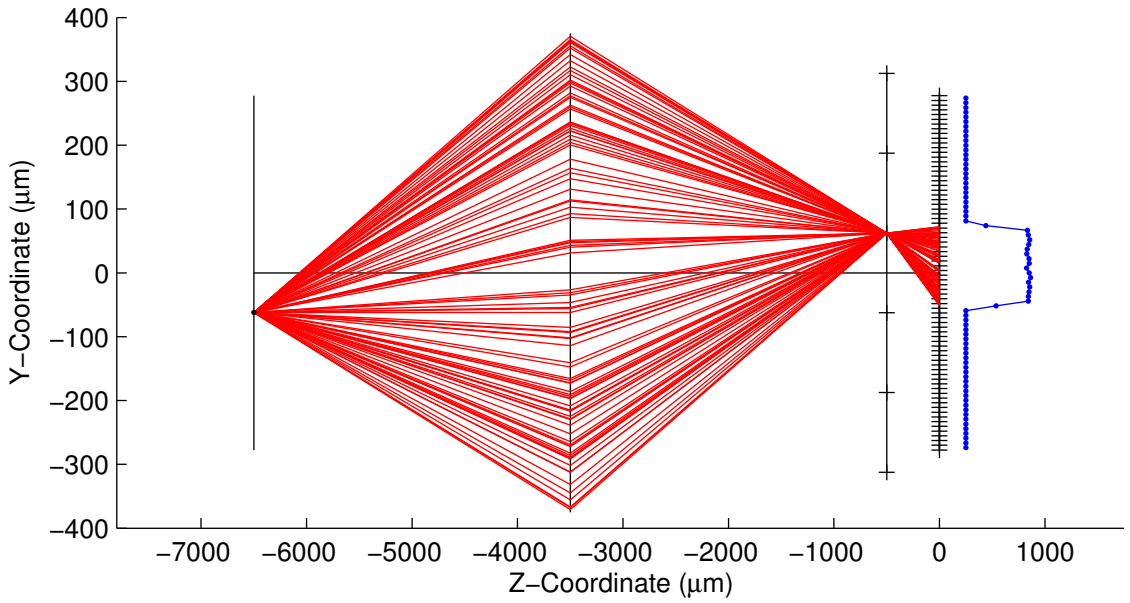


Figure 4.4: Spatial resolution testing, $dy = -62$ μm , $dz = 0$ μm.

particles that lie within the focal plane can not in general be resolved with a resolution greater than half the lenslet pitch divided by the magnification ($p_l/2M$). Figure 4.7 expands on this idea again, when the particle is moved a very small amount, however, causing the signal to dramatically change. The slight bias in edge pixels in these simulations does present a means to measure the displacement within a lenslet; however, the variation in intensities is likely too small to be captured with a detector or will be under the noise floor of the camera.

The next test illustrates a scenario for determining the spatial location of a particle with sub-lenslet accuracy. Figure 4.8 illustrates a case of where a blur spot is formed on the lenslet array. When the particle is moved 20 μm in the y-direction, a portion of the blur spot crosses over into the next lenslet and is deflected accordingly. This results in a substantial change in the resulting signal levels.

As seen in these cases, much more information is available on the location of the particle when the signal is spread out over multiple lenslets. Not only is a new set of angular information available, also the difference in signal levels is an indicator of the particle position. However, the worst-case scenario presented previously is important in quantifying

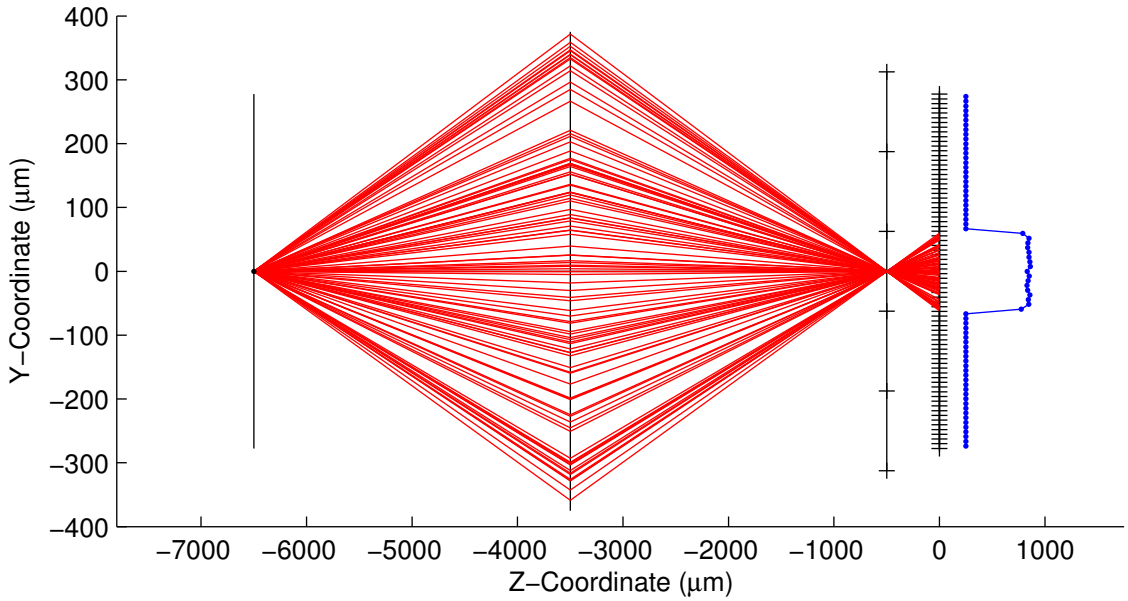


Figure 4.5: Spatial resolution testing, $dy = 0$ μm , $dz = 0$ μm.

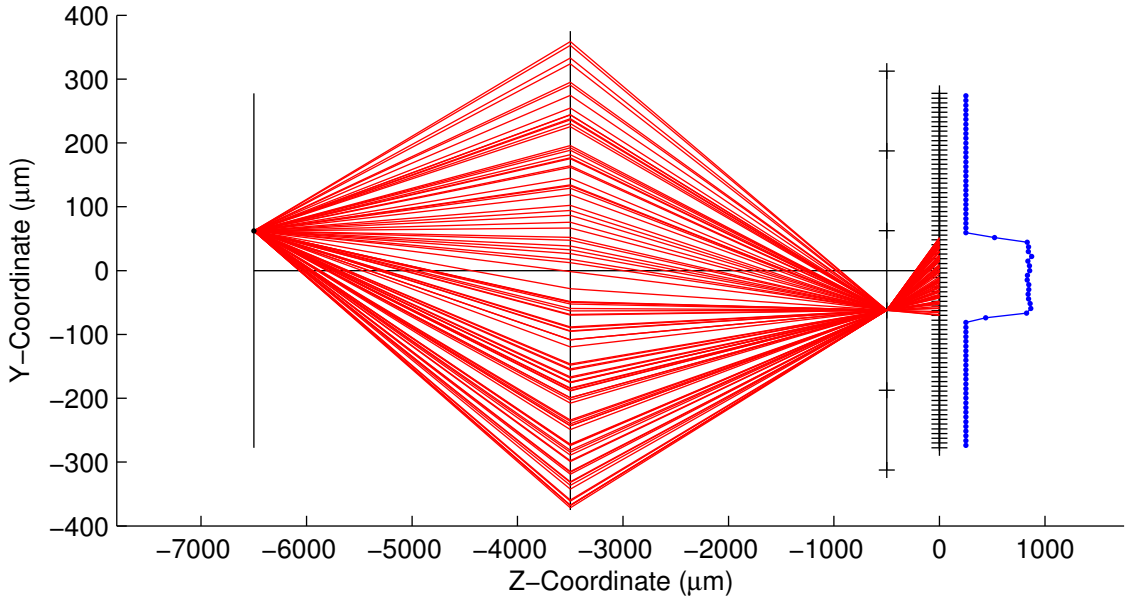


Figure 4.6: Spatial resolution testing, $dy = +62$ μm , $dz = 0$ μm.

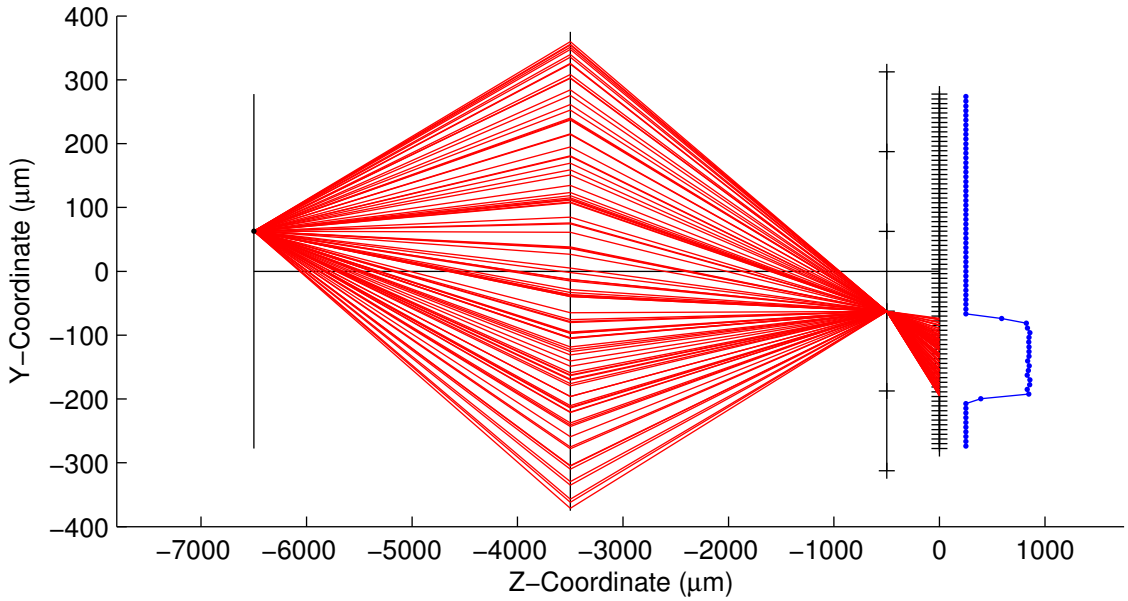


Figure 4.7: Spatial resolution testing, $dy = +63$ μm , $dz = 0$ μm.

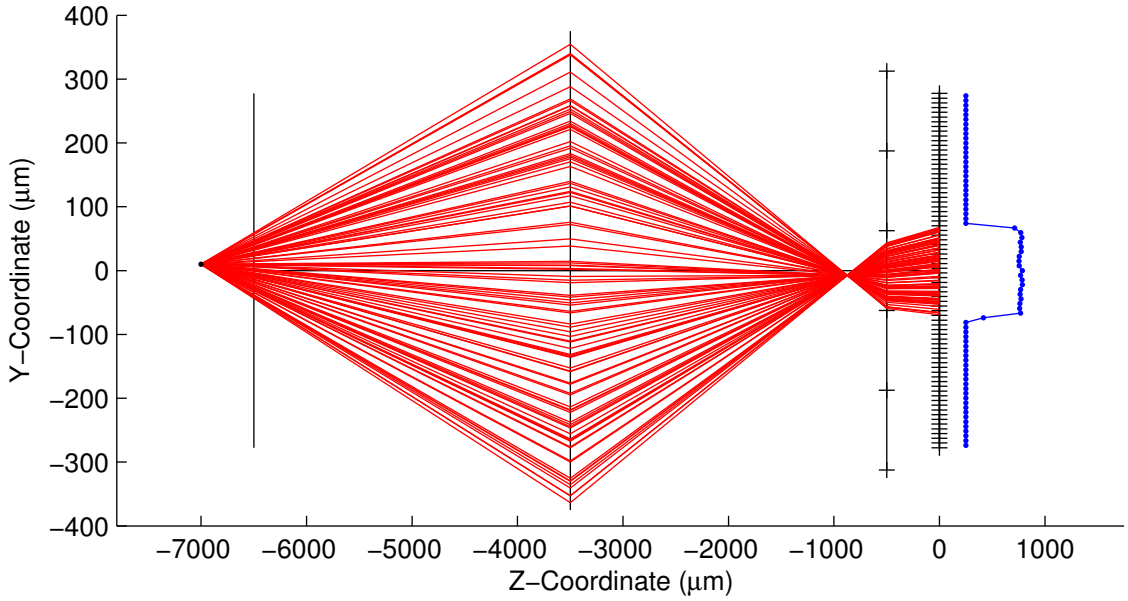


Figure 4.8: Spatial resolution testing, $dy = +10$ μm , $dz = +500$ μm.

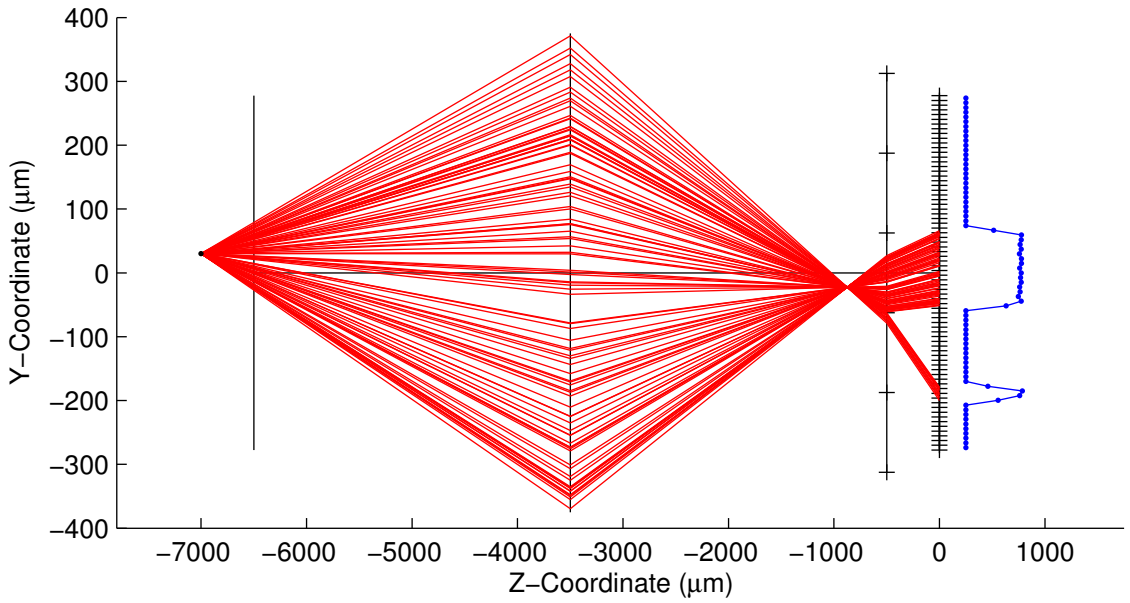


Figure 4.9: Spatial resolution testing, $dy = +30$ μm , $dz = +500$ μm.

the measurement uncertainty of the camera. Rather than using a large number of simulations to determine the bounds of the uncertainty, the depth-of-field equations can be used to generalize the procedure and will be discussed in the next section.

4.2.2 Depth of Field and Achievable Resolution

Consider the ray schematic shown in Figure 4.10, which is an illustration of the classic depth-of-field concept often used in photography. A point located a distance D_o in front of a convex lens will focus to a sharp point behind the lens at a distance D_i. A fundamental premise of geometrical optics is that the image of an infinitely small point is itself infinitely sharp (i.e., has an infinitely small spot size). However, real optics applications utilize detectors with sensor elements of a finite size. To characterize the effect this has on the optical resolution of a system, a circle of confusion c is defined as an arbitrary measure of spatial resolution at the image plane. As shown in Figure 4.10, this circle of confusion can be seen as a bounding diameter for the intersection of two light cones that originate at different locations $D_{F,o}$ and $D_{N,o}$, positioned on both sides of the original point D_o. These two spots are known

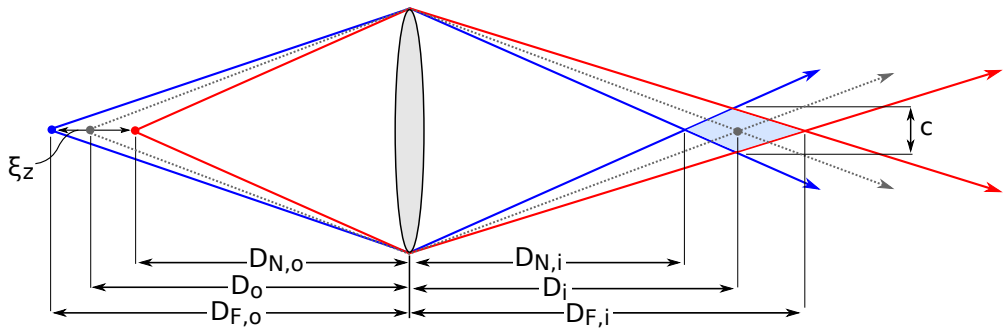


Figure 4.10: Schematic of dimensions for depth of field calculations.

as the far depth-of-field and the near depth-of-field, respectively, and the difference between them ξ_z is called the focal depth, or simply the depth-of-field. Another way to understand this concept is that if the circle of confusion is defined to be the size of a pixel on a sensor, the focal depth ξ_z indicates the smallest physical depth over which the pixel integrates light for the imaging condition defined by D_o and D_i.

This concept provides a simple way to estimate the achievable depth resolution of the plenoptic camera if the circle of confusion is defined to be the lenslet pitch (in the next chapter, this idea will be extended to give closed-form expressions for proper focal plane spacing in computationally refocused images). The depth-of-field equations can be derived using geometry and similar triangles; the resulting expressions are given in equations 4.16 through 4.18. In these equations, f is the main lens focal length, $f/\#$ is the main lens f-number, c is the circle of confusion, and s is the location of the particle in object space. The circle of confusion is defined as the lenslet pitch p_l. For a particle lying on the focus plane (i.e., $s = s_o$) and with conditions specified in Table 4.1, these equations yield $D_{N,o} = 2570$ μm (2.57 mm) and $D_{F,o} = 3600$ μm (3.60 mm), leading to a total ambiguity in depth $\xi_z = 1030$ μm (1.03 mm). The maximum in-plane ambiguity is given by equation 4.19, $\xi_y = 62.5$ μm (0.0625 mm).

$$D_{N,o} = \frac{sf^2}{f^2 + c(s-f)(f/\#)} \tag{4.16}$$

$$D_{F,o} = \frac{sf^2}{f^2 - c(s-f)(f/\#)} \tag{4.17}$$

$$\xi_z = D_{F,o} - D_{N,o} \tag{4.18}$$

$$\xi_y = \frac{p_l}{2M} \tag{4.19}$$

Now that analytical expressions have been developed for the maximum uncertainty for in-plane and out-of-plane displacement, the effect of other variables can be considered. In particular, magnification and focal length are two important parameters that can be varied in an experiment. Note that changes of magnification for a fixed focal length lens result in a change in the values of s_o and s_i. Secondly, the effect of focal length itself is analyzed for two common focal lengths: 50 mm and 200 mm. In Figure 4.11, the plot is made for a focal length of 50 mm. Perhaps intuitively, the ambiguities both increase as a less negative magnification is used (corresponding larger field of view and larger s_o). This can be visualized in the context of Figure 4.10. As the distance s_o from the main lens increases, the resulting diamond shape which defines the ambiguity will become stretched along the optical axis. The in-plane ambiguity also increases with a less negative magnification. This plot indicates that ambiguities can be minimized by using large negative magnifications (small values of s_o and small fields of view). Also, indicated in equations 4.16 and 4.17, the depth resolution is a function of f-number. For a lower f-number, the size of the ambiguity will decrease. Surprisingly, the plot in Figure 4.10 is the same for a focal length of 200 mm. This is due to the calculation of optical quantities; the diameter of the main lens aperture is increased for a 200 mm focal length, making the size of the ambiguity similar.

4.2.3 Including Diffraction Effects

The geometric optics model is an approximation to the actual behavior of light, and is essentially correct when the wavelength of light is vanishingly small. In this case, the use of discrete rays forming precisely defined spots in the image is appropriate. However, this simplified model does not take into account the realistic effects of diffraction that arise when the wavelength of light is a finite value. For a point-like image, diffraction causes the

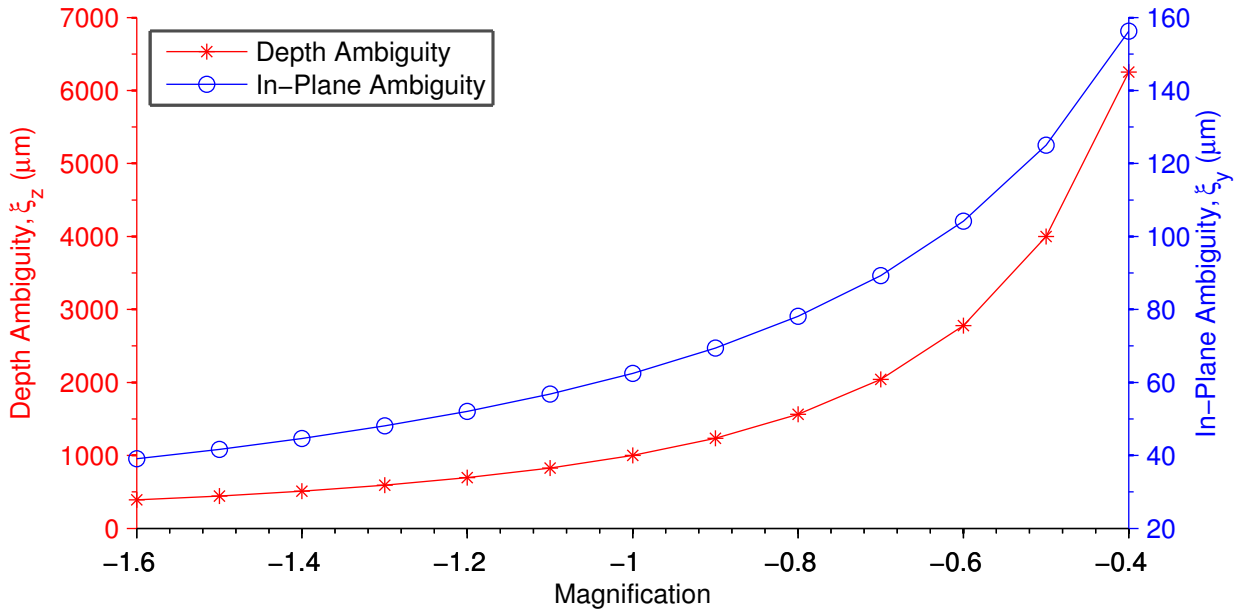


Figure 4.11: Effect of magnification on ambiguity, $f = 50$ mm.

precisely defined spot to spread and form a spot with a finite diameter. This has important implications for PIV imaging because the diffraction spot size is typically on the order of or larger than the pixel size. The effect of diffraction on general optical systems has been described by Smith [47], and is further defined for the specific case of particle imaging by Adrian and Westerweel [3]. In this section, the need for including diffraction effects is demonstrated and a method for incorporating diffraction into the simulation is discussed.

To fix thoughts, consider the current imaging configuration where the magnification $M = -1$, the f-number of the main lens $(f/\#)_m = 2$, the wavelength of light $\lambda = 532$ nm, and the diameter of a particle $d_p = 1$ μm. An approximation to the diameter of a particle image at the image plane, including diffraction effects, is given in equation 4.20, from Adrian and Westerweel [3, p. 101]. In this equation, d_s is the diffraction spot diameter given by equation 4.21. Note that this reference defines the magnification as a positive rather than a negative quantity, and thus positive magnification is used in these equations.

$$d_\tau \approx (M^2 d_p^2 + d_s^2)^{1/2} \tag{4.20}$$

$$d_s = 2.44(1 + M) \times (f/\#)_m \lambda \tag{4.21}$$

With the experimental parameters defined above, the diffraction spot size $d_s = 5.19$ μm and the resulting particle image diameter $d_\tau = 5.28$ μm. For a lenslet of 125 μm pitch, d_τ is approximately 4.2% of the lenslet diameter. At the sensor plane, the magnification must be recalculated for the lenslets themselves, $M = -f_l/s_i$. This value is close to zero due to the design of the camera. When this magnification is used in equations 4.20 and 4.21, the diffraction spot size at the sensor plane is $d_\tau = 5.2\mu$m, and the particle image diameter $d_\tau = 5.3\mu$m. This is approximately 70.5% of the pixel size based on a 7.4 μm pixel. Note that the reason both blur spots come to be the same size for both the lenslet and pixel is that the value of the magnification term is reduced by a factor of two for the pixel, which corresponds to the f-number being greater by a factor of 2 for the main lens. For this reason, diffraction effects must be included in the simulator at both the lenslet and sensor planes to produce realistic images. Also, diffraction causes an order of magnitude larger blur than the geometric image of the particle itself. Thus, only diffraction is taken into account and the negligible effect of finite particle diameter is ignored. These calculations are summarized in table 4.2.

Table 4.2: Definition of Diffraction Parameters

Diffraction-Limited Spot Size at Lenslet Array	$d_{s,m}$	5.2 μm
Std. Dev. of Normal Distribution at Lenslet Array	σ_m	0.96 μm
Diffraction-Limited Spot Size as a Percentage of Lenslet Pitch		4.2%
Diffraction-Limited Spot Size at Sensor Array	$d_{s,l}$	5.2 μm
Std. Dev. of Normal Distribution at Sensor Array	σ_l	0.96 μm
Diffraction-Limited Spot Size as a Percentage of Pixel Pitch		70.5%

The approach to implementing diffraction in this work appeals to the stochastic nature of image formation. In the ray tracing procedure, diffraction is simulated as a random shift of the spatial coordinate of the ray at both the lenslet plane and the sensor plane. The behavior of this shift is governed by the diffraction pattern of a planar wavefront passing through a circular aperture, which is then focused onto a planar surface or sensor. The

intensity pattern of the focused spot is given by the Airy function shown in equation 4.22. In this equation, r is a radial coordinate, whose origin is located at the central location of the point. D_a is the diameter of the aperture, Z_0 is the distance from the aperture to the point of focus, and J_1 is the first-order Bessel function of the first kind.

$$|h(r)|^2 = \left(\frac{\pi D_a^2}{4\lambda Z_0}\right)^2 \left[2\frac{J_1(\pi D_a r/\lambda Z_0)}{\pi D_a r/\lambda Z_0}\right]^2 \tag{4.22}$$

Equation 4.22 is the theoretical solution for the intensity distribution; however, calculation of the Bessel function is an iterative procedure which makes computation difficult and not well suited for massive computations. In contrast, the Gaussian function is calculated through simple closed-form expression and has advantages in terms of both computational efficiency and mathematical properties. Equation 4.23 is the Gaussian function presented in Adrian and Westerweel [3, p. 101]. The Gaussian function can be fit to the Airy function by setting the parameter $\beta^2 = 3.67$. A comparison of these two functions is given in Figure 4.12, showing a qualitatively good fit.

$$g(s) = ce^{\left(-4\beta^2 \frac{s^2}{d_s^2}\right)} \tag{4.23}$$

Since the Gaussian function approximates the actual diffraction pattern, we can assume that a focused particle image will have an intensity profile that is roughly Gaussian. The image is actually created from the statistical distribution of photons incident on a sensor plane. Thus, it can be understood as having been compiled by large numbers of individual photons arranged in a normal (Gaussian) statistical distribution, or probability density function (PDF). The definition of a normal distribution is generally given in terms of a standard deviation σ and a mean μ, as shown in equation 4.24. By taking the mean value to be zero and equating the exponential terms of equations 4.23 and 4.24, the standard deviation can be explicitly determined as given in equation 4.25. The values of the standard deviation at

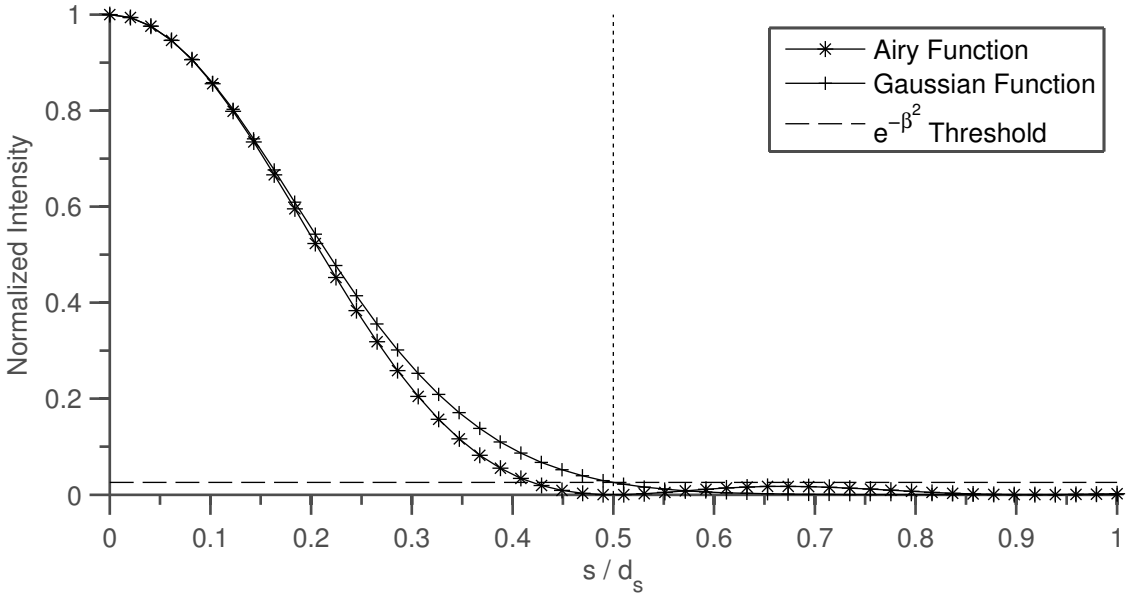


Figure 4.12: Comparison of Airy and Gaussian functions. Adapted from Adrian and Westerweel [3, p. 101].

both the lenslet and sensor plane are given in table 4.2.

$$f(x) = ce^{\left(\frac{(s-\mu)^2}{2\sigma^2}\right)} \tag{4.24}$$

$$\sigma = \sqrt{\frac{d_s^2}{8\beta^2}} \tag{4.25}$$

These definitions now allow diffraction to be introduced into the ray tracing. When a ray arrives at the lenslet plane, a normally distributed random number generator is used to generate a random shift of the spatial coordinates. Note that the mean of the random number generator is set to the initial spatial coordinate of the ray, and the standard deviation is set as defined in equation 4.25. This process is used independently in both coordinate directions. The ray is then propagated through the lenslet array using the appropriate shifts s_x and s_y as previously defined. When the ray reaches the sensor, a similar procedure is performed, but using the optical parameters of the lenslet to define d_s rather than the optical parameters of the main lens. This process is shown schematically in Figure 4.13.

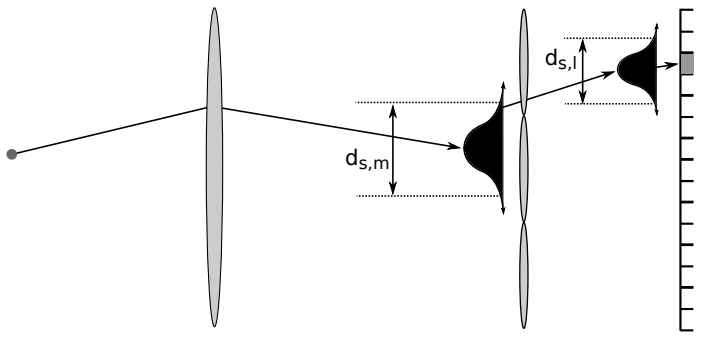


Figure 4.13: Schematic of diffraction simulation, applied at both the lenslet array and the sensor.

The algorithm was first tested in the 1-D simulator, using the MATLAB normally distributed random number generator `randn`. Figure 4.14 shows a run of the simulator using the same particle placement as Figure 4.6. The effect of diffraction is to spread the signal near the edge of the central lenslet to the adjoining lenslet. Potentially this effect could be exploited to reduce the ambiguity in spatial resolution by an amount equal to twice the diffraction-limited spot size at the lenslet plane. It also suggests that another method for reducing the ambiguity is to intentionally blur the image using a ground glass diffuser. These are commercially available for photographers, but will not be considered in this work primarily because they typically reduce the light level by 10-20%.

4.3 2-Dimensional Ray-Tracing Simulator

The simulator is now extended to produce 2-D images of multiple particles distributed within a volume. The ray tracing procedure is similar to the procedure developed earlier; however, the scale of the computation is much larger and a separate approach is needed. Consider the previous 1-D simulator, which traced 50,000 rays for a single particle. For the 2-D simulator, the same number or more rays are used, but the number of particles is much larger, in many cases greater than 1,000. Thus, the computational effort can be orders of magnitude greater. For this reason, the 2-D simulator has been concurrently programmed both in MATLAB to validate the computational algorithm and Fortran to provide increases

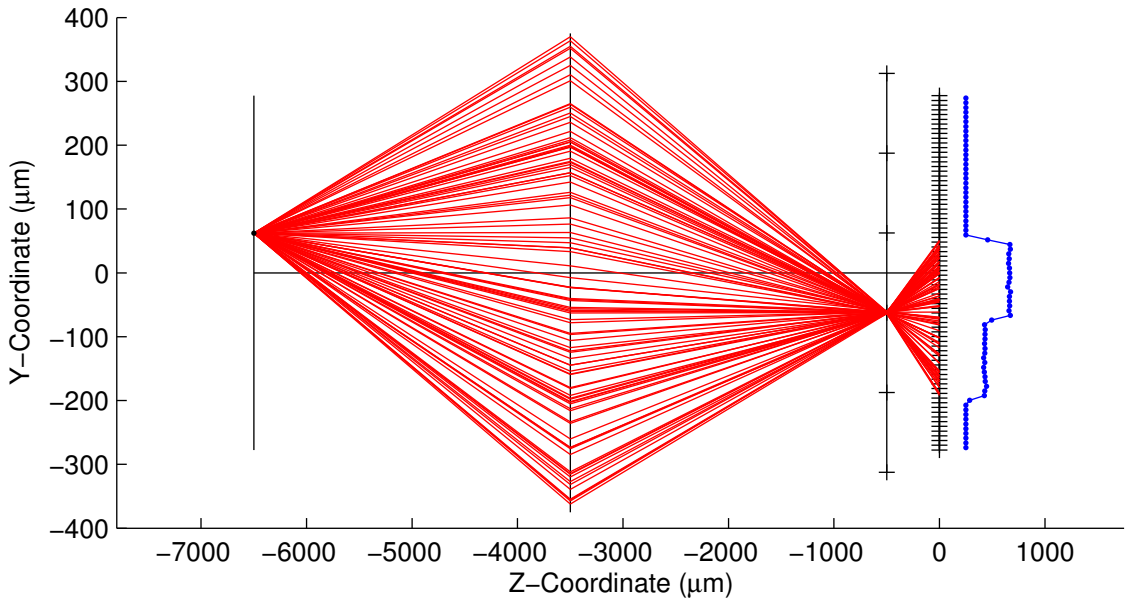


Figure 4.14: Diffraction testing using the same conditions as tested in Figure 4.6, $dy = +62\ \mu$m $, dz = 0\ \mu$m.

in speed. The Fortran version is supplied in the appendix. The use of Fortran requires the optical configuration and particle field to be generated and saved in separate text files to be processed by the simulator. The MATLAB code outputs directly to a TIFF format image file, while the Fortran code outputs an ASCII file which can be converted to an image file after processing.

4.3.1 Optical Configuration

The optical configuration for the 2-D simulator proceeds along the same lines as the 1-D simulator. An assumption made to simplify the extension to 2-D is the use of both square lenslets and square pixels; therefore, the pixel pitch in the x-direction is equal to that in the y-direction ($p_{p,x} = p_{p,y}$) and the lenslet pitch is equal in both directions ($p_{l,x} = p_{l,y}$). Therefore, the pixel and lenslet pitches can be defined as before with the variables p_p and p_l. This is a valid simplification as the vast majority of CCD sensors utilize square pixels, and the lenslet arrays considered for the current work are comprised of square lenslets.

Another modification is made to take into account the aperture stops on most common lenses. Equation 4.8 will return an exact value for the f-number; however, most lenses are only offered with a discrete number of f-number settings, known as f-stops. Thus, a list of common f-stops is used so that the f-number is rounded up to the nearest commonly available f-stop (i.e., an exact f-number of 3.2 would be rounded up to 4 rather than the nearest value of 2.8). This ensures there is no overlap in the lenslet images.

For fast computation the layout of the lenslet array is defined as a square grid (in addition to the individual lenslets themselves being square). Note that a hexagonal layout achieves an approximately 20% increase in spatial resolution by reducing unusable space between each lenslet; however, no manufacturer was able to provide a hexagonal lenslet array which forced the use of an array arranged in a square grid. The simulated lenslet array can be defined by five variables: the lenslet pitch p_l, the number of lenslets in the horizontal direction $n_{l,x}$, the number of lenslets in the vertical direction $n_{l,y}$, a horizontal offset o_x, and a vertical offset o_y. The offsets allow the array to be made larger than the sensor and then centered, similar to the 1-D simulator. Determining the lenslet that a ray strikes is then a computationally efficient process of division and rounding. The corresponding shifts require multiplying the lenslet number by the lenslet pitch, then adding a shift to move from the corner to the center of the lenslet, as defined previously in equations 4.10 and 4.11.

For visualization purposes, the center locations of the lenslets l_x and l_y can be calculated using equations 4.26 and 4.27, respectively. Figure 4.15 shows the upper-left and lower-right corners of the lenslet array and sensor. In this figure, the array covers all pixels of the sensor, and along the edge only portions of the individual lenslets overlay the sensor. This approximates how a real lenslet array would be aligned over a sensor.

$$l_x = n_x p_l/2 + o_x + p_l/2 \tag{4.26}$$

$$l_y = n_y p_l/2 + o_y + p_l/2 \tag{4.27}$$

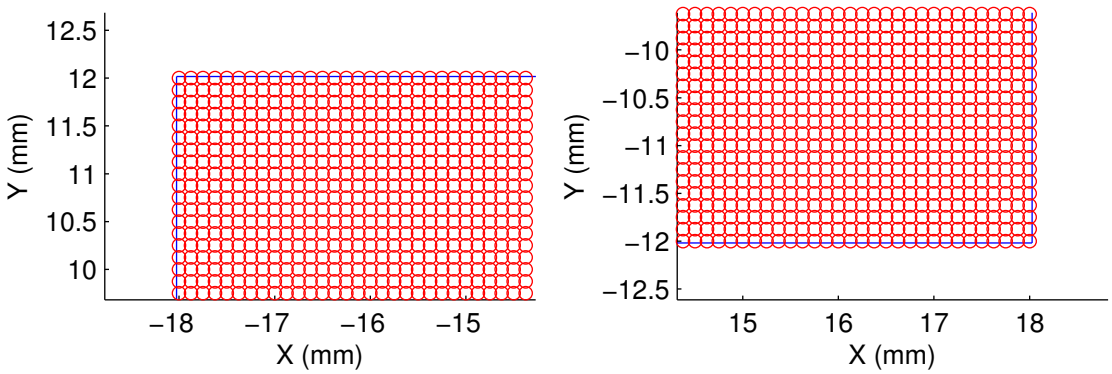


Figure 4.15: Schematic of the lenslet array (red) overlaying the edge of the CCD array (blue).

A summary of these and other optical parameters is given in table 4.3. Unless otherwise noted, these settings will be used for all 2-D simulations herein. The script outputs this information to an ASCII file which is used as an input to the Fortran ray-tracing program.

Table 4.3: Definition of 2-D Simulator Parameters, based on Kodak KAI-16000 sensor.

Lenslet Focal Length	f_l	0.500 mm (500 μm)
Lenslet Pitch	p_l	0.125 mm (125 μm)
Pixel Pitch	p_p	0.0074 mm (7.4 μm)
Number of Pixels, X-Direction (Horizontal)	$n_{p,x}$	4872
Number of Pixels, Y-Direction (Vertical)	$n_{p,y}$	3248
Sensor Size, X-Direction (Horizontal)		36.05 mm
Sensor Size, Y-Direction (Vertical)		24.04 mm
Number of Lenslets, X-Direction (Horizontal)	$n_{l,x}$	289
Number of Lenslets, Y-Direction (Vertical)	$n_{l,y}$	193
Main Lens Focal Length	f_m	50 mm
Magnification	M	-1

4.3.2 Particle Field Generation

Similar to the 1-D simulator, all particles are approximated as point sources of rays. The generation of particles is kept separate from the optical configuration script to allow different optical configurations to be tested with identical particle fields. A MATLAB script is used to generate the particle field, starting with two variables n_{pts} and n_{rays} which define the number of particles to generate, and the number of rays which will be simulated for

each particle. The particle field is defined over a volume given by the bounds x_{span}, y_{span}, and z_{span}. A schematic of this configuration is shown in Figure 4.16. Note that the z-axis is the optical axis of the system, and at $z = 0$, the x-y plane is defined as the object focal plane of the camera. Thus the origin of this coordinate system is the intersection of the focal plane and the camera optical axis. In addition to the coordinates of the particle, an intensity coefficient is defined for each particle to simulate variations in particle scattering. The intensity variation is defined by the bounds i_{min} to i_{max}.

Using this information, the particle field coordinates and intensity coefficients are created using a uniformly distributed random number generator. The output is written to an ASCII text file, with the first two lines defining n_{pts} and n_{rays}, followed by the particle information written out in four-column format specifying x, y, and z coordinates and the intensity coefficient, i.

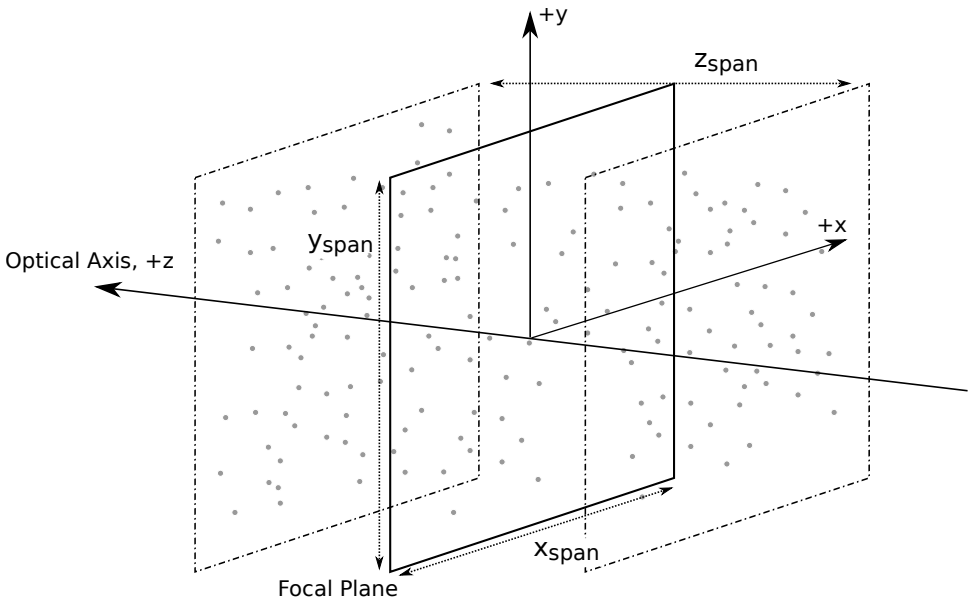


Figure 4.16: Particle field definitions.

4.3.3 Tests using the 2-D Simulator

Multiple test cases are used to show the 2-D performances of some test cases used earlier in the 1-D simulator. These include a positive and negative depth change varying

only dz and an in-plane displacement test varying only dy. Following these tests, fields of multiple particles will be simulated, first involving particle fields with no depth and then being extended to finite depths.

Figure 4.17 represents the 2-D image of an in-focus particle. In this case, a single lenslet is illuminated, as demonstrated earlier for the 1-D case. The light lines in the image represent the individual lenslets. The next extension comes by adding a change in depth. Figure 4.18 shows cases of movements farther and closer to the camera. The image of the particle is spread over multiple lenslets. Notably, the pattern is unique at each position, and is unique for identical magnitudes of positive and negative depth.

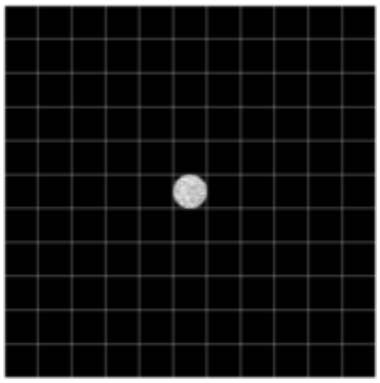
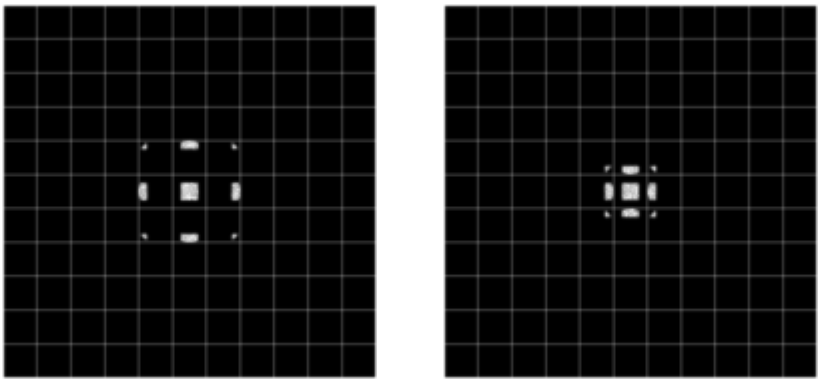


Figure 4.17: In-focus particle image ($dx = 0$, $dy = 0$, and $dz = 0$). Lenslet array outline given in light gray.

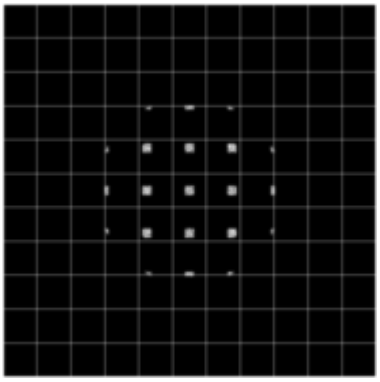
Changes in in-plane motion are shown in Figure 4.19. The left column of this figure shows an in-focus particle. In particular, 4.19b shows diffraction spreading some signal to a neighboring lenslet. The right column shows the motion of an out-of-focus particle. The motion of the particle can be determined more accurately because the signal is no longer constrained by the ambiguity over a single lenslet. This indicates the potential of sub-lenslet resolution in certain cases.

Another example of the robustness of the plenoptic approach is with two in-line particles. Figure 4.20 shows images of two in-line particles being displaced from each other a certain extent. The unique patterns on the signal indicate ways to determine the depth of each particle. This will be explored again when the refocusing algorithm is demonstrated.

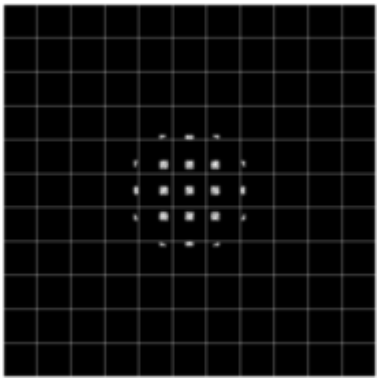


(a) $dz = +1$ mm

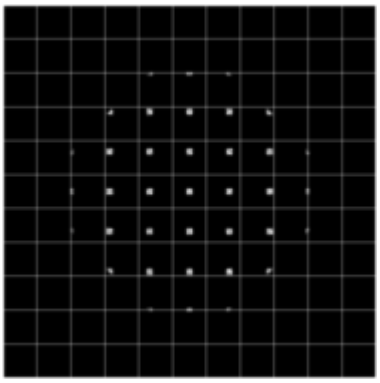
(d) $dz = -1$ mm



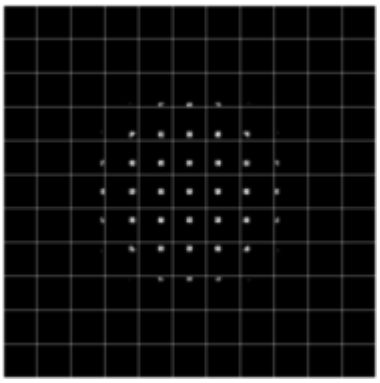
(b) $dz = +2$ mm



(e) $dz = -2$ mm



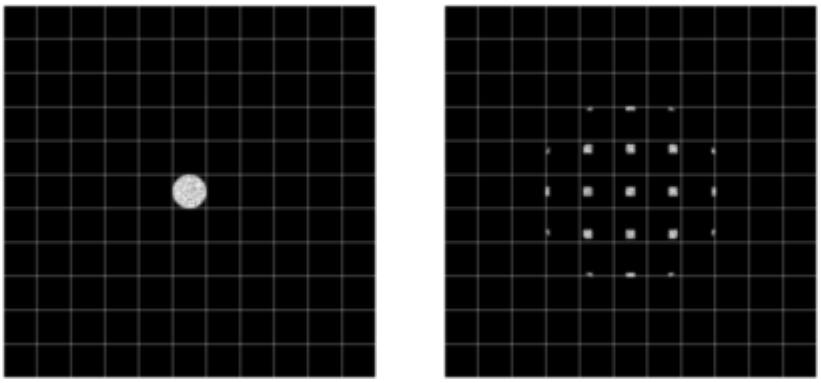
(c) $dz = +3$ mm



(f) $dz = -3$ mm

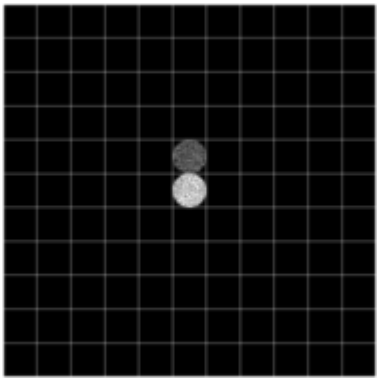
Figure 4.18: Effect of a depth change on a single particle image. Lenslet array outline given in light gray. In all cases, $dx = 0$ and $dy = 0$.

The next images demonstrate particle field images. Figures 4.21 through 4.24 show particle fields with various values of z_{span}. Note the unique patterns formed in each image, which are significantly different than a corresponding 2-D image. The light field algorithms

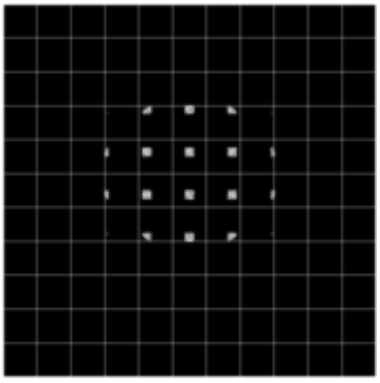


(a) $dy = +0$ mm

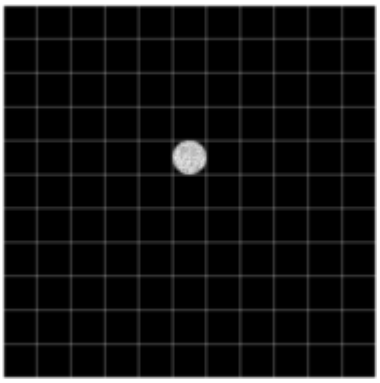
(d) $dy = +0$ mm



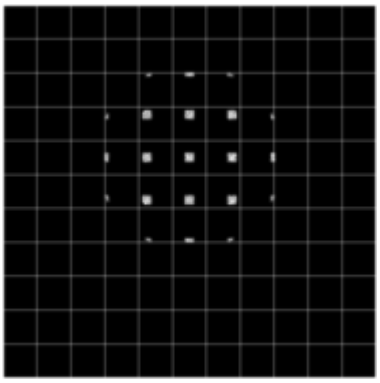
(b) $dy = +0.62$ mm



(e) $dy = +0.62$ mm



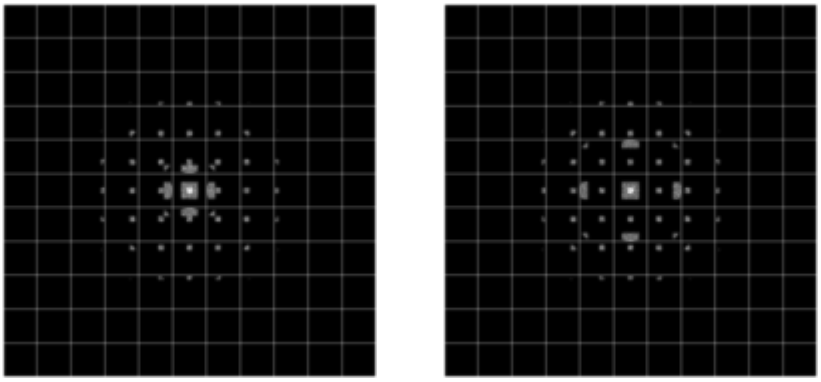
(c) $dy = +0.125$ mm



(f) $dy = +0.125$ mm

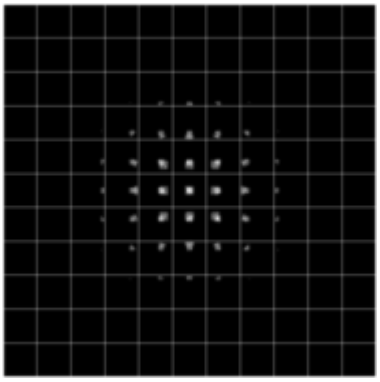
Figure 4.19: Effect of an in-plane and depth change on a single particle image. Lenslet array outline given in light gray. In the left column, both $dx = 0$ and $dz = 0$. In the right column, $dx = 0$ and $dz = +2$ mm.

to be discussed in the next chapter will be capable of extracting individual planes from this data.

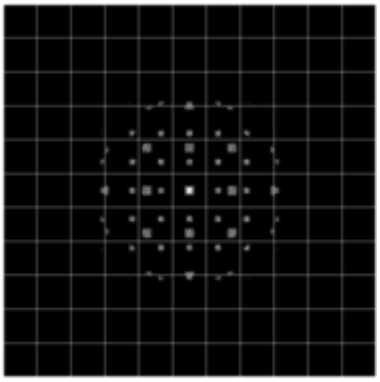


(a) $dz = -3, -1$ mm

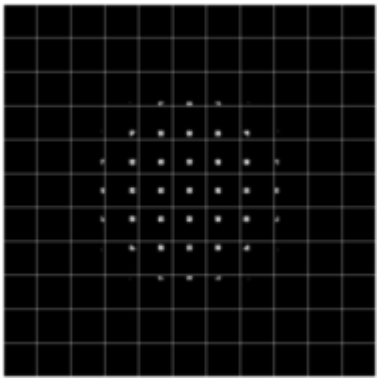
(d) $dz = -3, +1$ mm



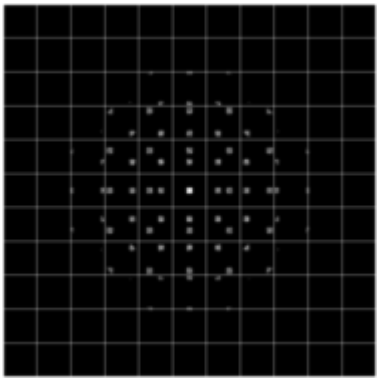
(b) $dy = -3, -2$ mm



(e) $dz = -3, +2$ mm



(c) $dy = -3, -3$ mm



(f) $dz = -3, +3$ mm

Figure 4.20: Example of two in-line particles, where the depth of the two particles are individually varied. Lenslet array outline given in light gray. In all cases, $dx = 0$ and $dy = 0$.

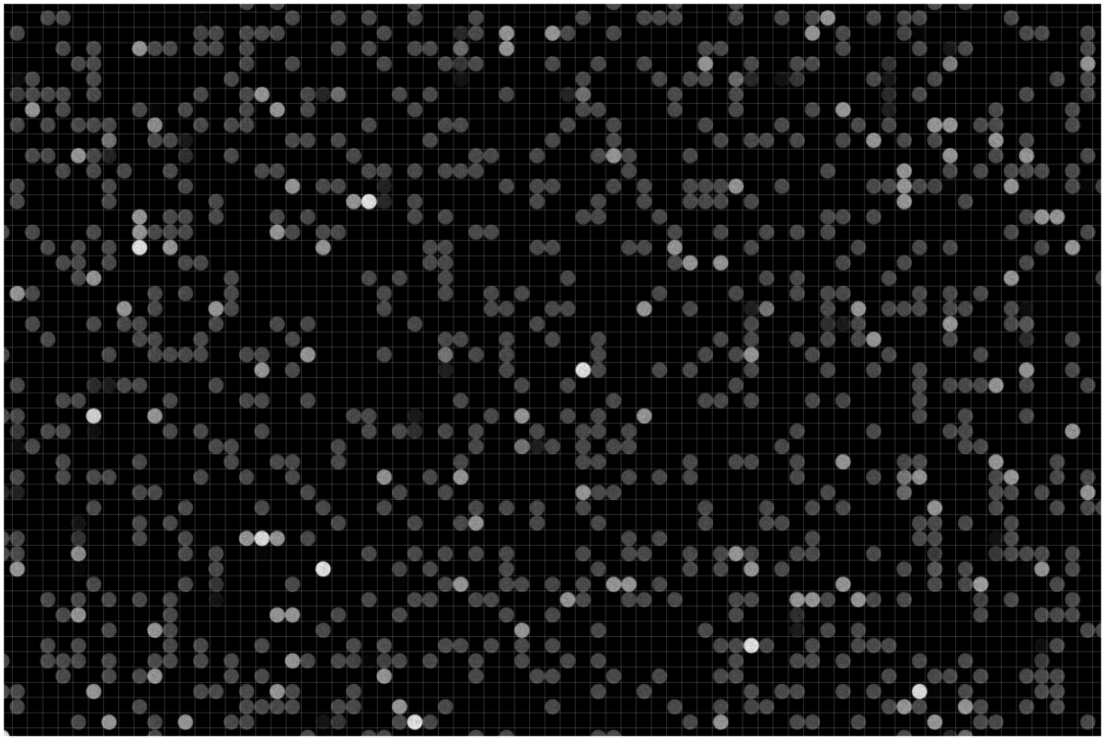


Figure 4.21: In-focus particle image ($z_{span} = 0$). 1,000 particles simulated with 25,000 rays each.

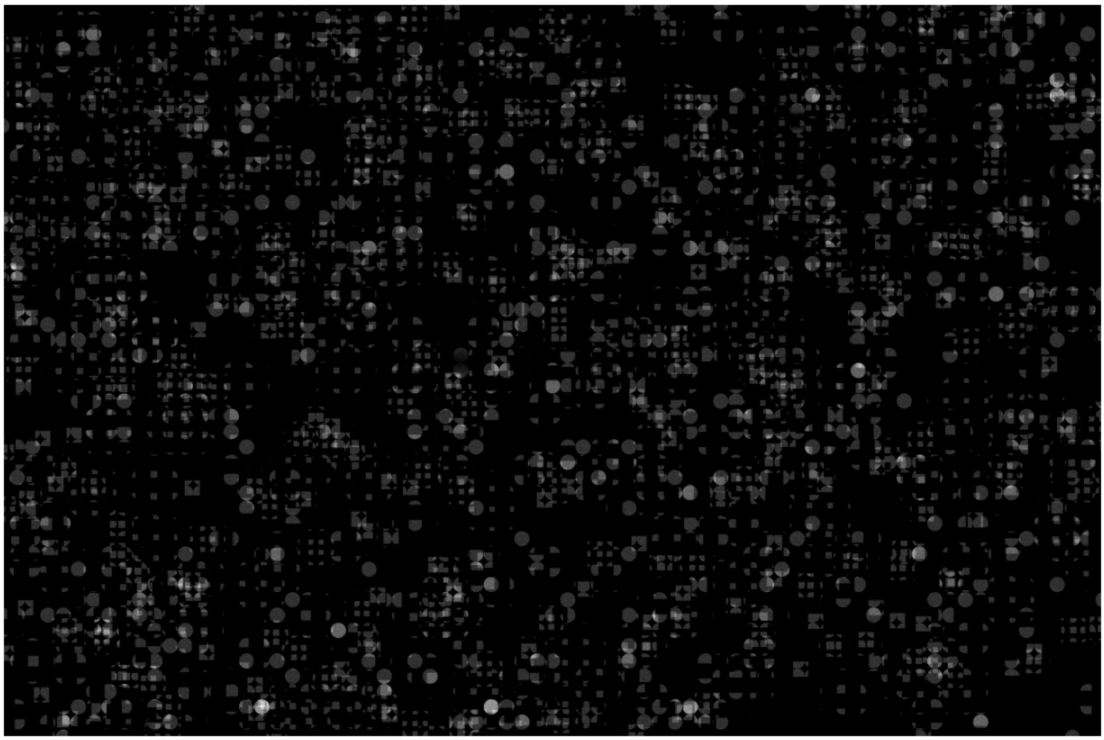


Figure 4.22: Particle image ($z_{span} = 3.0$ mm). 1,000 particles simulated with 25,000 rays each. The lenslet grid outline has been removed to improve clarity.

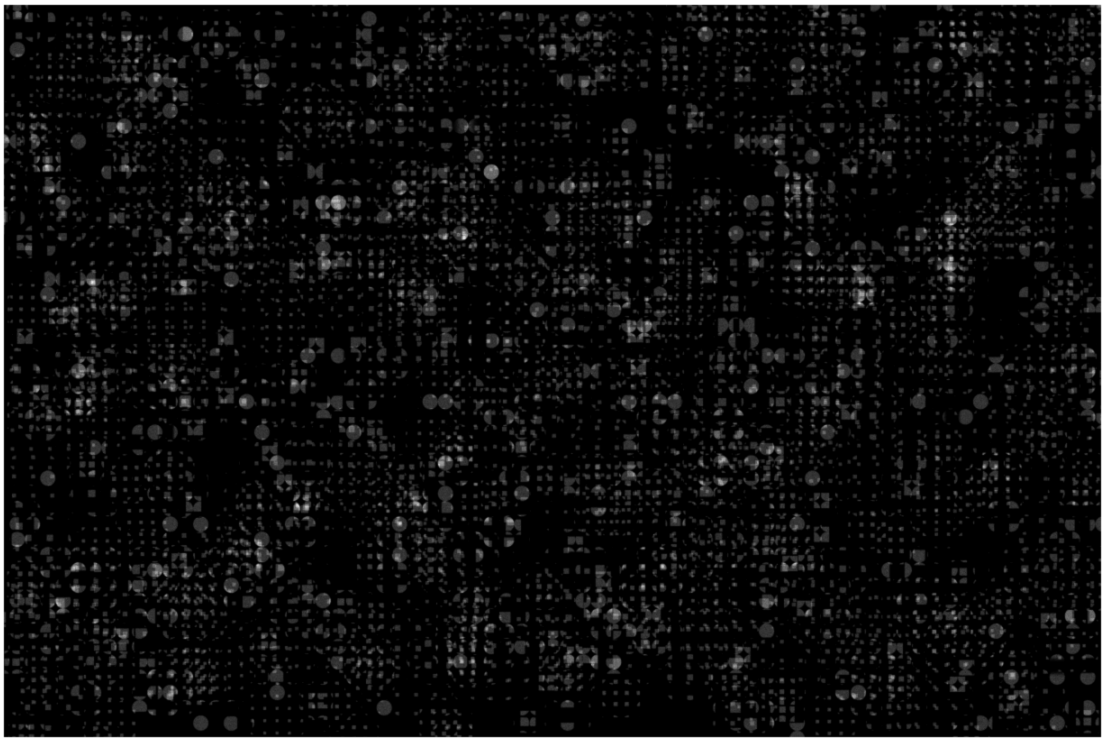


Figure 4.23: Particle image (z_{span} = 5.0 mm). 1,000 particles simulated with 25,000 rays each. The lenslet grid outline has been removed to improve clarity.

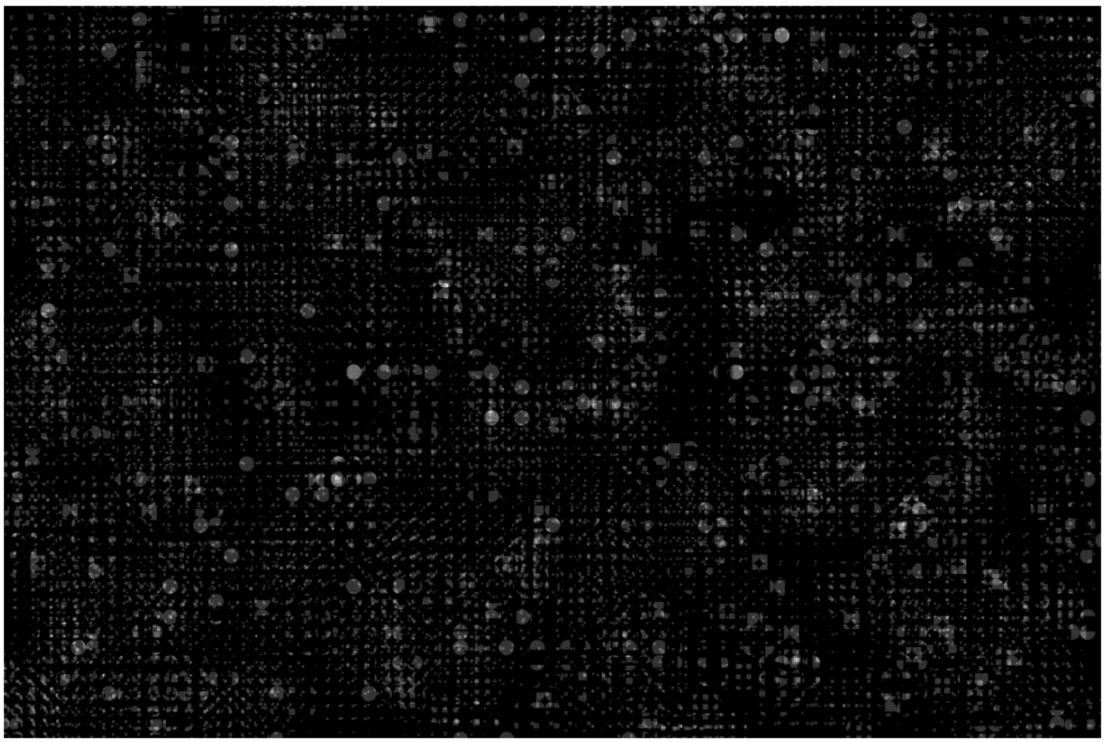


Figure 4.24: Particle image (z_{span} = 7.0 mm). 1,000 particles simulated with 25,000 rays each. The lenslet grid outline has been removed to improve clarity.

4.3.4 Angular Statistics

As an aside, the quantum efficiency of CCD cameras has an angular dependence because most modern sensors incorporate a microlens array positioned at the sensor surface to increase fill factor. For example, the Kodak KAI-16000 sensor equipped with a microlens array has a substantial reduction in relative quantum efficiency when light rays are incident on the detector by over +/- 10 degrees [48]. Figures 4.25 and 4.26 are histograms showing the distribution of angles at the sensor plane. Note that in both directions, a reasonable fraction of light rays are at angles greater than 10 degrees when arriving at the sensor. This indicates a microlens array could distort or attenuate the resulting signal values by a small amount. This analysis was done only for the specified conditions that have been simulated, and there may be scenarios where the angular distribution is more extreme. As a worst-case bound on the angle incident on the sensor, equation 4.28 can be used. For the conditions specified in table 4.3, the maximum angle is approximately 19 degrees, which is consistent with the histogram values.

$$\gamma_{max} = \frac{p_l}{2f_l} + \tan^{-1}\left(\frac{\sqrt{(n_{p,x}p_p/2)^2 + (n_{p,y}p_p/2)^2}}{s_i}\right) \tag{4.28}$$

It should be noted that this could be alleviated with the use of a field lens, as shown by Adelson and Wang [35]. The purpose of this lens is to eliminate the "bulk angle" described in the next chapter, which would reduce the magnitude of the angles at the edge of the sensor.

4.4 Simulating Particle Displacements

Evaluating the velocity field requires two snapshots of the particle field to be acquired. The displacement of the particles and the time interval Δt between the two snapshots allows the velocity to be calculated. To simulate this process, two light field images are rendered

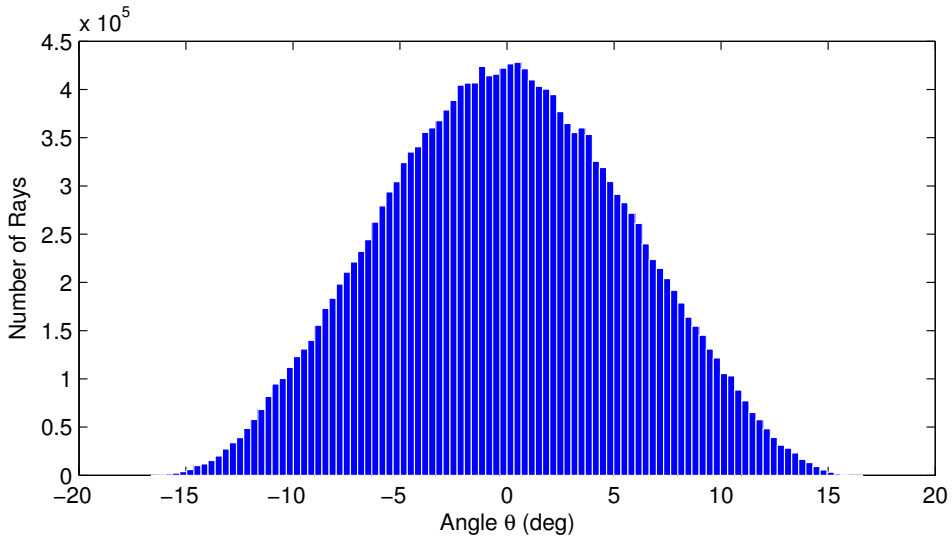


Figure 4.25: Distribution of θ angles in degrees for a 1,000 particle simulation.

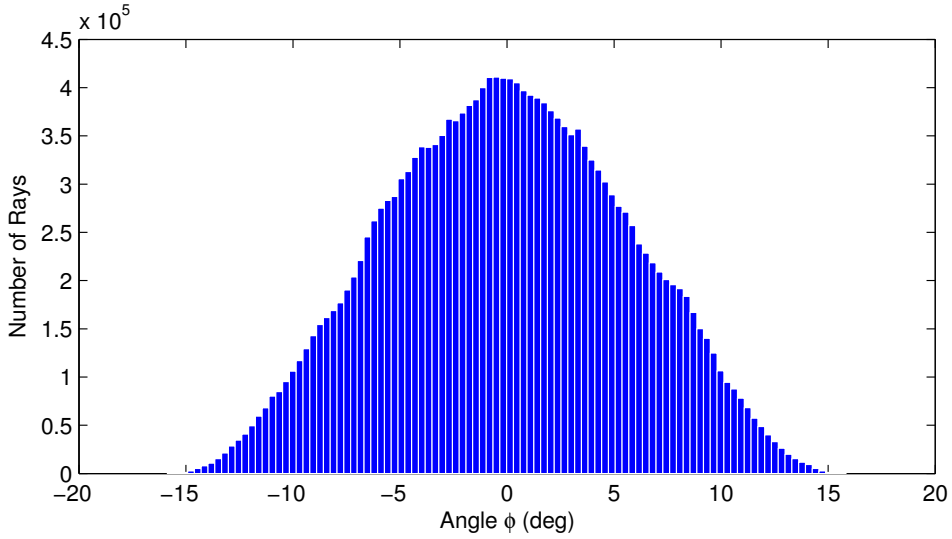


Figure 4.26: Distribution of ϕ angles in degrees for a 1,000 particle simulation.

which correspond to an initial particle field and a displaced particle field. A separate MATLAB script was used to read the original file created by the particle field generation script (Section 4.3.2), and apply displacements to each particle in accordance with a velocity field specified by one or more equations.

In this work, two simulated flow fields are evaluated. The first is a uniform flow field, which can be defined as a constant displacement for each particle in any of the three coordinate directions. Using uniform displacement provides a simple way to evaluate the resolution of the technique, including minimum detectable displacement for both in-plane and depth motions. Another advantage is that a uniform displacement field does not contain shear, which allows the accuracy of the technique to be evaluated using very basic PIV correlation algorithms.

The second flow field used for this work is a simulated vortex ring, which was used in the initial tests of both Tomographic and Synthetic Aperture PIV. Vortex rings can be easily defined using closed-form equations and exhibit large-scale 3-D structure which is important for evaluating camera performance along all coordinates. For these reasons, the Hill's spherical vortex is used as the reference flow field for this work. This is a particular solution of the vorticity equation for the generation of a steady inviscid vortex ring in which non-zero values of vorticity are confined within a sphere. The theoretical approach and derivation of the stream functions for this flow are given by Green [49]. By taking the derivatives of the stream function the velocity field can be obtained. To simplify the equations, the derivation of these equations is based on a cylindrical coordinate system in which the two primary coordinates are the distance R perpendicular to the induced motion, and the distance Z parallel to the induced motion. The origin of this coordinate system is the center of the vortex. The size of the vortex based on the diameter of the vorticity-containing sphere is denoted as a, and the induced velocity is given as u_0. The stream function is a set of piecewise equations defined independently for flow internal to the vortex and external to the vortex. The internal velocity field is given by equations 4.29 and 4.30, and the external velocity field is given by equations 4.31 and 4.32.

$$u_{int} = \frac{3}{2} u_0 \left(1 - \frac{2R^2 + Z^2}{a^2} \right) \tag{4.29}$$

$$v_{int} = \frac{3}{2} u_0 \left(\frac{ZR}{a^2} \right) \tag{4.30}$$

$$u_{ext} = u_0 \left[\left(\frac{a^2}{Z^2 + R^2} \right)^{5/2} \left(\frac{2Z^2 - R^2}{2a^2} \right) - 1 \right] \tag{4.31}$$

$$v_{ext} = \frac{3}{2} u_0 \left(\frac{ZR}{a^2} \right) \left(\frac{a^2}{Z^2 + R^2} \right)^{5/2} \tag{4.32}$$

To combine the solutions from the internal and external velocity fields, a simple test is performed by equations 4.33 and 4.34 to check if the coordinate location being evaluated lies within the bounds of the vortex, given by the vortex radius a. These statements are taken to be boolean, which will evaluate to either 0 or 1. These values are used in equations 4.35 and 4.36 to give the final value of velocity.

$$t_{int} = (R^2 + Z^2) \leq a^2 \tag{4.33}$$

$$t_{ext} = (R^2 + Z^2) > a^2 \tag{4.34}$$

$$u = u_{int} t_{int} + u_{ext} t_{ext} \tag{4.35}$$

$$v = v_{int} t_{int} + v_{ext} t_{ext} \tag{4.36}$$

With the velocity defined, the displacement of the particles can be calculated if a time interval Δt is specified. The general equation for particle displacement is actually a Taylor series expansion, as indicated in equation 4.37. Unfortunately, this is an infinite series that requires multiple derivatives of the velocity field to be calculated (typically only a few terms are needed, but the derivatives can still be complicated expressions). The approach used instead is to divide the total time step into a large number of smaller time steps (typically 1,000) to approximate the actual displacement.

$$x' = v\Delta t - x\frac{\partial v}{\partial x} + \frac{x^2}{2}\frac{\partial v^2}{\partial^2 x} - \frac{x^3}{6}\frac{\partial v^3}{\partial^3 x} + \ldots \tag{4.37}$$

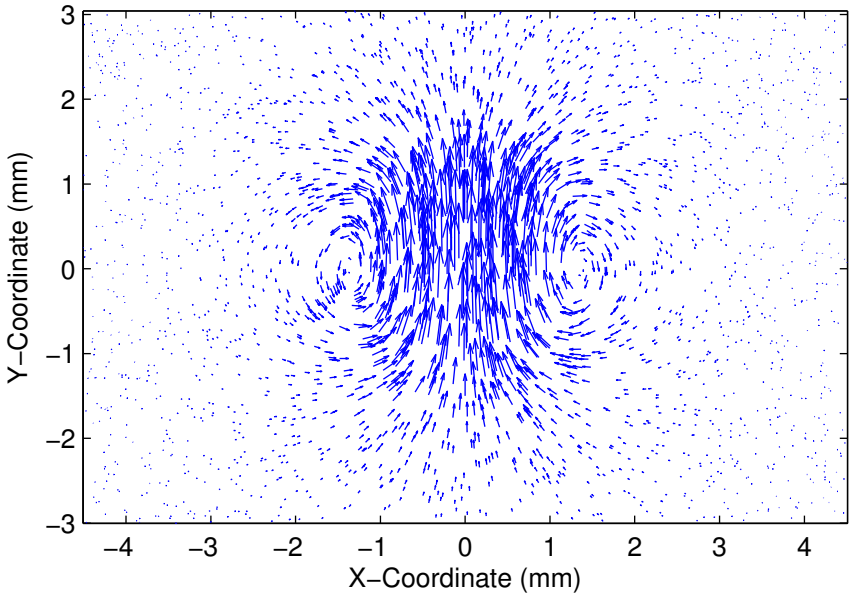


Figure 4.27: X-Y slice from particle displacement simulation of the Hill spherical vortex. Parameters $a = 1.5$ mm.

This approach to evaluating displacement allows curved streamlines to be generated. A consequence of this fact is that the flow field contains sharp velocity gradients which are known to degrade PIV correlation accuracy [3, p. 345]. This can be partially compensated by the use of window deformation techniques, which will be discussed in Chapter 6. Nevertheless, this is an excellent test for evaluating the utility of the technique for a physically realistic flow field. It should be noted that the previous discussion exposes a limitation of PIV in that it only evaluates linear displacements. This is *not* a limitation caused by the plenoptic camera.

Chapter 5

Light Field Rendering Algorithms

Generating volumetric intensity data from a simulated or experimentally acquired light field image requires the use of light field rendering algorithms. This chapter details the algorithms used, including registration of the light field data, computational refocusing, and intensity thresholding. More advanced reconstruction schemes using 3-D deconvolution are also discussed and will be the subject of future investigations.

The primary concept behind light field rendering is to select a subset of rays from the complete 4-D light field to generate a conventional 2-D image. The unique aspects of the light field, such as the complete parameterization of spatial and angular information, allows modifications to be made prior to the generation of the final 2-D image. Herein, the computational refocusing approach uses the angular information contained in the light field to propagate light rays to an image focal plane that is different from the original focal plane. This procedure is similar to the ray-tracing developed previously in the context of the image simulator. By integrating a subset of rays at this new location, a 2-D image at a new location is created. To enable this process, the light field image must be parameterized correctly. This is addressed in the first topic of this chapter, which is then followed by a discussion of refocusing and rendering.

5.1 Light Field Registration

The parameterization of the 4-D light field uses two spatial coordinates (x, y), and two angular coordinates (θ, ϕ). The process of registration refers to assigning specific values of these four coordinates to each pixel of a light field image acquired with the plenoptic camera.

With the light field parameterized in this manner, it is possible to perform ray-tracing to synthetically refocus the camera to an arbitrary image focal plane.

The registration first begins by either simulating or acquiring a "pinhole" image to determine the central locations of each lenslet. To create a simulated pinhole image, a large number of points within the focus plane are defined ($z_{span} = 0$), and a large f-number is used ($(f/\#)_m = 16$) to generate distinct points on the sensor plane. The diffraction parameters are recalculated, allowing the diffraction spot size to be representative of a $(f/\#)_m = 16$ lens. Figure 5.1 is a representative pinhole image rendered using the simulator. The variation in spot intensities is due to the random distribution of particles used to form the image. In other words, the simulator models light as discrete rays emanating from point sources and an approximation to a uniform field is created by using a large number of particles placed on a single plane. Due to this, there can be inhomogeneities in the resulting image. This has been compensated to an extent by using a large number (500,000) particles with a small number of rays (100) emanating from each. Actual pinhole images will not be subject to this effect.

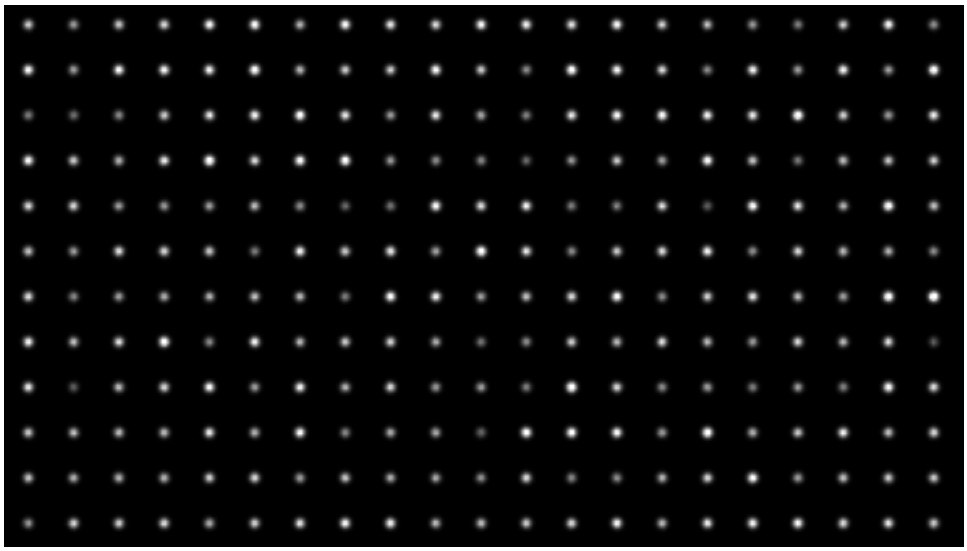


Figure 5.1: Cropped portion of a simulated pinhole image with $(f/\#)_m = 16$.

A Cartesian grid is placed over the image which defines the search regions for each lenslet image. This is performed in MATLAB by specifying mesh node points using the

function `meshgrid`, which is manually offset as needed to position the grid appropriately. The parameters of the grid are set through visual inspection of the resulting overlaid mesh to ensure each grid cell contains only a single lenslet image. The finite spot size created by diffraction provides an excellent way to resolve the lenslet locations to subpixel accuracy by using a 2-D Gaussian peak fit on the spot formed under each lenslet. This is performed in MATLAB by first determining the intensity centroid of each grid cell and a first-order estimate of the standard deviations in both coordinate directions. These are used as a seed for the MATLAB function `fminsearch`, which operates on the 2-D Gaussian function given in equation 5.1.

$$f(x) = ce^{-\left(\frac{(x-c_x)^2}{2\sigma_x^2}+\frac{(y-c_y)^2}{2\sigma_y^2}\right)} \tag{5.1}$$

After a number of iterations, the function returns updated estimates of the peak locations. This procedure is relatively quick; for an iteration tolerance of 1×10^{-7}, each grid cell takes approximately 0.03 seconds to evaluate, and the centers of all lenslet images could be found in about 10 minutes. Note that this is a one-time procedure for a particular camera configuration. The lenslet center locations are saved to a file and used on all subsequent images that use identical camera settings. An example of a small portion of a processed image is shown in Figure 5.2, where the grid is shown and the centroid values determined from the Gaussian fit. Upon visual inspection, excellent agreement is seen with the center of the lenslet images.

The next step in the registration procedure is to convert the lenslet centroid positions c_x and c_y from image coordinates back to the optical axis coordinates (l_x and l_y) using equations 5.2 and 5.3. Also, the angle between the optical axis at the aperture and the lenslet is determined, denoted l_θ and l_ϕ. These are termed the "bulk angles" in this work, and are calculated with equations 5.4 and 5.5 using the `atan2` function.

$$l_x = c_x p_p - p_p n_{p,x}/2 \tag{5.2}$$

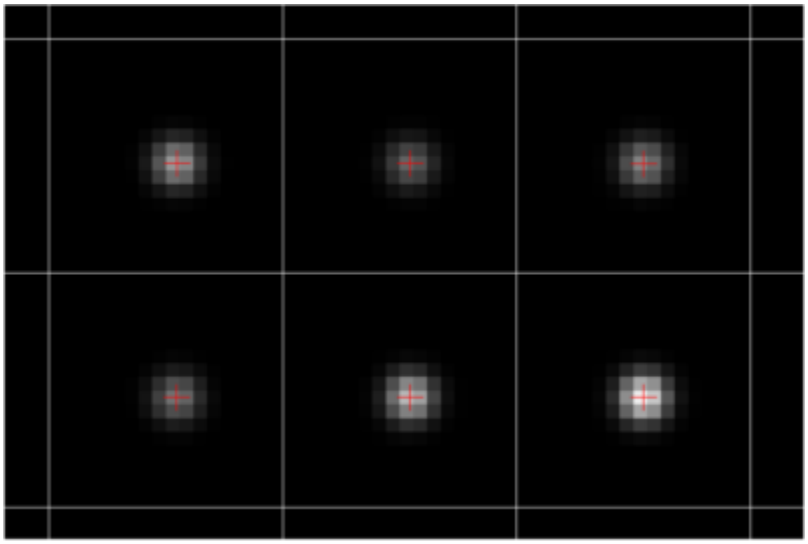


Figure 5.2: Cropped portion of processed image showing grid cells and center locations.

$$l_y = -c_y p_p + p_p n_{p,y}/2 \tag{5.3}$$

$$l_\theta = \tan^{-1} l_x/s_i \tag{5.4}$$

$$l_\phi = \tan^{-1} l_y/s_i \tag{5.5}$$

A basic working principle of the plenoptic camera is that the spatial resolution is sampled at the lenslet location, and thus all pixels corresponding to a particular lenslet are set to the same spatial coordinate. Determining the individual pixels that are paired with a lenslet is found by using a distance formula in both coordinate directions. Since the location of each lenslet is known, the average distance and radius can be calculated. This is used to set the bounds on the lenslet size used in the distance calculation. Once the proper pixels are identified, the x and y coordinates for each of the pixels are set to the l_x and l_y coordinates of the lenslet. Next, the angles for each pixel p_θ and p_ϕ are calculated through the use of equations 5.6 through 5.9, where f_l is the lenslet focal length and x_l and y_l are local spatial coordinates for the pixel.

$$x_l = x_p p_p - \frac{n_{p,x}}{2} p_p - l_x \tag{5.6}$$

$$y_l = y_p p_p - \frac{n_{p,y}}{2} p_p - l_y \tag{5.7}$$

$$p_\theta = \tan^{-1} x_l / f_l \tag{5.8}$$

$$p_\phi = \tan^{-1} y_l / f_l \tag{5.9}$$

This process is repeated for each lenslet in the image, such that each pixel has a spatial position defined by the nearest lenslet and an angular position defined by the relative location with respect to the lenslet centroid. An example registration is shown in Figure 5.3, with both angular coordinates listed. This registration procedure is more general than the approach taken by Ng et al. [36], which used image rotation, resizing, and cropping to ensure that an integer number of pixels defined each lenslet. Their approach is simpler to implement and allows computationally efficient algorithms to be used such as Fourier slice refocusing [50]. However, both rotation and resizing involve bilinear interpolation procedures which have been shown to degrade traditional particle imaging [3, p. 394]. The technique given here requires no interpolation in the registration process.

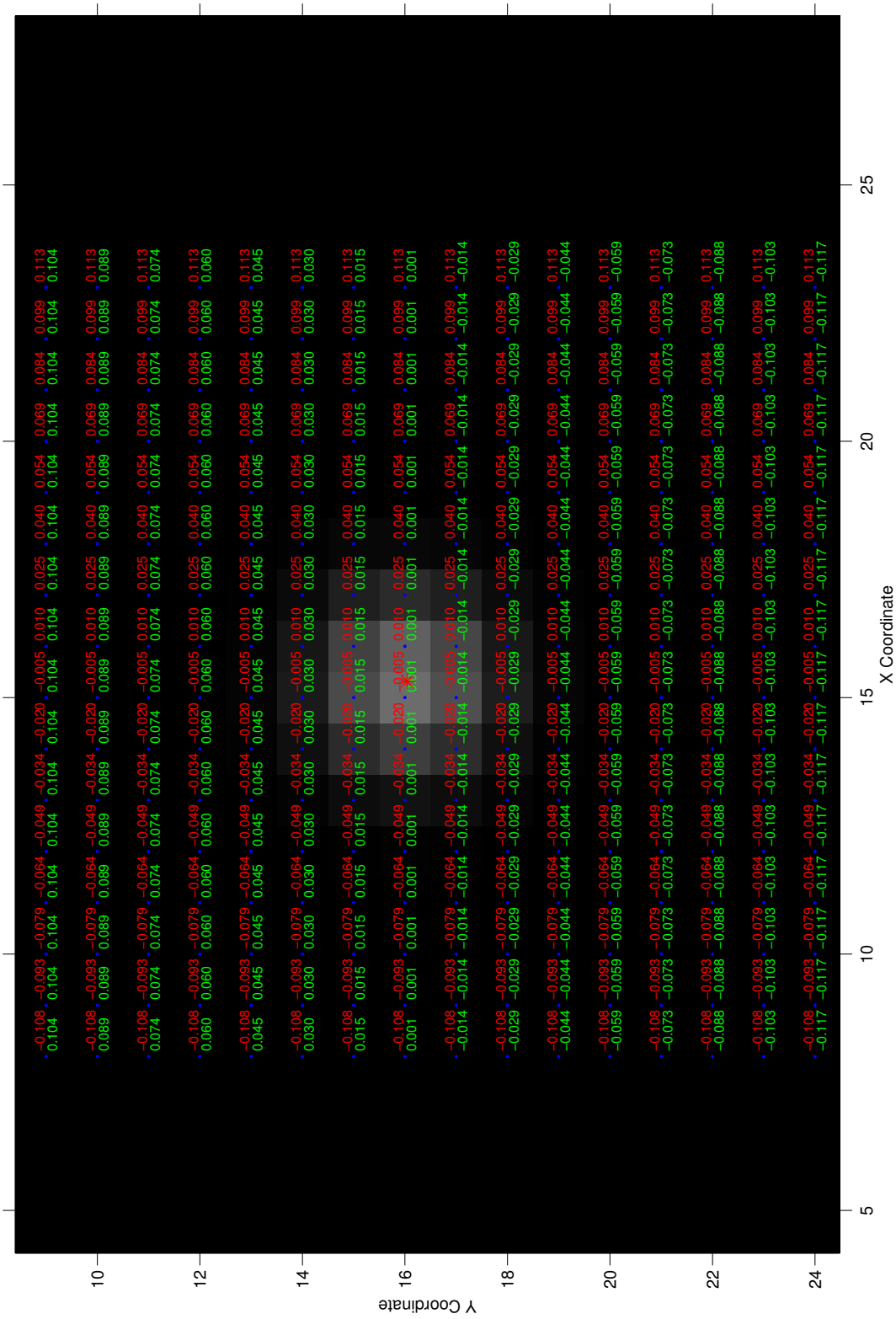


Figure 5.3: Registration for a single lenslet. Top number (red) indicates the angle θ. Bottom number (green) indicates the angle ϕ. Actual pixel locations given by blue points. Lenslet centroid shown as red asterisk.

5.2 Computational Refocusing

Rendering a single pixel of a computationally rendered image is a straighforward process of integrating the signal from all pixels underneath a particular lenslet. By performing this operation for each lenslet, a large number of pixels are generated, corresponding to the final computationally rendered image. The computed image will have a depth of field given by the effective f-number of the system and the size of the lenslets. Note that other rendering techniques exist which use a subset of the pixels under each lenslet. This is known as synthetic aperture refocusing, and allows the size and position of the aperture to be adjusted after the image is captured to simulate various depths of field and perspective changes. This capability is not used for the algorithms discussed here.

Computational refocusing is a simple extension of light field rendering, which uses the angular information defined at each pixel to effectively propagate the intensity to a new plane. This is similar to the ray-tracing defined in the previous chapter, and actually uses equation 4.2 for the propagation. The lateral coordinate of the ray at this new plane can be located within the domain of a different lenslet, causing the resulting rendered image to have a difference in intensity. The algorithm first begins by defining the size of the synthetic sensor. This differs from the actual sensor size, as indicated in Figure 5.4. The synthetic size $synth_x$ and $synth_y$ is simply defined by the size of the usable lenslets on the sensor. In the figure, x_{max}, x_{min}, y_{max}, and y_{min} are the maximum and minimum lenslet centroid locations, and avg_x and avg_y are the average size of the individual lenslets determined from the registration procedure.

This definition of the synthetic sensor size is valid when the refocus plane is located at the original focus plane (i.e, $t = 0$). However, when the refocus plane is translated to a different distance, the synthetic sensor size must be modified to take into account the trapezoidal shape of the imaging volume. This is illustrated in Figure 5.5, where the sizes of the pixels and overall sensor can be seen to increase as the distance is increased from the lens. This scaling can be accounted for through the use of similar triangles, giving a scale

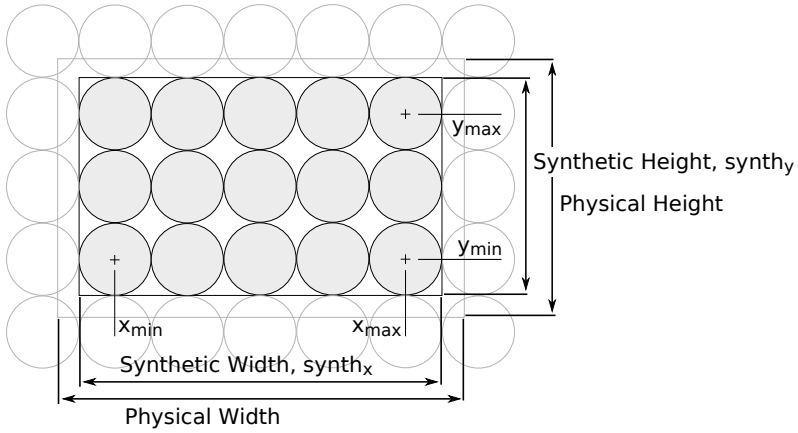


Figure 5.4: Differences in synthetic sensor size vs. actual sensor size.

factor $SF = (t + s_i)/s_i$. Using this scale factor, the synthetic sensor size and synthetic pixel size can be calculated using equations 5.10 through 5.13.

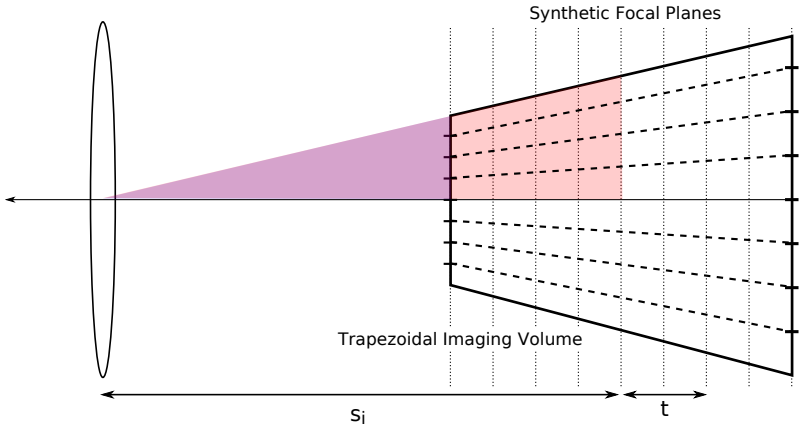


Figure 5.5: Change in synthetic sensor size with refocus distance.

$$avg_x = SF \times \left(\frac{x_{max} - x_{min}}{n_x - 1} \right) \tag{5.10}$$

$$avg_y = SF \times \left(\frac{y_{max} - y_{min}}{n_y - 1} \right) \tag{5.11}$$

$$synth_x = SF \times (x_{max} - x_{min}) + avg_x \tag{5.12}$$

$$synth_y = SF \times (y_{max} - y_{min}) + avg_y \tag{5.13}$$

With the size of the synthetic sensor defined at a refocus distance, each ray of the light field can be rendered. First, it must be determined where the ray intersects the synthetic focal plane in terms of the lenslet number. This is done through equations 5.14 and 5.15. Note that these equations will return a fractional value.

$$n_{s,x} = (synth_x/2 + avg_x/2 + x)/avg_x \tag{5.14}$$

$$n_{s,y} = (synth_y/2 + avg_y/2 - y)/avg_y \tag{5.15}$$

The fractional value returned by these equations allows for a sub-pixel interpolation of the intensity in the rendered images. Consider the diagram shown in Figure 5.6. If the ray of the light field is considered to be discretized into pixel-sized elements, and the particular ray falls on a non-integer lenslet number, the signal from the particular ray will be spread over multiple elements. Calculating the contribution of the signal to the overlapped pixels is done using the normalized coefficients TL, TR, BL, and BR. This is schematically outlined in Figure 5.6a, and the resulting distribution of signal is given in Figure 5.6b. Note that the intensity levels are inverted in this figure, and a darker gray value indicates greater intensity. The code for performing this operation is not listed here, but is provided in the appendix.

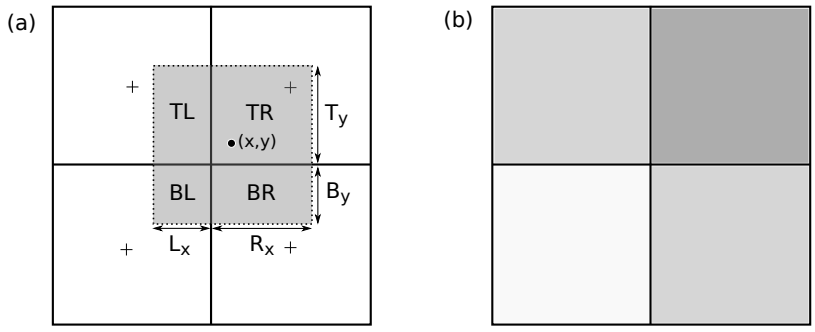


Figure 5.6: Signal interpolation technique, a) Definition of normalized coefficients, b) Resulting rendered pixels.

A set of four low-particle-density, half-size refocused images is generated and shown in Figure 5.7. In this case, arbitrary focal plane spacing was chosen as a demonstration. In Figure 5.7a, the image is rendered with no refocusing performed. Clear points can be

identified corresponding to imaged particles. Also notable are collections of low-intensity blur spots throughout the image. As shown in figures 5.7b-e, bright points from one image appear as blur spots in the remaining images. From a casual inspection, it appears that each plane is unique in that the pattern of high-intensity, in-focus particles is unique in each image.

These images provides insight on two important issues. First, it is clear that the choice of focal plane locations is key for ensuring the uniqueness of each refocused image. This will be the topic of the next section. The second finding is the clarity by which in-focus particles can be identified amidst a background of blurred particles. This contrast in signal values may allow simple thresholding to be used for volume reconstruction, a topic which will also be discussed in the following sections.

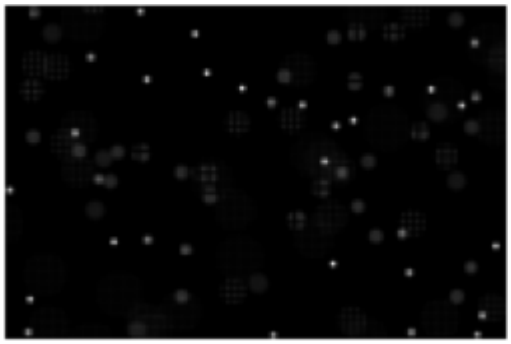
5.3 Focal Plane Spacing

The depth-of-field concepts developed earlier in Section 4.2.2 provide a starting point for determining the focal plane locations. The basic premise is that each refocused image will contain unique information if the focal depths of each refocused image *do not overlap*. This is schematically shown in Figure 5.8. Here, multiple light cones are emanating from the object space, forming multiple diamonds in the image space. As previously stated, the height of these diamonds is equal to the circle of confusion, c. In this analysis, c is taken to be the calculated lenslet pitch.

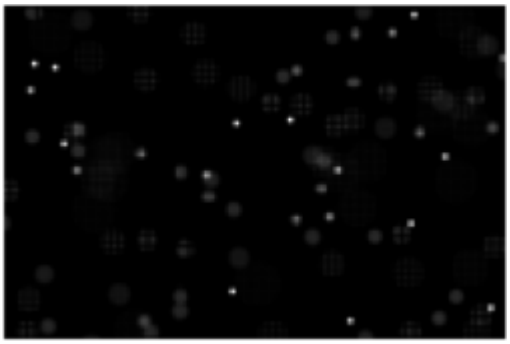
From an overview of this figure, it is apparent that unique information will only be generated when refocus planes are chosen that lie within separate diamonds (i.e., two planes should not fall within the same diamond; these would contain redundant information). For this work, the focal plane is placed at the center of this diamond.

To determine the actual locations, a marching approach is required that first solves for the original focus plane, then steps to the remaining planes. Since this discussion is focused on the image plane rather than the object plane, the earlier expressions for depth of field are

(a) $t = 0$ mm



(b) $t = -2$ mm



(c) $t = +2$ mm

(d) $t = -4$ mm

(e) $t = +4$ mm

Figure 5.7: Example of refocused images at various refocusing planes.

not used. Instead, the term *depth of focus* is used to describe the equations dealing with the image plane. For determining the initial bounds $D_{N,i}$ and $D_{F,i}$ of the original focus plane, equations 5.16 and 5.17 are used.

$$D_{N,i} = \frac{fD_i}{f - (f/\#)p_l} \tag{5.16}$$

$$D_{F,i} = \frac{fD_i}{f + (f/\#)p_l} \tag{5.17}$$

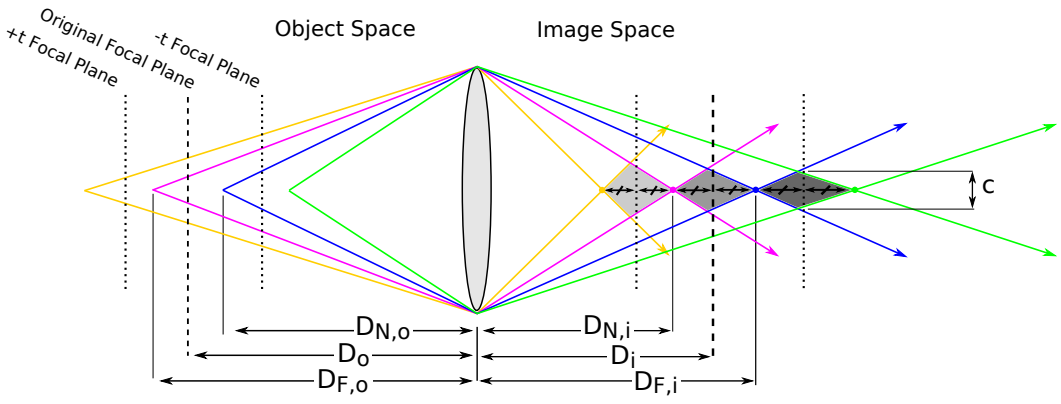


Figure 5.8: Unique depths of field for calculation of the focal plane spacing.

By rearranging the equations, they can be rewritten to remove the term corresponding to the distance D_i. The resulting equations 5.18 and 5.19 are much better suited for a marching approach in both directions from the original image focal plane.

$$D_{N,i} = \frac{fD_{F,i}((f/\#)c/f + 1)}{f - (f/\#)c} \tag{5.18}$$

$$D_{F,i} = \frac{fD_{N,i}(1 - (f/\#)c/f)}{f + (f/\#)c} \tag{5.19}$$

$$D_i = D_{N,i}(1 - (f/\#)c/f) \tag{5.20}$$

These equations have been applied with the reference imaging conditions given in table 4.3. As suggested by Levoy [37], the number of unique focal planes capable of being generated by a plenoptic camera is equal to the number of pixels underneath each lenslet. For the camera considered here, this is taken to be 16. Thus, a total of 8 planes are found on either side of the original image focal plane. The relevant values of the calculation are shown in table 5.1. The second and third columns are the bounds on the depth of focus for each plane. Column 4 shows the size of this depth of field; note, the size of the depth of field changes depending on which direction the image is refocused. The fifth column takes these values and divides it by the circle of confusion c (in this case, lenslet pitch p_l). This shows, as before, that the ambiguity in depth is almost an order of magnitude larger than in-plane

ambiguity. The location of each of the refocus planes is given in the sixth column, and finally, the distance the light field must be propagated t is given in the last column.

Table 5.1: Synthetic Depth of Field and Focal Plane Positioning

Plane No.	$D_{N,i}$ (mm)	$D_{F,i}$ (mm)	$D_{F,i} - D_{N,i}$	$(D_{F,i} - D_{N,i})/c$	D_i (mm)	t
1	91.8523	92.7755	0.9231	7.3851	92.3116	-7.6884
2	92.7755	93.7079	0.9324	7.4593	93.2393	-6.7607
*3	93.7079	94.6497	0.9418	7.5343	94.1764	-5.8236
4	94.6497	95.6009	0.9513	7.6100	95.1229	-4.8771
5	95.6009	96.5617	0.9608	7.6865	96.0789	-3.9211
*6	96.5617	97.5322	0.9705	7.7638	97.0445	-2.9555
7	97.5322	98.5124	0.9802	7.8418	98.0199	-1.9801
8	98.5124	99.5025	0.9901	7.9206	99.0050	-0.9950
*9	99.5025	100.5025	1.0000	8.0002	100.0000	0.0000
10	100.5025	101.5126	1.0101	8.0806	101.0050	1.0050
11	101.5126	102.5328	1.0202	8.1618	102.0202	2.0202
*12	102.5328	103.5633	1.0305	8.2438	103.0455	3.0455
13	103.5633	104.6041	1.0408	8.3267	104.0811	4.0811
14	104.6041	105.6554	1.0513	8.4104	105.1272	5.1272
*15	105.6554	106.7173	1.0619	8.4949	106.1837	6.1837
16	106.7173	107.7898	1.0725	8.5803	107.2509	7.2509
17	107.7898	108.8731	1.0833	8.6663	108.3288	8.3288

Table 5.1 provides the actual values which will be used herein for all refocused images unless otherwise stated. Another, possibly easier, way to visualize this data is with a simple plot. Figure 5.9 shows each of these points, and the corresponding depth-of-field "diamonds" created by each. In this figure, the cross markers represent the refocus plane positions, and the vertical line at 100 mm indicates the original image focal plane position. Note the unequal axes in this figure, which are used for ease of visualization. Also, the optical axis in this figure does not correspond to the optical axis in the actual simulators; here, the optical axis is taken to be the distance from the lens s_i.

As an aside, it is worth exploring the differences focal length and f-number play in determining unique focal planes. For a short focal-length lens, having a large f-number can cause drastic nonlinearity in the sizing and spacing of the focal planes, as shown in Figure 5.10 for the case of a 50 mm lens operated at an f-number of 11. However, as focal length

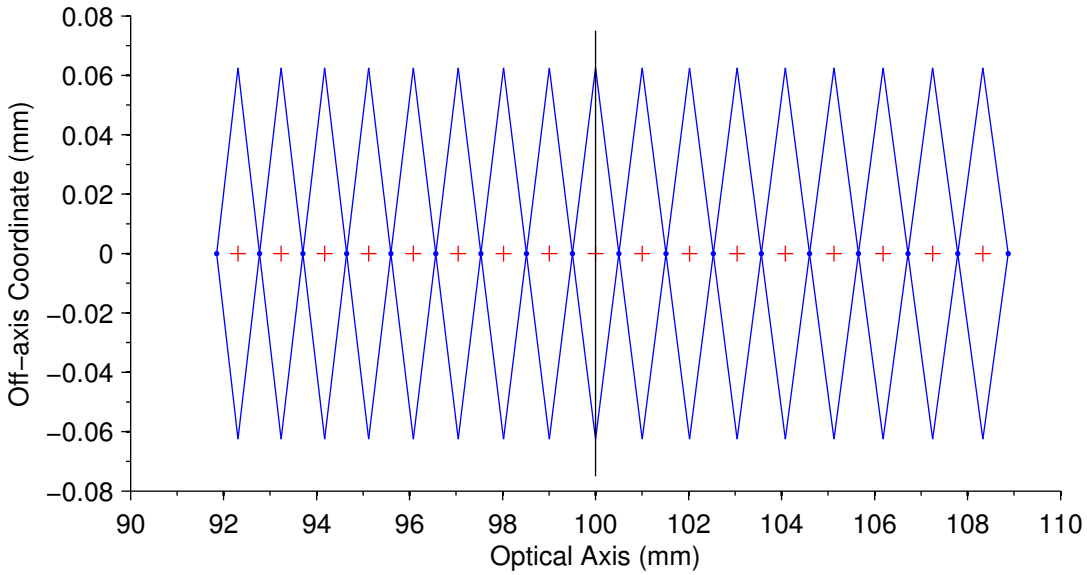


Figure 5.9: Unique depths of field for calculation of the focal plane spacing. $f/\# = 2$, $f = 50$ mm.

is increased, this effect becomes much less significant. Figure 5.11 shows a 200 mm focal-length lens operated at the same f-number. The primary reason for this is the distances from the lens; for the 50 mm/f-number 11 case, the limited angles of light allowed to enter the camera require a large physical distance in order to generate a unique focal plane spacing. For the 200 mm case, the distance required to generate the unique focal planes is the same; however, it is a smaller percentage of the total imaging distance and thus doesn't result in a substantial nonlinearity.

5.4 Single-particle refocusing

With the focal planes defined, a more detailed look can be used to evaluate the rendered images. First, the case of refocusing a single particle is considered. In the following plots, a cropped 2-D image is shown as well as a 1-D slice of pixel intensity values through the center of the image. Figures 5.12 through 5.16 present these various views. In each case, clear changes in image intensity and diameter can be seen. One of the best ways to evaluate changes in shape is by defining a particle image diameter, which can be found by performing

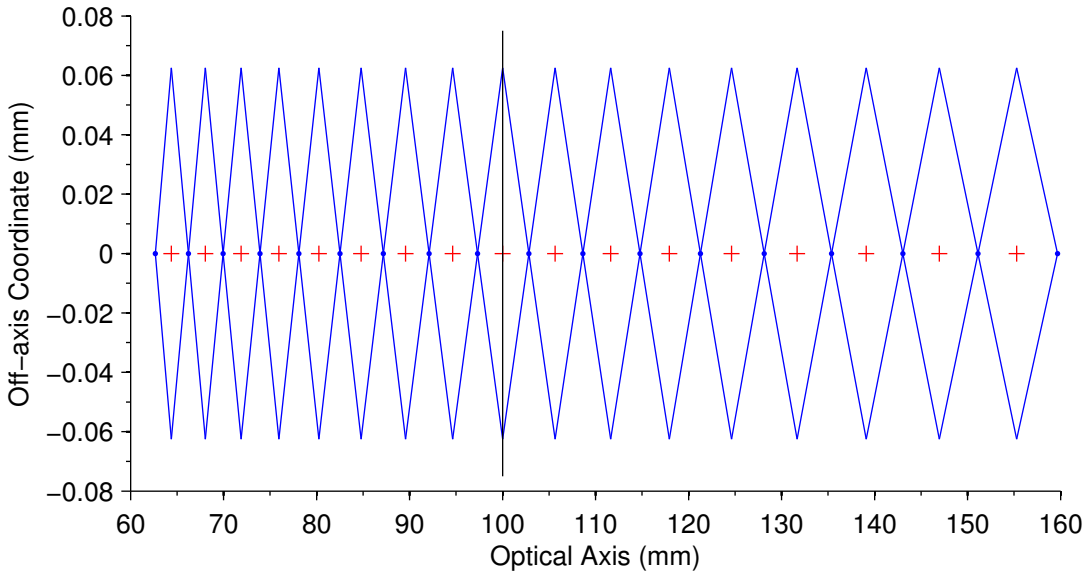


Figure 5.10: Unique depths of field for calculation of the focal plane spacing. $f/\# = 11$, $f = 50$ mm.

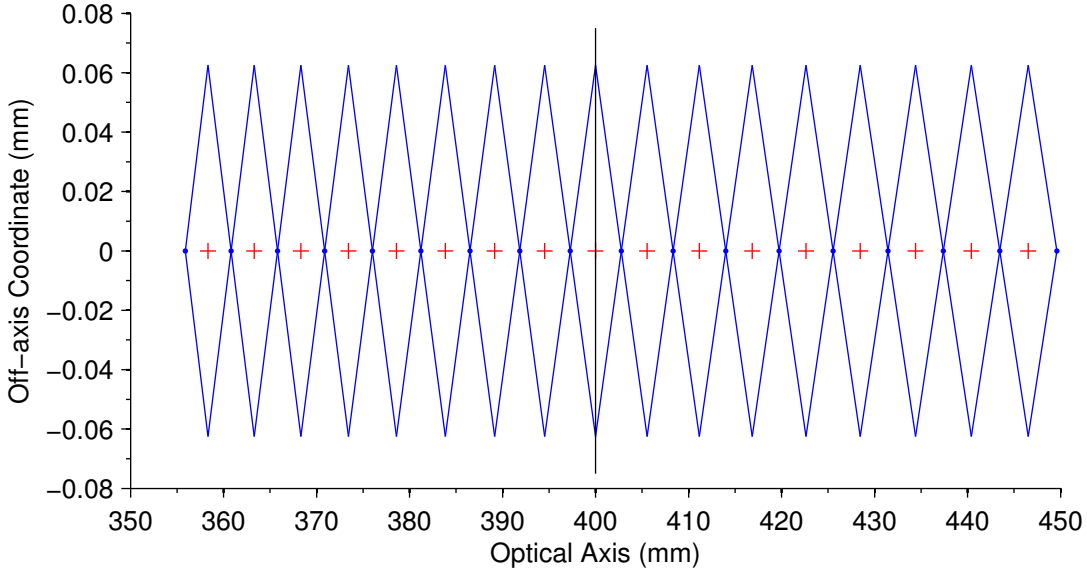


Figure 5.11: Unique depths of field for calculation of the focal plane spacing. $f/\# = 11$, $f = 200$ mm.

a Gaussian fit to the data. By using the previously derived equation 4.25 and rearranging to solve for d_s (the spot diameter defined in Adrian and Westerweel [3, p. 101]), the particle image diameter in pixels can be determined. This is a critical parameter for PIV evaluation; the majority of PIV algorithms perform best when the particle image diameter is greater

than 1.5 pixels. In the figures shown here, this does not appear to be an issue. Note that in Figure 5.14 the Gaussian function is slightly skewed; there actually is a small value of image intensity which is not easily visible because the lenslet is not exactly positioned on the optical axis. Regardless, this shows that even for an approximately in-focus image, the particle diameter is suitable for PIV. Another interesting feature of this data is the rapid change in intensity values. The in-focus particle in Figure 5.12 exhibits a peak intensity of around 30,000, whereas the out-of-focus particle in Figure 5.14 has a peak intensity much lower, roughly 800. This suggests that out-of-focus particles could potentially be removed from images using a simple thresholding process. This will be the topic of the next section.

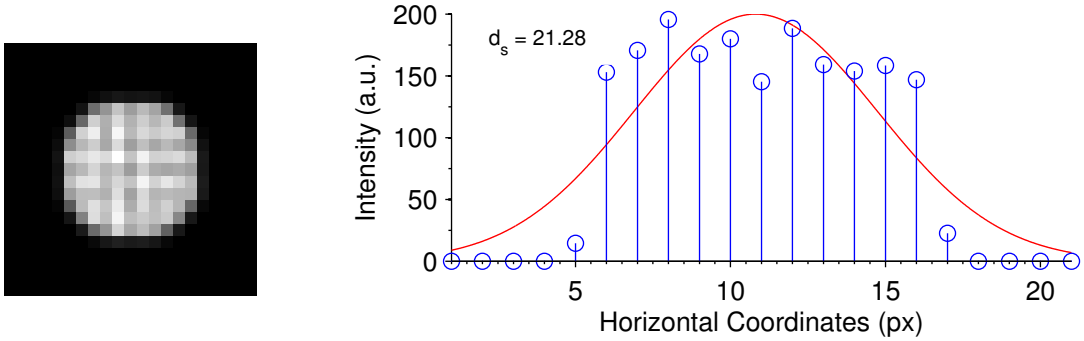


Figure 5.12: Refocus plane 3, $t = -5.8236$ mm

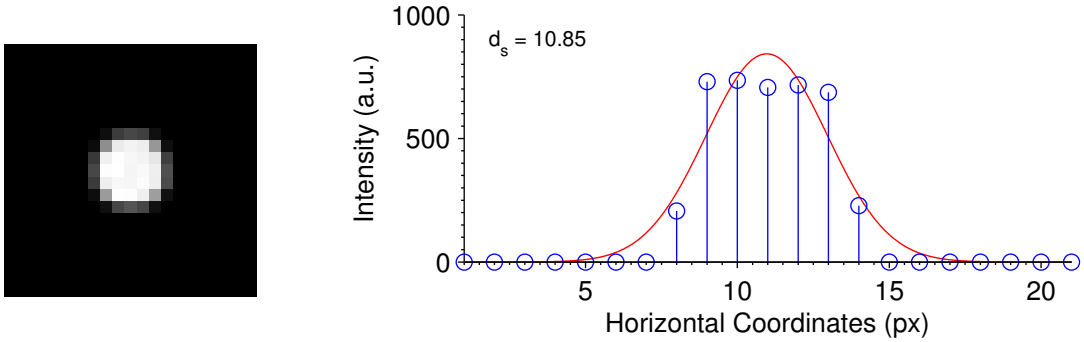


Figure 5.13: Refocus plane 6, $t = -2.9555$ mm

The focal plane spacing derived earlier is the minimum spacing required in order for a particle located at any position within the volume to be focused on a plane in the refocused images. However, to ensure that the resulting volume of the focal stack is scaled correctly

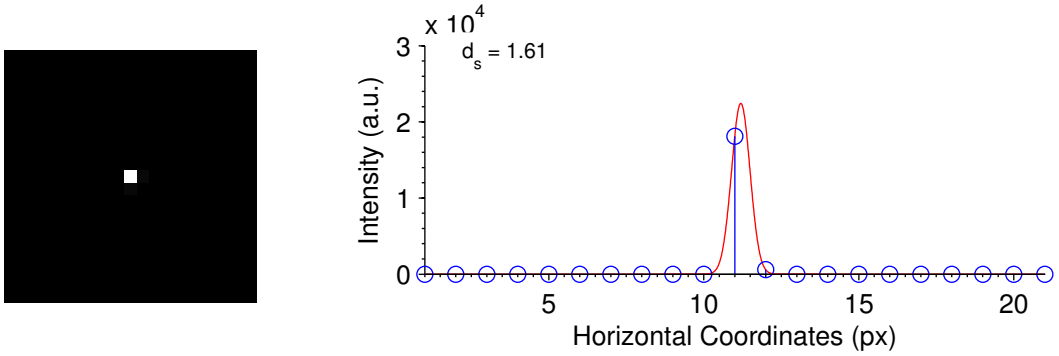


Figure 5.14: Refocus plane 9, $t = 0.0000$ mm

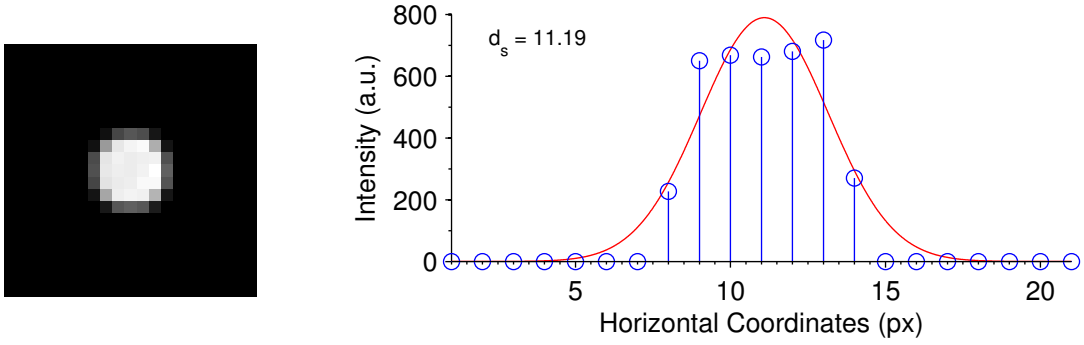


Figure 5.15: Refocus plane 12, $t = +3.0455$ mm

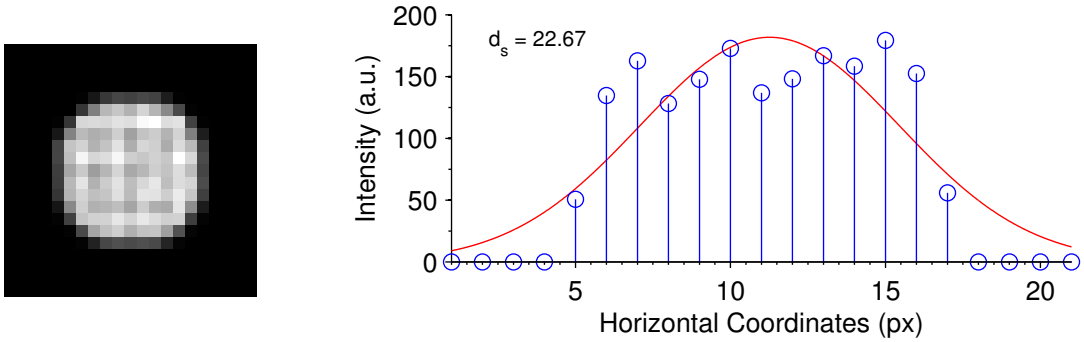


Figure 5.16: Refocus plane 15, $t = +6.1837$ mm

in all three dimensions, the refocus planes are typically supersampled. For example, in the current configuration, each pixel of a 2-D rendered slice represents an area of 125 μm^2, defined by the area of the lenslets. For the reconstructions to be properly scaled, the refocus plane spacing (depth-dimension) must be equal to the image or in-plane dimension.

5.5 Intensity Thresholding

The refocused planes contain multiple contributions from out-of-focus particles. These contributions would distort the correlation in PIV algorithms if not removed from the 3-D data. For this reason, an approach used by Belden et al. [32] for synthetic aperture PIV data is adapted for this work. In their paper, they considered refocused images to be composed of high-intensity in-focus particle images and low-intensity out-of-focus noise, both of which are randomly distributed in space. They found that the intensity distribution of a refocused image was approximately Gaussian, allowing the convenient parameters of mean μ and standard deviation σ to be used for quantifying the intensity range. The refocused images generated in this work do not appear to have the same Gaussian intensity distribution, but can be described in general terms using one; for example, Figure 5.17 shows an intensity histogram of a sample refocused image with a Gaussian fit overlaying the histogram. While the Gaussian curve doesn't closely follow the data, it does describe the general trends well. Also included on this graph are the locations of the mean intensity value, and the value located three standard deviations above the mean. In this work, this threshold level is adopted.

A modification made to the technique of Belden et al. is to dilate the thresholded images by a radius of 1 pixel. This serves the primary purpose of ensuring that each particle image contains neighboring information and has an effective diameter of greater than 1 pixel. The reason for this change is that the primary sub-pixel evaluation routines in PIV processing algorithms rely on either a three-point Gaussian or polynomial fit to the correlation plane. The shape and information of the correlation peak in PIV algorithms is dictated by the form of the particle images; thus, including neighboring information in the particle images ensures that the correlation plane will also contain multiple pixels of information available to sub-pixel estimators. Also, the artifact known as "peak locking," or integer biasing in PIV vector fields, is primarily driven by particle image sizes being less than roughly 2 pixels.

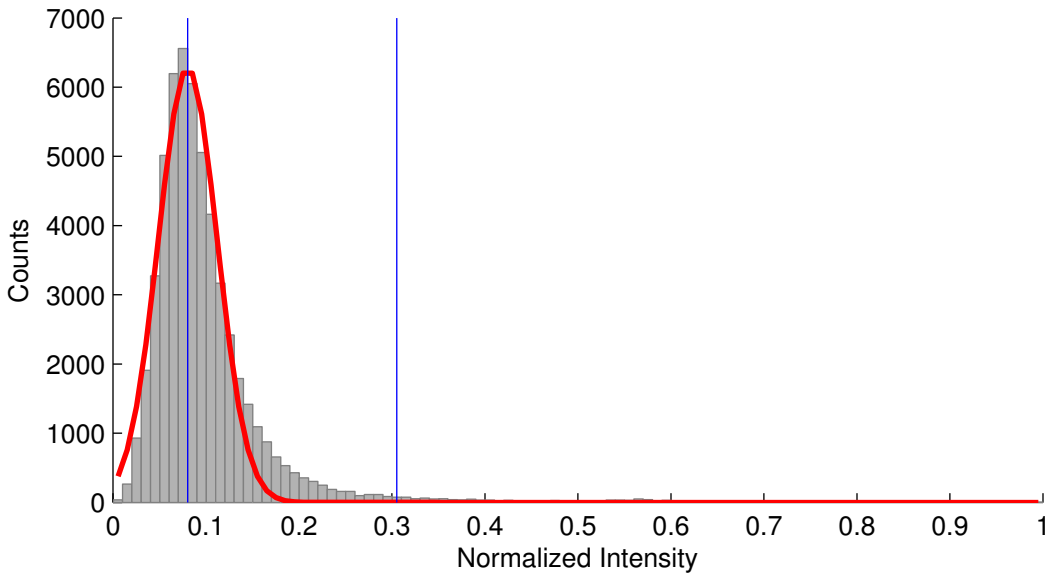


Figure 5.17: Intensity histogram for a refocused image using 100 bins. Gaussian distribution fitted to the distribution, and markers of the mean intensity, and three standard deviations above the mean intensity given.

To perform the dilation, the thresholded image is converted into a binary image. MATLAB has a number of convenient routines for modifying binary images. In particular, the `strel` function is used to produce the correct dilation filter, which is then passed to `imdilate`. The resulting dilated binary image is then multiplied by the original image, to yield a final thresholded image. The results of this operation are shown in figures 5.18 through 5.20. The former is an original refocused image, and the latter represents the image after thresholding and dilation. Clearly, the thresholding removes the low-intensity background due to out-of-focus particles and retains the in-focus particle images. Such an image is suitable as a preliminary input for correlation algorithms.

5.6 3-D Deconvolution and Limited-Angle Tomography

The thresholding procedure described above is used to generate inputs to the correlation algorithms. This procedure is simple but is not well-suited for dense particle fields where particles may not be easily identified. Another disadvantage with the thresholding approach is that only information from a single refocused plane is used to make the thresholding

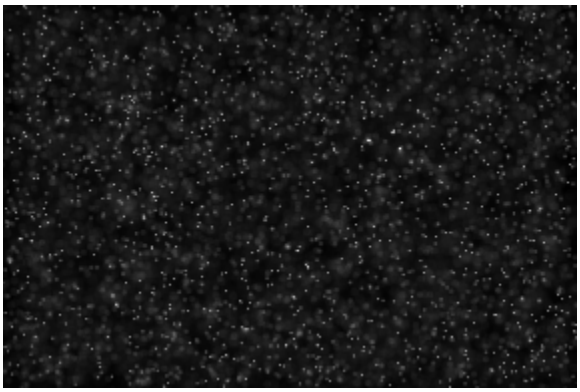


Figure 5.18: Original light field image without thresholding.

Figure 5.19: Light field image with non-dilated thresholding.

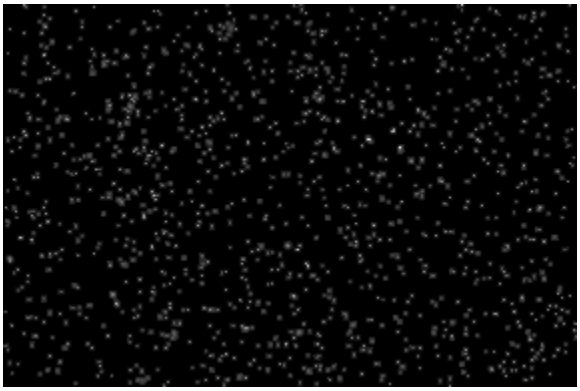


Figure 5.20: Light field image with dilated thresholding.

calculation (as opposed to using information contained in all planes to make an accurate determination). For this reason, the techniques of 3-D deconvolution and limited-angle

tomography can be used, which compute the 3-D volume by using information from more than one plane simultaneously, and could potentially give more accurate results.

3-D deconvolution was pioneered in the early 1980s as the mathematical foundation to deconvolution microscopy, and has advanced over the past twenty years to include multiple deconvolution algorithms and implementations. It was developed as a method to increase the spatial resolution in microscopes. Resolving small structures within biological cells is not typically possible using traditional microscopy due to the fundamental limiting effects of diffraction for both imaging and illumination. Some of these issues have been addressed using clever illumination techniques and fluorescent markers; however even with these methods, out-of-focus scatter from other regions of the specimen reduce the effective resolution of the in-focus images. 3-D deconvolution uses information on the blurring behavior of the microscope objective to reduce out-of-focus scatter in in-focus images.

An introduction and review of deconvolution techniques is given by Sibarita [51] and McNally et al. [52], with a short summary given here. The first step in a deconvolution process is either the calculation or measurement of the 3-D point spread function (PSF) of the optical system. In microscopes, this is a complex function due to the effects of diffraction. Typically this is performed by imaging tiny fluorescent beads at different focal depths. In traditional photography, the PSF can be approximated much more simply since the effects of diffraction are negligible, but it can be determined in a similar fashion by measuring the manner in which an approximately point light source becomes blurred. For light field photography, the PSF can be calculated in exactly the same way, except rather than using manual refocusing, the image of a point source can be taken once and computationally refocused. However, there is a complication of the PSF for traditional and light field photography which involves the radial asymmetry of the PSF. Because a traditional camera acquires perspective views of a scene, the angles of light incident on the sensor vary. This is in contrast to microscopy, where all views are effectively orthographic.

Once the PSF is known, a variety of computational algorithms can be used. Each of these algorithms uses the knowledge of the PSF and the data contained in the focal stack to remove out-of-focus contributions. In a basic sense, it can be regarded as an inversion method. The simplest algorithm used is the nearest neighbors algorithm, which restricts itself to using neighboring planes of data rather than the entire focal stack. The justification for this is that the nearest two planes make the most substantial contribution to the in-focus plane. This algorithm is very fast to apply and does not require iteration. Constrained iterative algorithms such as the Jansson Van-Cittert algorithm and the Gold algorithm can generate good results; however, they are typically slow and very sensitive to measurement noise. This may preclude them from being used in images with low signal-to-noise ratio. Another class of methods is known as statistical methods. One example is the maximum likelihood estimation (MLE) algorithm, which uses the knowledge of the image noise sources (Poisson noise) to make an estimate of the resulting images. Another approach is to use a regularization function in addition to the MLE algorithm. This makes an assumption that the noise is Gaussian to simplify the mathematical formulation, which is quite complex. The review by Sibarita [51] gives examples of each of these applied to microscope data. In this paper, the author mentions that no one algorithm is well suited for all cases; however, the maximum likelihood estimation algorithm and one-step algorithms not discussed here (Wiener filtering and unconstrained Tikhonov-Miller) perform poorly in terms of final image quality.

It should be mentioned at this point that deconvolution microscopy has been used as a method for performing 3-D micro-PTV measurements. Park and Kihm [53] used a microscope with a 40X objective lens to visualize particle motion in a 100 μm^2 square micro-channel. The technique was able to resolve particles within a 5.16 μm cubic volume. The basic processing methodology was to take the particle images and use the knowledge of the microscope PSF to determine the depth. The entire volume under consideration was intentionally defocused to ensure no ambiguity in depth location occurred. Their work was

able to obtain fairly accurate measurements to within of +/- 7% (limited partially due to Brownian motion). However, it is doubtful the methodology used in their work can be applied to this work primarily since it deals with microscopic imaging and does not concern the reconstruction of a 3-D intensity field, but rather the identification and matching of the PSF.

Chapter 6

Volumetric Correlation Algorithms

In order to extract the displacement information from a pair of 3-D particle fields, a statistical interrogation scheme is required. These schemes are typically known as PIV algorithms, which rely on taking two images acquired at different times and finding the proper displacement to yield the maximum match between the two images. Typically the images are discretized into hundreds or thousands of interrogation windows which allow spatial variations in the displacement field throughout the images. By multiplying the pixel displacement by the magnification of the imaging system and the pixel size, the physical displacement is determined. The development of these algorithms for 2-D image data began as early as the 1980s but was severely restricted due to the limited computing capabilities available. A history of the early efforts and developments is not given here; the reader is referred to the review by Adrian [54] for additional historical context. However, as computing technology evolved, increasingly sophisticated interrogation procedures were developed which allowed for improved evaluation of particle displacement. A variety of approaches have been used, most of which can be classified into one or more groups:

- single-pass correlation
- multiple-pass correlation with discrete window offset
- grid refinement schemes
- multiple-pass correlation with image deformation
- spatially adaptive correlation
- optical flow

The single pass correlation technique, described in [55], was one of the first algorithms to be developed and tested on digital PIV data. Since then, researchers have found that an iterative solution approach can be used to increase the dynamic velocity range (DVR) and dynamic spatial range (DSR). The DVR is defined as the ratio of the maximum to minimum resolvable velocity, and the DSR is defined as the ratio of the maximum to minimum resolvable length scale. Substantial improvements in DVR can be achieved using a multi-pass iterative interrogation with a window offset, which reduces the in-plane loss-of-pairs and achieves greater accuracy and range in velocity measurement [56, 57]. Furthermore, by using a grid refinement in addition to the multi-pass window offset method, each interrogation region can be reduced in size which results in an increase in DSR [57].

The accuracy of PIV interrogation was further advanced by the application of image deformation techniques. These techniques perform a traditional single-pass correlation to determine a velocity field, and deform the two images such that the deformation follows the velocity field [58, 59]. In this manner, the correlation becomes far more robust with respect to velocity gradients that have a spatial scale on the order of the interrogation window size. These techniques also allow the use of symmetric offsets on both images to achieve second-order accuracy, as first suggested by Wereley and Meinhart [60].

A variety of other schemes have been developed and are still under development. Spatially adaptive PIV eliminates the use of a Cartesian grid for performing the interrogation, and results in interrogation windows spaced and sized to the scales of the flow field under evaluation [61]. This technique does yield increases in both DVR and DSR; however, the complexity of the algorithms make applying the technique difficult. Additionally, since the interrogation grid is uniquely adapted for each image pair, it is difficult to compile turbulent statistics since measurements are located at different positions in each correlation. Typically measurements must be interpolated onto a uniform grid, which can have the detrimental effect of smoothing the data.

Optical flow presents another approach to measuring displacement, which uses concepts developed originally for computer vision research. Optical flow algorithms are fundamentally different from correlation-based PIV algorithms in that the interrogation is performed by determining the gradient of intensity between an image pair. The displacement vector is simply a vector aligned along the gradient [62]. The advantage to this technique is the substantial increase in data yield (each pixel of the image results in a velocity vector). However, a fundamental limitation exists due to the calculation of the gradient: particle displacements cannot in general be greater than the particle image diameter. This greatly limits the DVR of the technique. However, research is still being conducted in this field on better understanding the connection between optical flow and fluid motion to extend the range of velocities that can be measured. Note that the references given for each of the algorithms discussed above are by no means exhaustive, but give the most well known work related to each algorithm.

Based on the previous discussion, the algorithms that appear capable of achieving the highest DVR, DSR, and accuracy while still being relatively straightforward enough to implement are the multi-pass discrete offset methods with grid refinement. The results from the third international PIV challenge (collected and published by Stanislas et al. [63]) showed that image deformation techniques are recommended; however, particular attention must be paid to the validation procedures and low-pass filters used, which greatly increases the difficulty of accurately implementing the algorithm. Some other findings were a general agreement that interrogation window sizes smaller than 16 x 16 pixels should be almost universally avoided. Also, the application of optical flow algorithms in the same tests did not achieve similar performance as the advanced correlation-based algorithms.

These findings motivated the choice in this work to use a multi-grid, multi-pass discrete offset method. Herein, the technique will be referred to as WIDIM (window displacement iterative multigrid), an acronym first used by Scarano [57]. This chapter begins with a basic description of the technique and outlines the processing steps required. The algorithm is

developed first for the evaluation of 2-D PIV images and is then extended to work for 3-D data based on modifications to the 2-D algorithm.

6.1 WIDIM Algorithm Development

A schematic outline of the WIDIM algorithm is given in Figure 6.1 and can be described as consisting of two separate methods. First, a grid refinement algorithm is used (also referred to as multi-grid), where the interrogation window size is decreased as a means to increase the DSR of the result. The second is the multi-pass algorithm, where multiple passes are performed using the displacement data from a previous pass to offset the interrogation windows prior to the correlation. The primary advantage to this iterative approach is the ability to eliminate the in-plane loss of pairs which occurs with no prediction step.

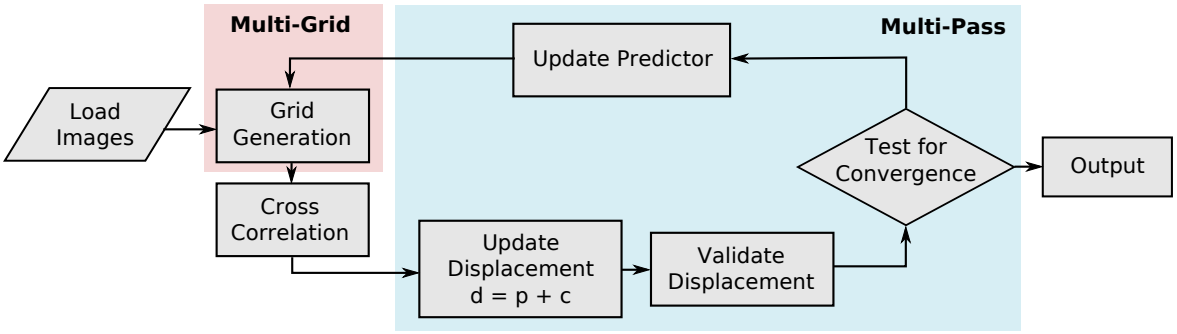


Figure 6.1: WIDIM algorithm flow chart.

6.1.1 Grid Generation

A Cartesian grid is generated that allows each interrogation window to fit within the image domain while also allowing for interrogation region overlap. The grid methodology is shown for a single interrogation row in Figure 6.2. The sizes of the interrogation windows are given as x_{size} and y_{size}, and the spacing between each window (which allows for overlap) is denoted as $x_{spacing}$ and $y_{spacing}$. Note that in Figure 6.2, the actual window sizes are distorted to show the window overlap, in this case 25%, more clearly. The number of windows that can fit within a given image size are determined by using a loop to fill the image until another

interrogation window cannot fit in the image area. This will result in a corner of the image not being covered by interrogation windows. For this reason, the grid is then offset by an amount x_{offset} and y_{offset} to center the grid within the image. The grid is defined as integer node points x_{node} and y_{node} and by the locations of the centroids x_{cent} and y_{cent} which can be fractional values.

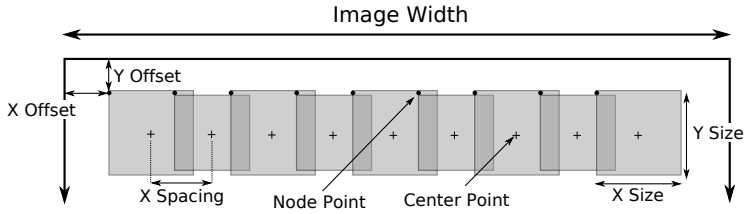


Figure 6.2: Schematic of PIV gridding.

6.1.2 Cross-Correlation

The interrogation of displacement is achieved using the discrete spatial cross-correlation R of two image windows, I_A and I_B, as defined in equation 6.1. The values of p and q are the shift of the interrogation regions with respect to each other.

$$R(p,q) = \frac{1}{x_{size} \times y_{size}} \sum_{m=1}^{x_{size}} \sum_{n=1}^{y_{size}} I_A(m,n) I_B(m+p, n+q) \tag{6.1}$$

The cross-correlation can be computed directly from this equation, however this is a computationally intensive operation requiring $(x_{size}y_{size})^4$ operations. An alternative is the use of an efficient Fast Fourier Transform (FFT) algorithm which reduces the complexity to $(x_{size}y_{size})^2 \log_2(x_{size}y_{size})^2$.

To perform the FFT-based spatial correlation, the image windows are obtained by using the previously defined grid node points and window sizes to extract a subset of data from the individual images. Each window is subtracted by the mean value of itself to eliminate the correlation of the mean and fluctuating intensities as shown in [3, p. 320]. This subtraction procedure is valid for the case of relatively homogeneous illumination within

each interrogation window, which is almost always a valid assumption. An example of two windows is shown in Figure 6.3a,b. Next, the two-dimensional FFT of both windows, F_A and F_B, is computed using the `fft2` function in MATLAB. The shifted absolute magnitudes of these Fourier transforms are shown in Figure 6.3c,d. The resulting pair of Fourier transforms are then multiplied together. Specifically, the complex conjugate of F_A is multiplied by F_B. This quantity is known as the cross-spectral density and is shown in Figure 6.3e. Taking the inverse FFT using the MATLAB function `ifft2` yields the spatial correlation R. Some implementations of the FFT, such as the FFTW library used by MATLAB, shuffle the FFT such that high frequencies are placed at the center, and the DC-component is located at the corners. For this reason, the function `fftshift` is used to reshuffle the values of the correlation plane so the center of the plane corresponds to zero displacement. The shifted correlation plane is shown in Figure 6.3f.

The drawback of using FFTs instead of a direct computation approach is that the FFT is defined only for periodic signals. As detailed in Adrian and Westerweel [3, p. 372], the exact value of the spatial correlation can only be recovered by zero-padding the input values such that the resulting size of the inputs I_A and I_B are $\geq 2x_{size}$ and $\geq 2y_{size}$. In practice, selection of window sizes in accordance with the one-quarter rule (displacements should be less than one-quarter of the interrogation window size) reduces this effect because the correlation peak will not be "folded" back into the actual correlation plane through the periodicity effect. However, to ensure correlation artifacts do not skew the results of this work, all spatial correlations are zero-padded by twice the input size in both coordinate directions. Note that even with zero-padding, the computational efficency of the FFT method is still an order of magnitude better than the direct method. For example, using 64 x 64 px interrogation windows, the direct approach requires almost twenty million operations, while the zero-padded FFT requires just 200,000.

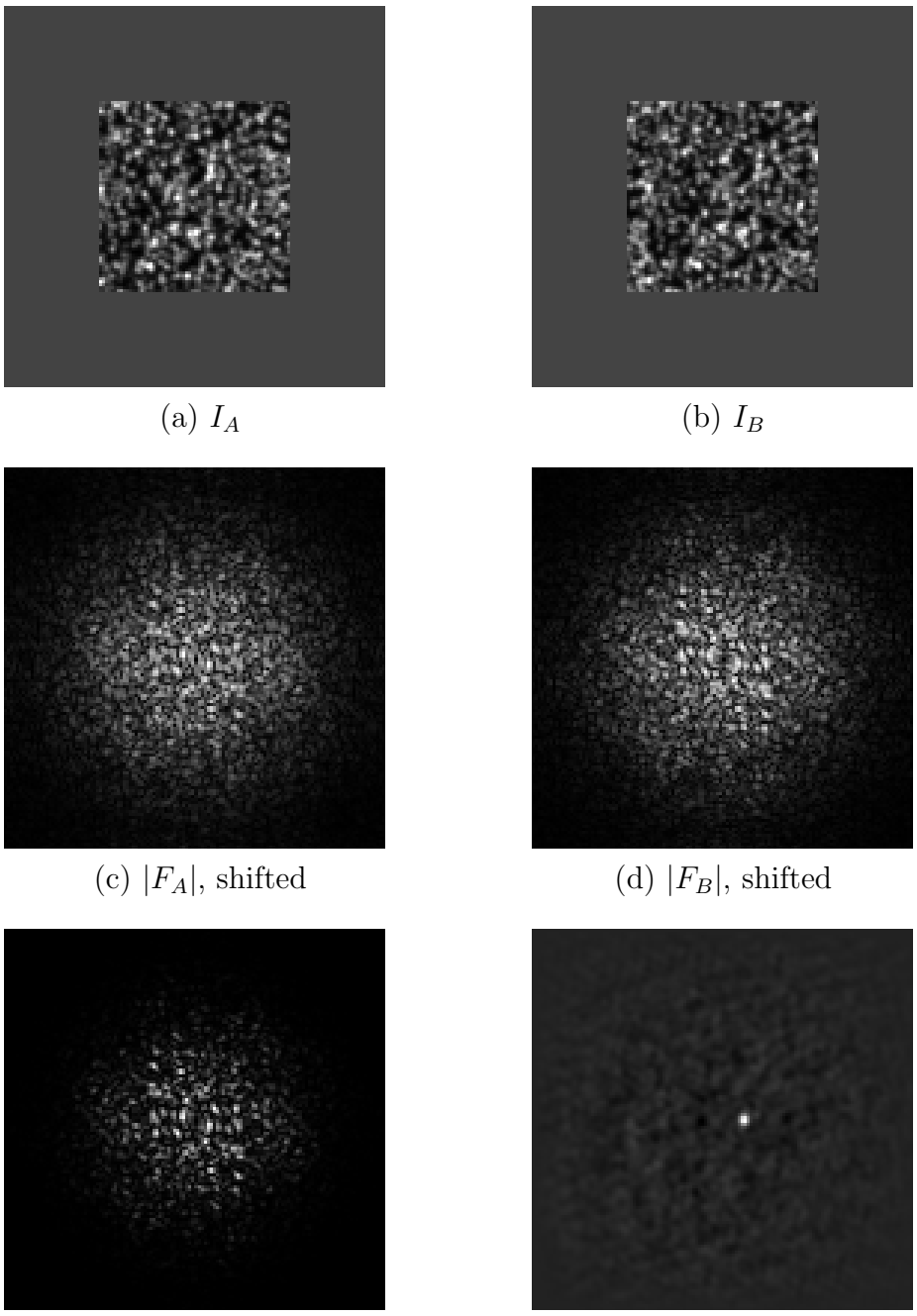
A further modification to the FFT method described here is to properly scale the correlation plane. As noted in Raffel et al. [7, p. 139], the *correlation coefficient* is the proper

normalization of the cross correlation, which allows a comparison between different correlation planes to be made. Additionally, the values of the correlation coefficient give a first-order approximation to the relative strength of the correlation. To modify the correlation plane to yield the correlation coefficient, the data is scaled by dividing it by the standard deviations of the original windows. Note that the standard deviation is calculated prior to the application of zero-padding. After this is performed, the values of the correlation plane will nominally fall in the range $-1 \leq R \leq 1$.

With the correlation plane calculated, the displacement is found by estimating the precise location of the peak value. The estimate can be broken into two components, the integer displacement m_0 and n_0, and the fractional estimate ϵ_X and ϵ_Y. The value of the total displacement is the sum of the integer and fractional displacements for each coordinate. The integer estimate is found using a simple maximum search within the correlation plane, excluding boundary pixels. The fractional displacement is determined using a Gaussian three-point estimator, which has become the standard method for subpixel peak detection in PIV. The Gaussian estimate is defined in equation 6.2, R_0 is the maximum value of the correlation plane at (m_0, n_0), and R_{-1} and R_{+1} are the values of the neighboring points. This equation can be applied separately and independently along all coordinate directions.

$$\epsilon = \frac{\ln R_{-1} - \ln R_{+1}}{2 \ln R_{-1} - 4 \ln R_0 + 2 \ln R_{+1}} \tag{6.2}$$

Note that the form of equation 6.2 requires a neighboring value of the correlation in each direction. For this reason, the integer estimator defined previously excludes boundary points. Also, the Gaussian estimator requires all values of the correlation plane to be positive (the natural logarithm of zero or a negative number is undefined). Due to this, the correlation plane is scaled in the estimator by subtracting the minimum value of the correlation plane, making all values of the plane greater than or equal to zero. To eliminate any possibility that the zero value could be encountered, a value of 1×10^{-10} is added to ensure all values are positive. These modifications to the correlation plane do not shift the estimated peak



(a) I_A (b) I_B

(c) $|F_A|$, shifted (d) $|F_B|$, shifted

(e) $|S|$, shifted (f) R, shifted

Figure 6.3: Intermediate steps for the calculation of the spatial correlation with the FFT. Window size 64 x 64 px. Data from case A2 of the 2005 PIV Challenge.

location, since the estimate is based on the ratio of the surrounding pixel intensities and not on the absolute intensity value of the pixels.

As an final step to determine the actual displacement, the estimate is reduced by an amount equal to half of the correlation plane size. This is because for zero displacement the peak is located at the center of the correlation plane while the estimators provide values given in image coordinates which begin at the upper-left corner of the plane. Equations 6.3 and 6.4 perform this shift. The additional subtraction of one is due to the nature of MATLAB image indexing, which begins at one.

$$d_x = (m_0 + \epsilon_X) - R_{x,size}/2 - 1 \tag{6.3}$$

$$d_y = (m_0 + \epsilon_Y) - R_{y,size}/2 - 1 \tag{6.4}$$

An example of this cross-correlation procedure without any window offset applied to an entire image is given in Figure 6.4. From a qualitative standpoint, the flow field of a turbulent jet is represented quite well. However, there do appear to be obvious outliers in the data. Therefore, validation procedures must be developed to detect outliers and replace them with suitable displacement values.

6.1.3 Vector Validation

To identify invalid vectors and replace them, a local validation process is used for detection. The multi-step procedure outlined here is similar to the one outlined by Adrian and Westerweel [3, p. 512].

1. Local outlier detection using the normalized median test. Vectors exceeding a threshold value of two are tagged as invalid.

2. Bilinear interpolation to replace vectors tagged as invalid.

3. Repeat process until convergence.

The normalized median test is described by Westerweel and Scarano [64] and is a local validation test based on the median value of the velocity components of the neighboring

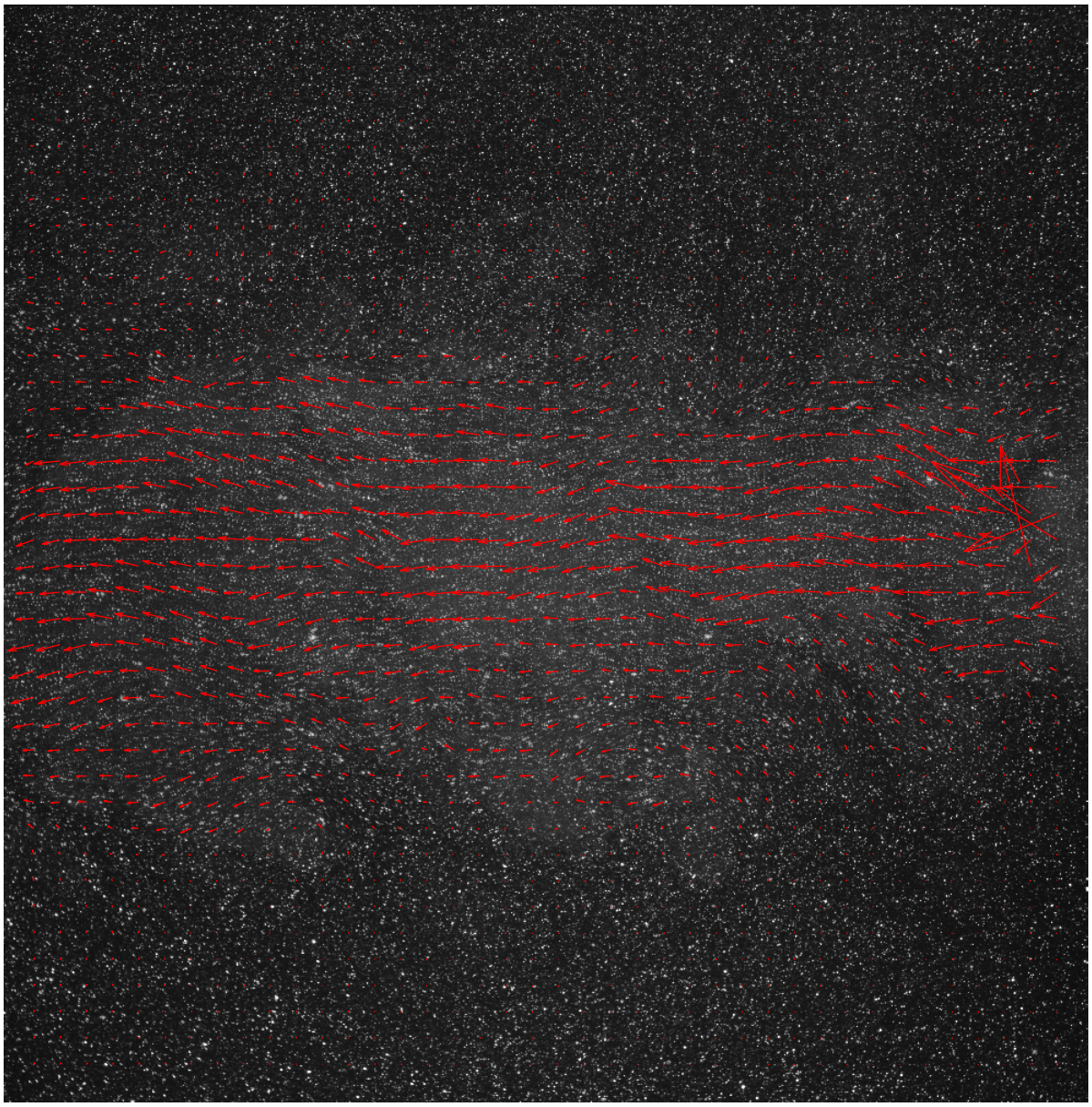


Figure 6.4: Example of a cross-correlation performed using 48 x 48 px interrogation window sizes with 24 x 24 px spacing and no window offset. Vectors scaled by a factor of 5. Data from case A, image 2 of the 2003 PIV Challenge.

eight vectors. The technique is robust and applies to a wide variety of flows, including those with large velocity gradients, and is now the predominant validation technique used in PIV codes.

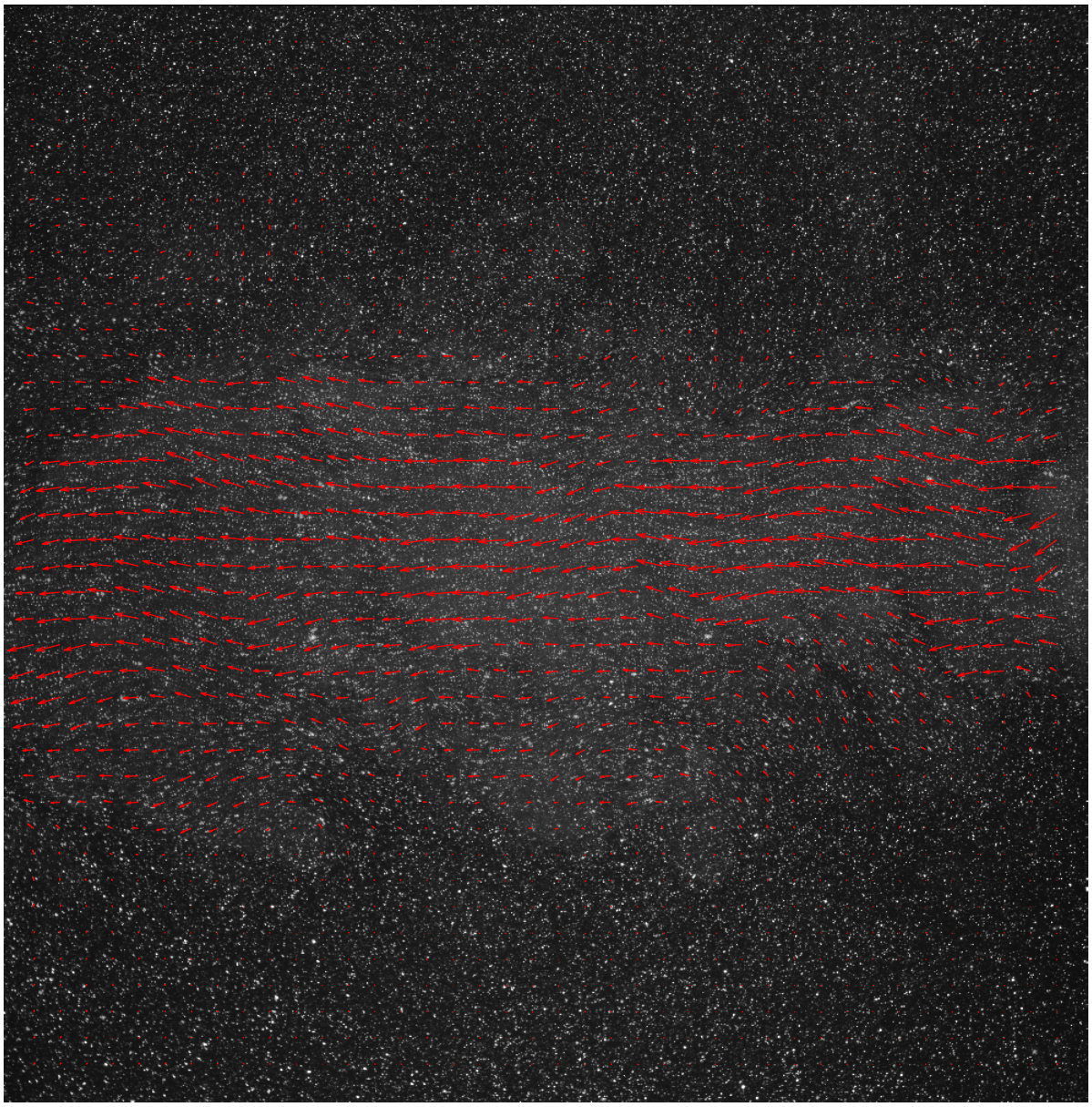


Figure 6.5: Vector field from Figure 6.4 using the iterative validation and replacement procedure.

6.1.4 Predictor Generation

After a initial interrogation pass performed without a window offset, the predictor for the next iteration can be generated by simply rounding the value of the displacement to the nearest integer value. The resulting values p_x and p_y are used to offset the individual

windows of image B from image A. An example of the predictor field is shown in figures 6.6 and 6.7.

A more complex approach is needed to apply the predictor to a refined grid. As the grid resolution increases, the values of the predictor must be interpolated from the displacement field of the previous grid. For this reason, a bilinear interpolation is performed with the MATLAB function `interp2`. The results of the interpolation are then rounded to yield integer values.

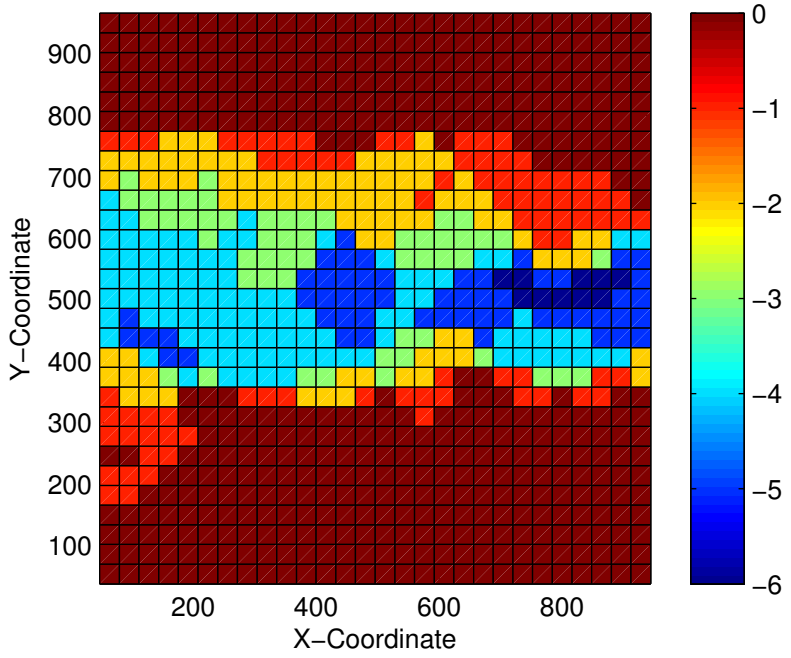


Figure 6.6: X-Coordinate Predictor using 64 x 64 px interrogation window sizes with 32 x 32 px spacing. Data from case A, image 2 of the 2003 PIV Challenge.

6.1.5 Grid Refinement

After multiple passes using identical grid settings, the velocity field will converge. To increase the spatial resolution of the algorithm, a new grid can be generated using a smaller window size and window spacing. To accomplish this a refinement factor R_f is defined, which is taken to be an integer value going from 1 to as much as 4. The grid parameters at

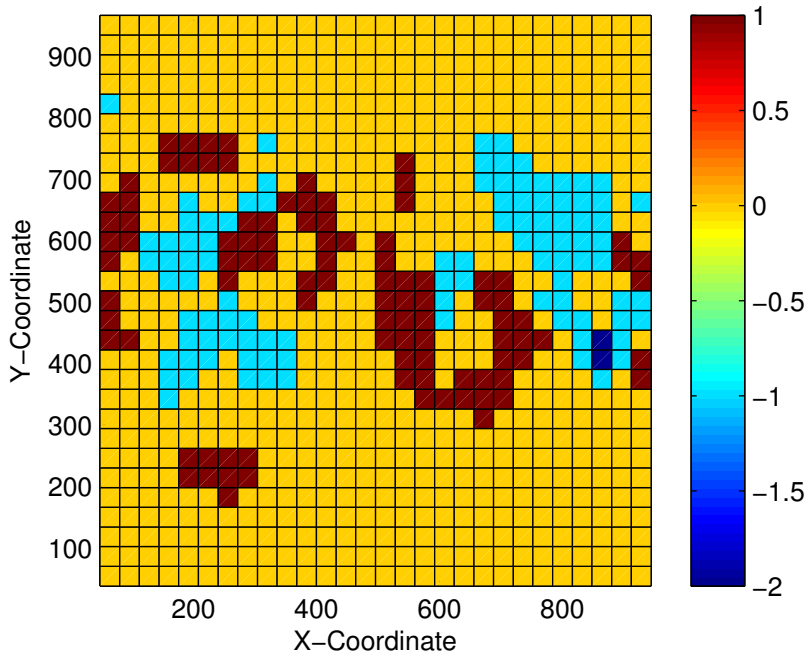


Figure 6.7: Y-Coordinate Predictor using 64 x 64 px interrogation window sizes with 32 x 32 px spacing. Data from case A, image 2 of the 2003 PIV Challenge.

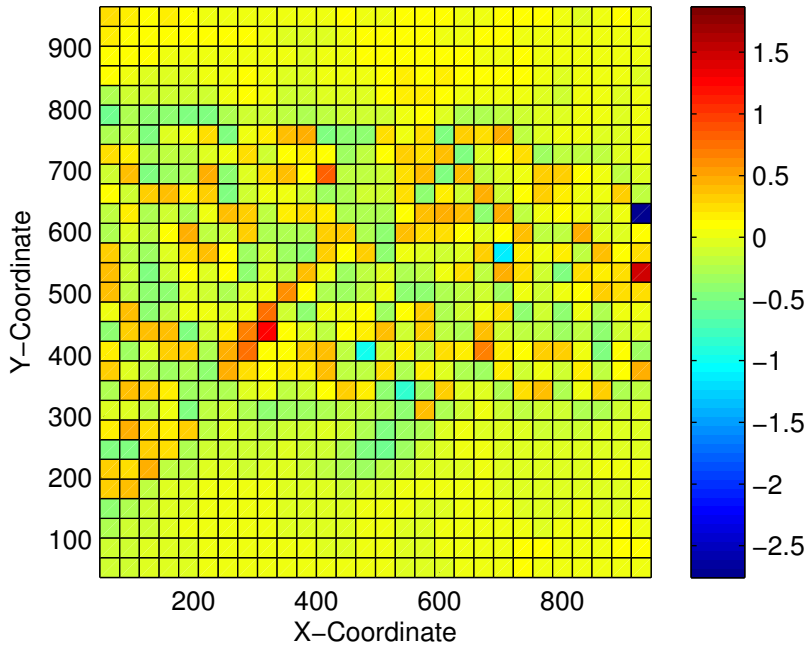


Figure 6.8: X-Coordinate Corrector using 64 x 64 px interrogation window sizes with 32 x 32 px spacing. Data from case A, image 2 of the 2003 PIV Challenge.

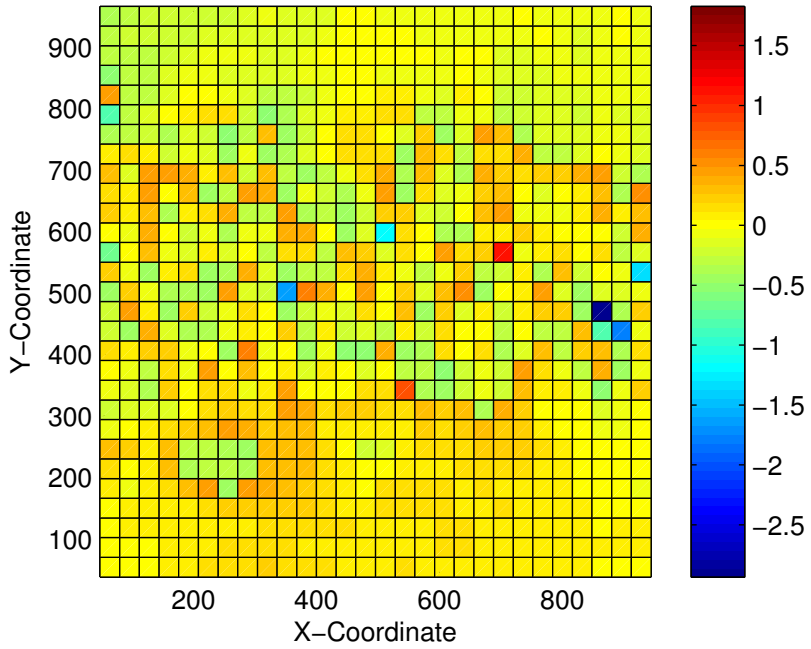


Figure 6.9: Y-Coordinate Corrector using 64 x 64 px interrogation window sizes with 32 x 32 px spacing. Data from case A, image 2 of the 2003 PIV Challenge.

each refinement level are then defined in equations 6.5 to 6.8, where $x_{size,i}$, $y_{size,i}$, $x_{spacing,i}$, and $y_{spacing,i}$ are the initial grid settings.

$$x_{size} = \text{ceil}(x_{size,i}/R_f) \tag{6.5}$$

$$y_{size} = \text{ceil}(y_{size,i}/R_f) \tag{6.6}$$

$$x_{spacing} = \text{ceil}(x_{spacing,i}/R_f) \tag{6.7}$$

$$y_{spacing} = \text{ceil}(y_{spacing,i}/R_f) \tag{6.8}$$

In the above equations, the `ceil` function is used to ensure integer values of all inputs. Additionally, a check is performed on both x_{size} and y_{size} to ensure that they are even values. This restriction is due to the way the `fftshift` function in MATLAB reshuffles the correlation plane. After the grid parameters are set, the grid generation function generates

new node and centroid points for use in the calculations. An example of the grid refinement is given in figures 6.10 through 6.13.

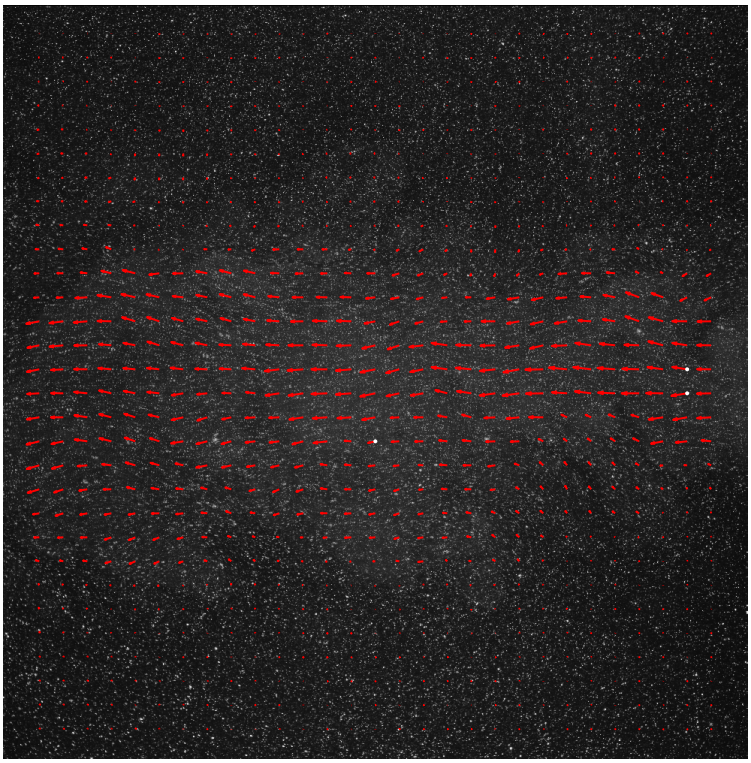


Figure 6.10: $R_f = 1$. Initial percentage of valid vectors: 97%.

From 6.10 to 6.13, the spatial resolution increases dramatically; however, at high refinement levels a high percentage of invalid vectors exist. This is due to the reduced number of particles forming a distinct correlation peak, as detailed in Adrian and Westerweel [3, p. 346]. In this work, the acceptable maximum refinement level is defined as the highest level which can be achieved with greater than 95% valid vectors on the first validation pass. Another way to visualize this behavior is in figures 6.14 and 6.15. In the former, the RMS of the corrector values are plotted alongside the refinement value. Note that due to the multi-pass procedure, within each grid refinement step the RMS error generally decreases significantly. The second figure shows the performance of the vector validation and replacement scheme along with the refinement number. Note the substantial drop in valid vectors when the refinement factor $R_f = 4$.

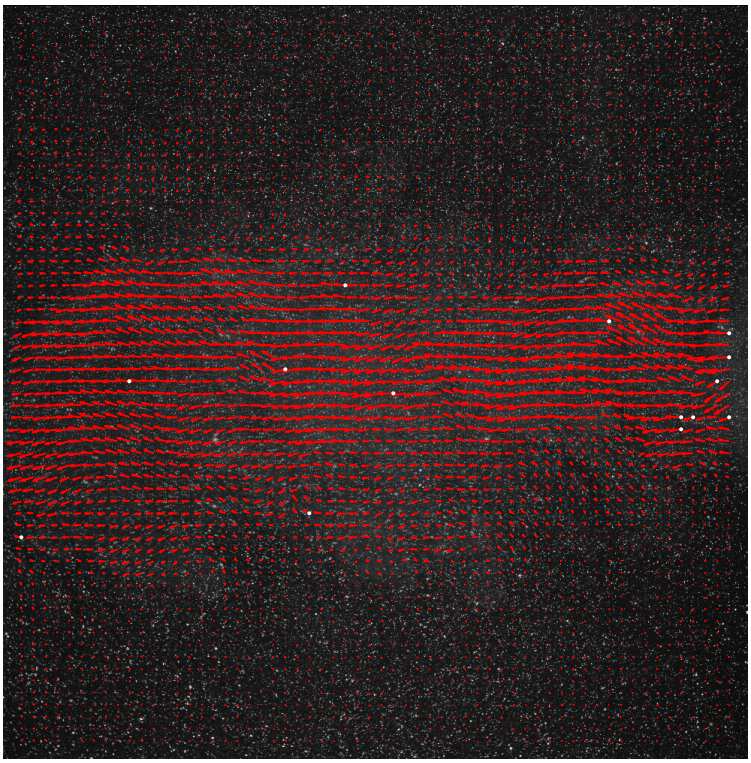


Figure 6.11: $R_f = 2$. Initial percentage of valid vectors: 97%.

6.2 Extension to 3-D PIV

Extending the WIDIM algorithm to 3-D requires each component of the algorithm to be adapted for 3-D correlation. This is a relatively simple procedure which does not involve substantial modification of the PIV code. A notable change is the use of the `fftn` and `ifftn` commands for performing the 3-D FFT and inverse FFT, respectively. However, the images do need to be conditioned prior to being loaded into the code. This is because the output from the rendering code has equal resolution at each plane but actually represents a trapezoidal volume in which the magnification changes at each plane. Thus, if a modification is not made to the individual refocus planes, the differences in magnification through each interrogation volume will result in a slight shear. To correct for this, the data in each of the refocused images is resized in a manner such that the magnification is set to a constant value while no information is lost. This is done by calculating the image size for each plane using the relation $(t + s_i)/s_i$. The base resolution is chosen as the plane with the largest scale

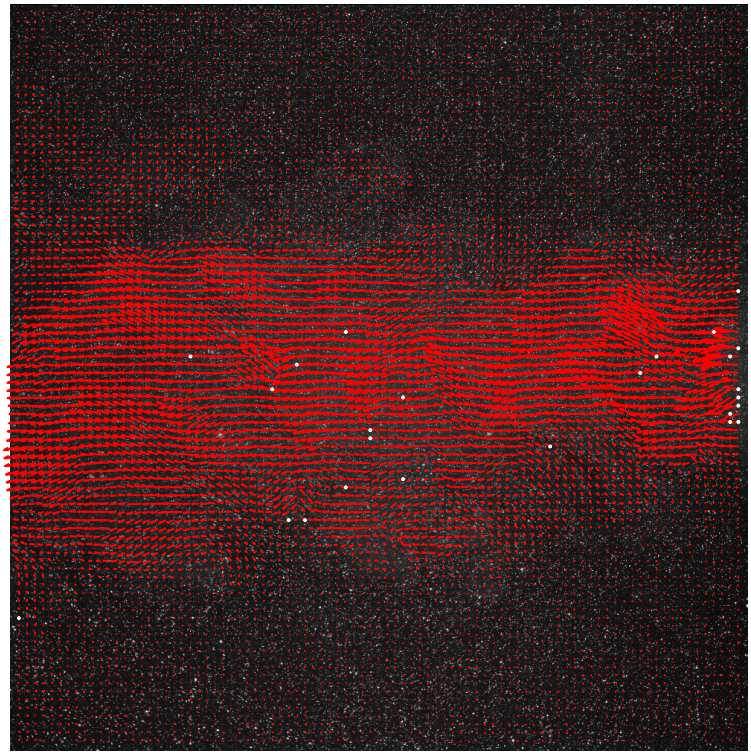


Figure 6.12: $R_f = 3$. Initial percentage of valid vectors: 95%.

factor. This procedure is shown schematically in Figure 6.16. This procedure takes a data set consisting of multiple planes at identical pixel resolution but different magnifications due to their distance from the lens. By performing a resizing operation on each of the planes, the magnification can be adjusted such that the image data now represents constant magnification. For the resizing, data outside of the image range is taken to be zero. The shaded regions in Figure 6.16 correspond to areas filled with zeros. An example of this applied to a set of 2-D data is given in Figure 6.17.

One of the most important parameters in PIV interrogation is particle number density within the interrogation windows, which is designated N_I. It can be defined as the average number of particle images within each interrogation window or volume. It is well documented that increasing the number density directly increases the peak detection probability and accuracy [3, p. 343], and suitable values typically occur for $N_I > 10$ [3, p. 350]. This parameter can be controlled in an experiment by introducing or removing seeding particles,

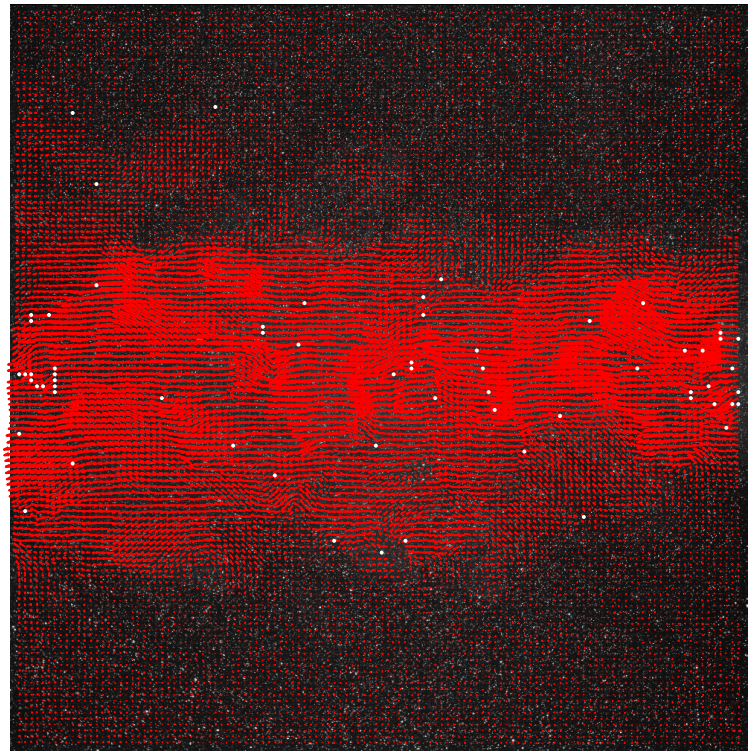


Figure 6.13: $R_f = 4$. Initial percentage of valid vectors: 85%.

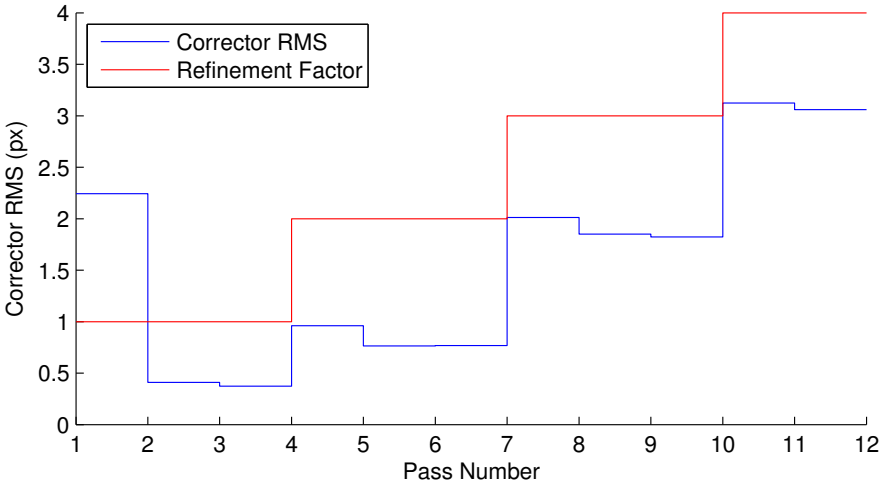


Figure 6.14: Corrector RMS for the multi-pass algorithm. The refinement factor R_f is also plotted for clarity.

or at processing time by modifying the size of the interrogation windows or volumes. For measurements desiring high spatial resolution, the interrogation window size must be made as small as possible. Thus to ensure N_I is a large enough value in these cases, the seeding

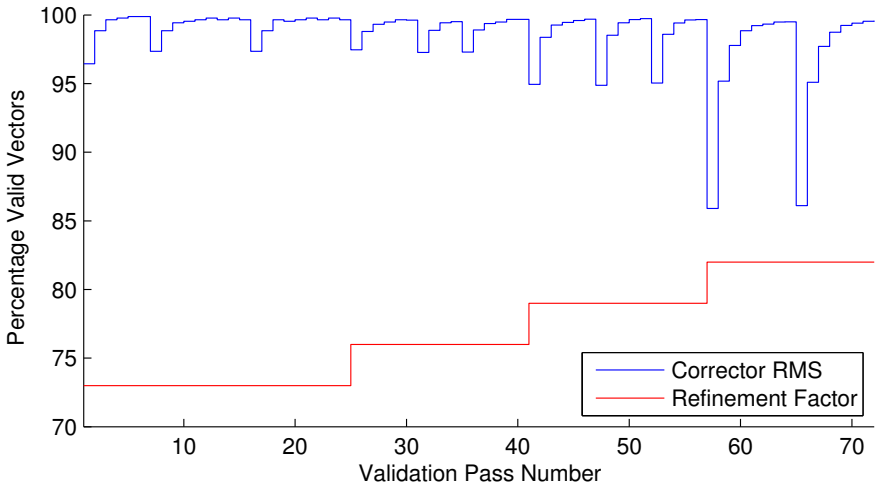


Figure 6.15: Percentage of valid vectors for the multi-pass algorithm. The refinement factor R_f is also plotted for clarity.

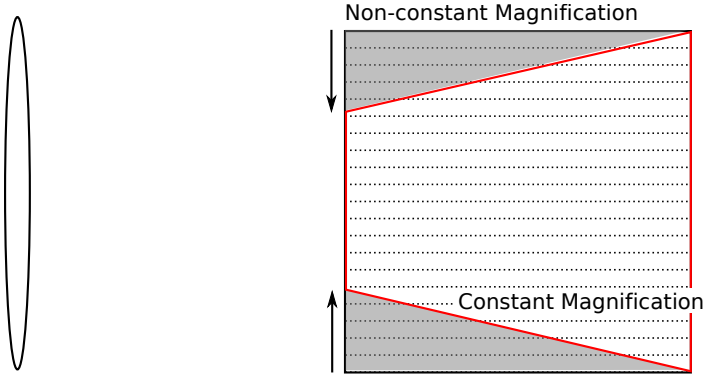
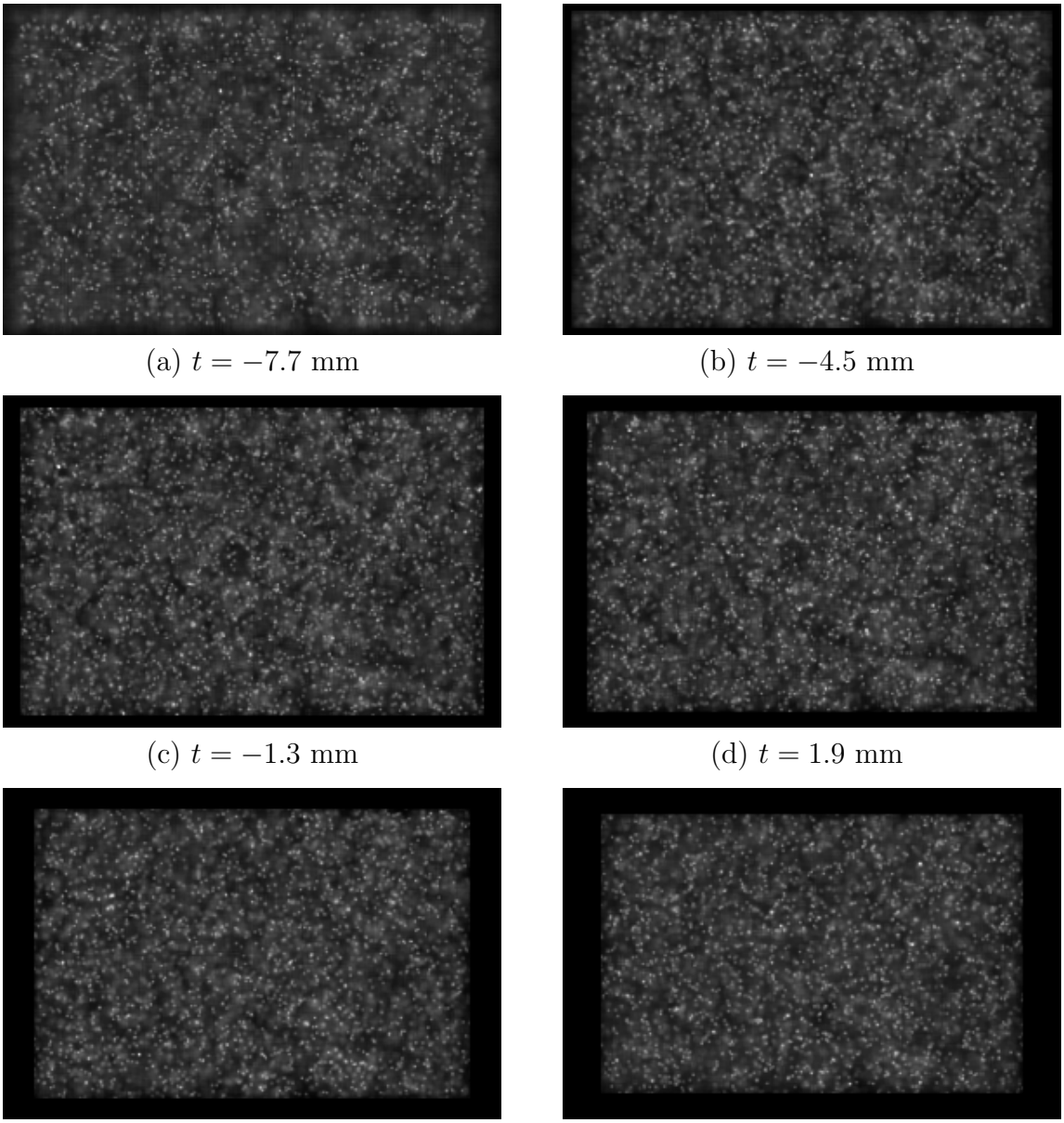


Figure 6.16: Schematic of resize operation.

concentration is often increased in the case of a traditional PIV measurement. The limitation in seeding concentration typically occurs when laser speckle noise becomes the dominant factor in the imaging or the line of sight loses transparency due to the dense particle field.

For 3-D PIV, most techniques do not directly measure the intensity field and for this reason require reconstruction procedures to generate a volumetric intensity field. Through the use of Monte Carlo simulations and actual experiments, all reconstruction procedures (MART, SAPIV Thresholding, etc.) have limits on the particle density of fields to yield acceptable reconstructions. In essence, the primary limitation of spatial resolution in 3-D PIV is due to the limited particle density able to be reconstructed. It is estimated that the



(a) $t = -7.7$ mm (b) $t = -4.5$ mm

(c) $t = -1.3$ mm (d) $t = 1.9$ mm

(e) $t = 5.1$ mm (f) $t = 8.3$ mm

Figure 6.17: Example of resizing operation applied to a set of refocused planes. Note that in these images the thresholding operation has not been applied to give a clearer view of the resizing procedure.

technique described here will have similar limitations to the SAPIV thresholding technique, but this must be verified. Additionally, the 3-D PIV code itself must be verified. For this reason, the technique is first tested on simulated uniform velocity fields independently in both the x and z directions. This allows the accuracy of the technique to be evaluated for

both in-plane and out-of-plane motion. The variable to be changed in these simulations is the particle number density, measured in particles per pixel (ppp).

Chapter 7

Results

Testing of the simulation, rendering, and PIV algorithms is performed in this section by analyzing cases of uniform in-plane and out-of-plane displacements and a spherical vortex simulation. The resulting velocity fields are screened for bias errors, and the accuracy is estimated for motions in the in-plane and out-of-plane directions.

7.1 Uniform X-Direction Displacements

A uniform displacement in the x-direction was tested to identify any bias errors which could be present in the rendering codes or PIV algorithms. A uniform shift of 4.5 pixels was used; the pixel size was determined by the 125 μm pitch of the lenslets, so the physical shift input into the simulator was 0.562 mm. For these tests a relatively low particle density of 0.001 particles per pixel (ppp) was used. Particles per pixel is defined in this case as the number of pixels of the CCD, not of the refocused images. By comparison, tomographic PIV and other 3-D techniques typically use seeding densities an order of magnitude greater. 16 refocused planes are generated using the plane spacing derived from the depth of field equations. An example of the velocity field generated in this configuration is given in Figure 7.1. In this run, 4800 vectors are generated while only 1 out of every 3 vectors is shown for clarity. From a qualitative perspective, it is clear from this overview that the PIV algorithms do not result in substantial outliers. A more rigorous comparison of the accuracy is performed by generating displacement histograms for each coordinate direction.

Figure 7.2 is a histogram of the x-displacements using 50 bins. A Gaussian function is fit to the data to show the excellent agreement between the histogram and the normal distribution. This is an indication that the velocity vectors are complying with the statistical

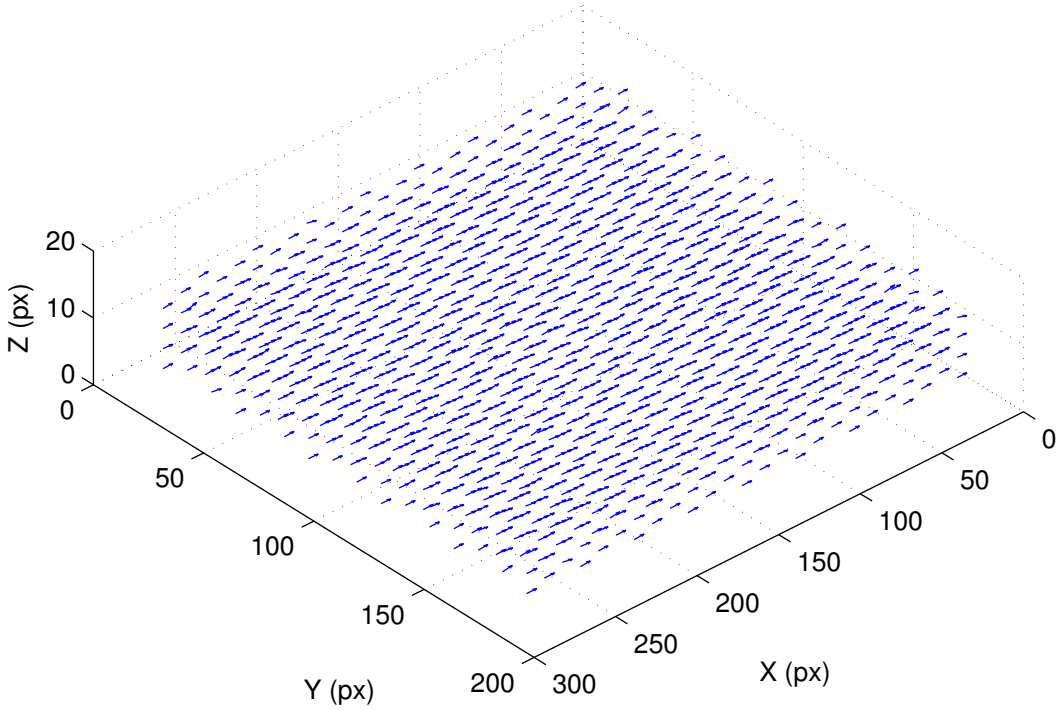


Figure 7.1: Velocity field using 0.001 particles per pixel and a simulated 4.5 pixel shift in the x-direction.

central limit theorem, which states that the compilation of many independent, identical random variables (such as the random error in PIV, [3, p. 380]) will tend towards a normal distribution. The annotations in Figure 7.2 indicate the mean and standard deviation of the samples, not of the Gaussian curve fit.

Note that the mean value appears to show a slight underestimation of the actual displacement (4.5 px). This is likely a bias error inherent in the PIV processing itself. The value is consistent with the magnitude of the displacement bias error described in Adrian and Westerweel [3, p. 356]. This underestimation occurs when the values of the correlation peak are slightly skewed due to the finite size of the interrogation regions. This leads to a systematic underestimation of the actual displacement. Modern PIV interrogation methods eliminate this issue by using fractional window offset techniques to position the correlation peak at the center of the correlation plane which eliminates the bias error. These techniques are more complicated due to the use of image interpolation schemes which introduce

additional complexity in the interrogation steps. However, the magnitude of this error is relatively small and is on the order of the random error magnitude for PIV.

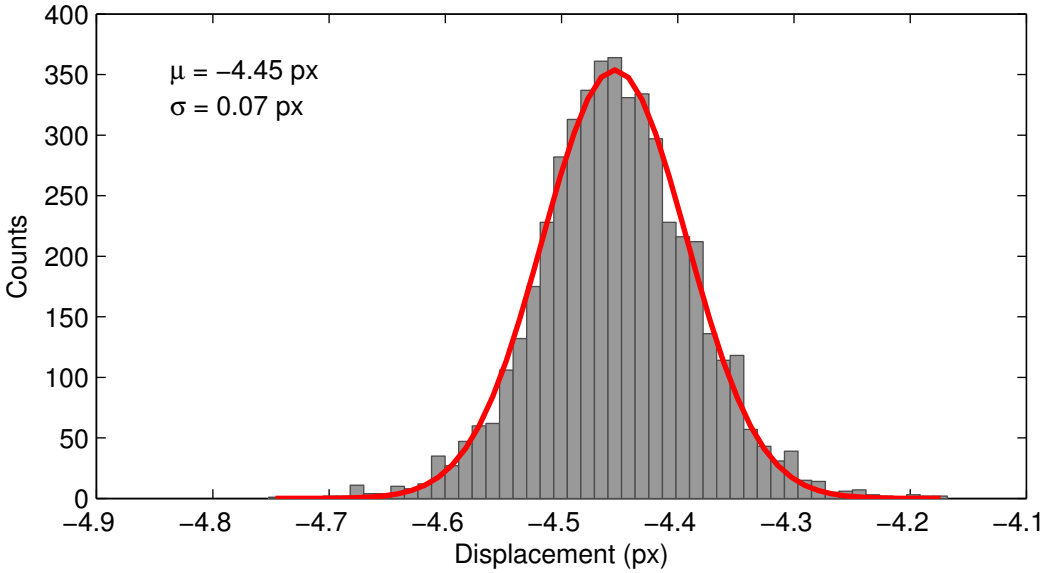


Figure 7.2: Histogram of x-displacements. 50 bins used over 4800 data points.

Histograms were also generated in the other coordinate directions for the case of uniform x-coordinate displacement. Figure 7.3 presents the displacements in the y-direction, and Figure 7.4 presents the data in the z-direction. In both cases, the mean value of the displacement is zero, and the standard deviation of the displacement is in agreement with the random error magnitude typical of PIV interrogations [3, p. 380].

As a further test to evaluate the accuracy of the rendering codes, the image resizing step was removed to gauge the effect on the PIV interrogation. By removing the image resize step, the magnification is no longer constant throughout the image, and it should be expected that a depth-dependent bias should exist. Figure 7.5 shows a histogram of x-component vectors for this scenario. The bimodal distribution in the histogram is due to the change in magnification. This is clarified in Figure 7.6, which shows the error increasing as the distance from the central plane also increases. In this figure, the average displacement was subtracted to better illustrate the spatial distribution of the bias.

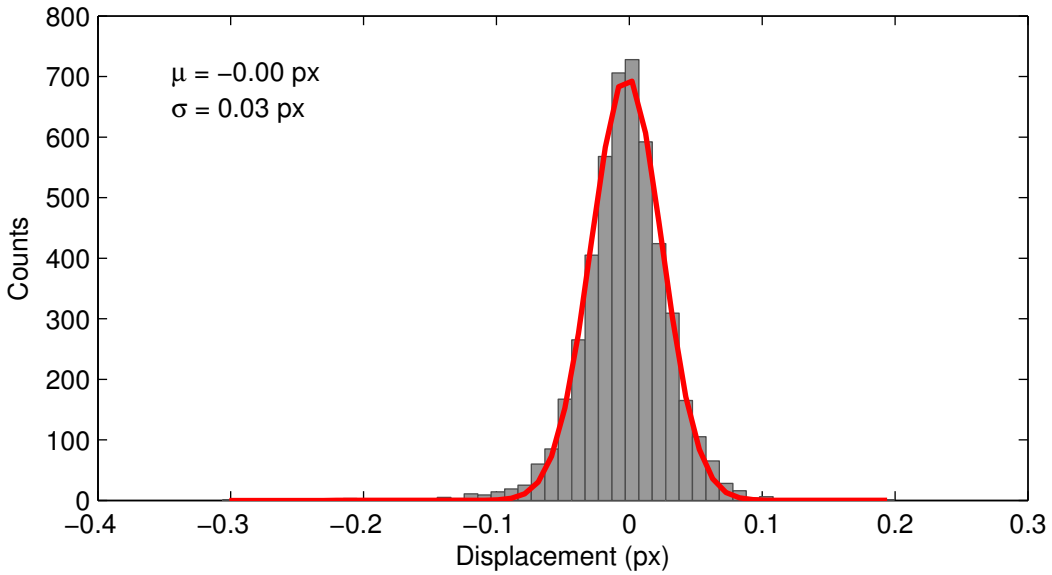


Figure 7.3: Histogram of y-displacements. 50 bins used over 4800 data points.

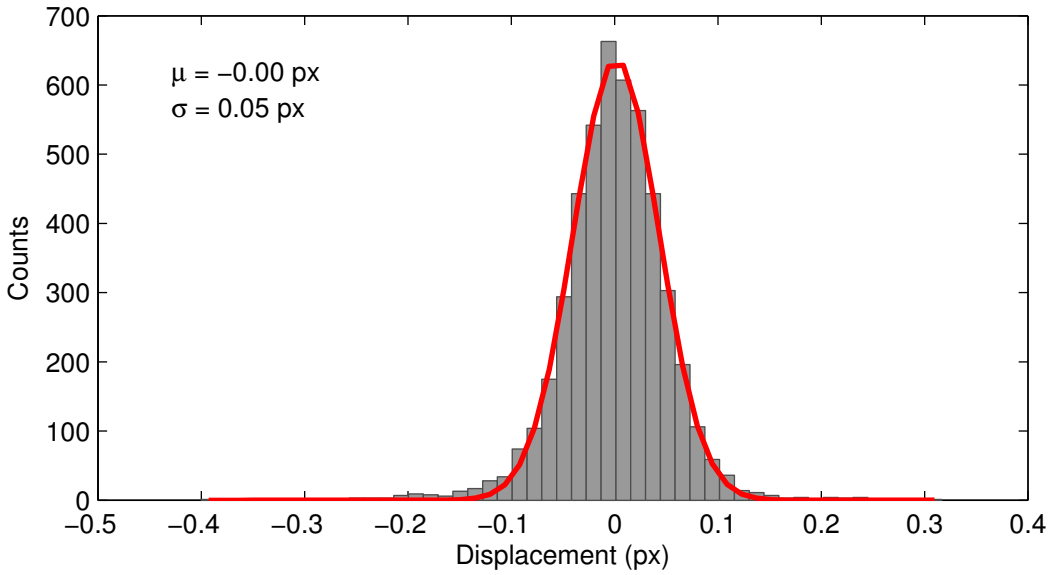


Figure 7.4: Histogram of z-displacements. 50 bins used over 4800 data points.

7.2 Uniform Z-Direction Displacements

Particle displacements in the z-direction are particularly challenging as the number of pixels in the depth direction (16) is significantly lower than that in the in-plane directions. For this reason, the interrogation volume size in the z-direction must be reduced to allow for multiple planes to be measured. For the cases studied here, the interrogation window

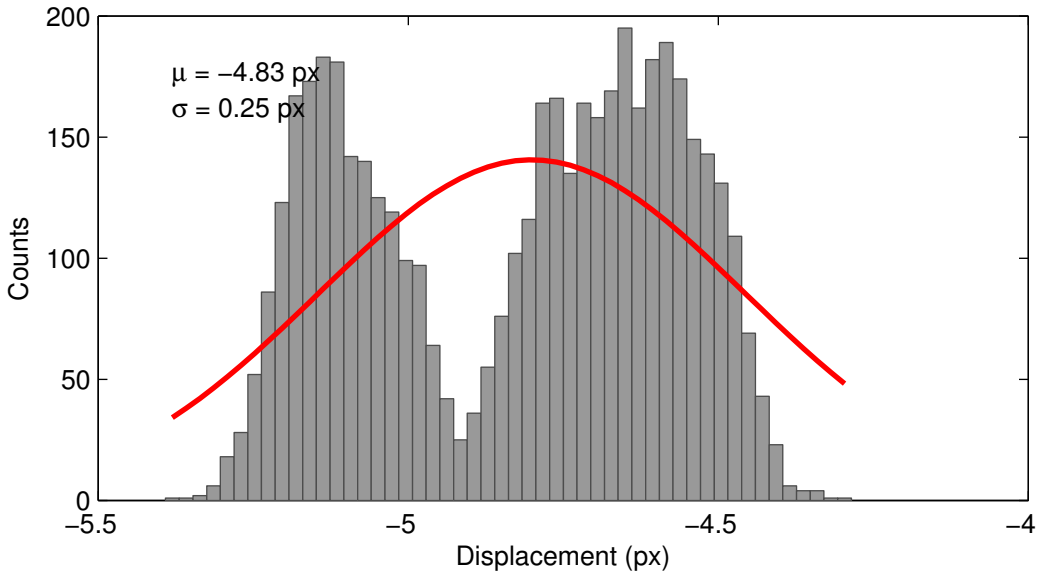


Figure 7.5: Histogram of x-displacements without using image resizing. 50 bins used over 4800 data points.

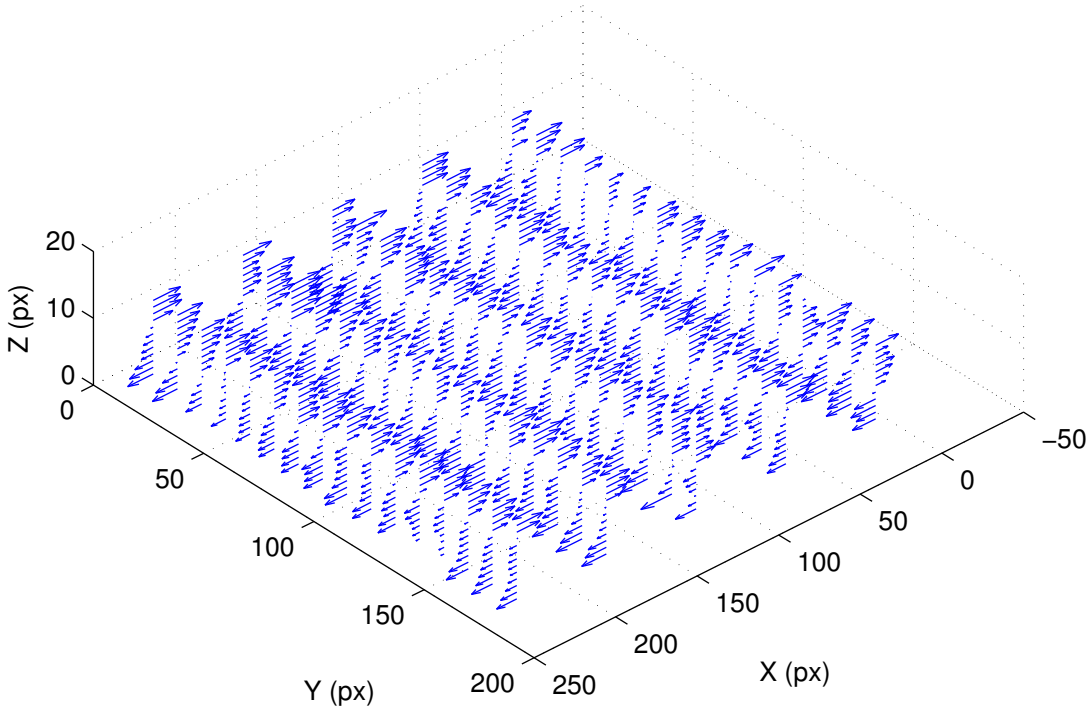


Figure 7.6: Vector field showing one of every 5 vectors without image resizing. Mean value of x-component subtracted, and both y and z components set to zero for effective visualization.

initial z_{size} was set to 16 px, and the initial $z_{spacing}$ is 8 px. Additionally, the first tests of the uniform z-displacement showed severe bias errors related to particles leaving the rendered

measurement volume. For this reason, the rendering was expanded by 50% for both positive and negative depths, and the number of refocus planes generated was doubled (32). This allows for particles located near the edge of the measurement volume to gradually fade out of the measurement volume rather than be cut off. This procedure was found effective for eliminating biases near the boundaries of the volume.

Figure 7.7 shows the velocity field obtained for the uniform z-displacement case with a nominal displacement of 2.5 pixels. One out of every four vectors is shown for clarity. From a qualitative standpoint, it captures the displacements with few major outliers. As for the previous case, a more substantial analysis can be made by analyzing displacement histograms. Figure 7.8 shows a histogram of the z-displacements. Notable in this figure is the presence of a bimodal structure centered on integer displacements, likely due to the "peak-locking" effect in PIV. There also appears to be a small outlier at 0 px, likely due to a region of interrogation windows without particles near the edges of the volume.

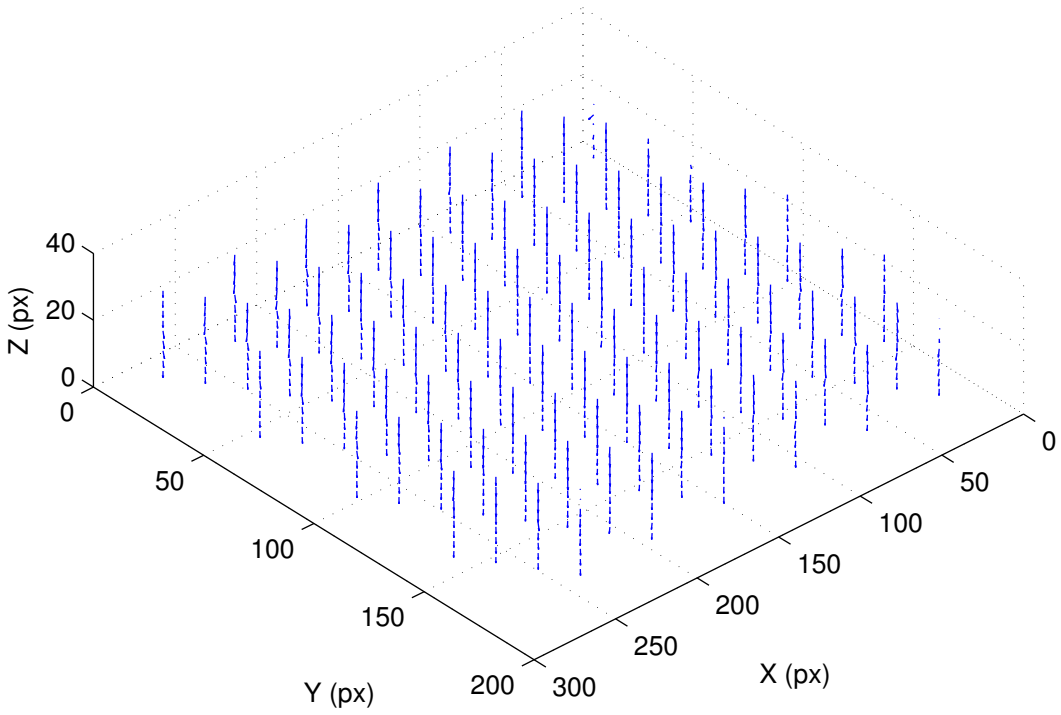


Figure 7.7: Vector field for uniform z-displacement showing one of every 4 vectors.

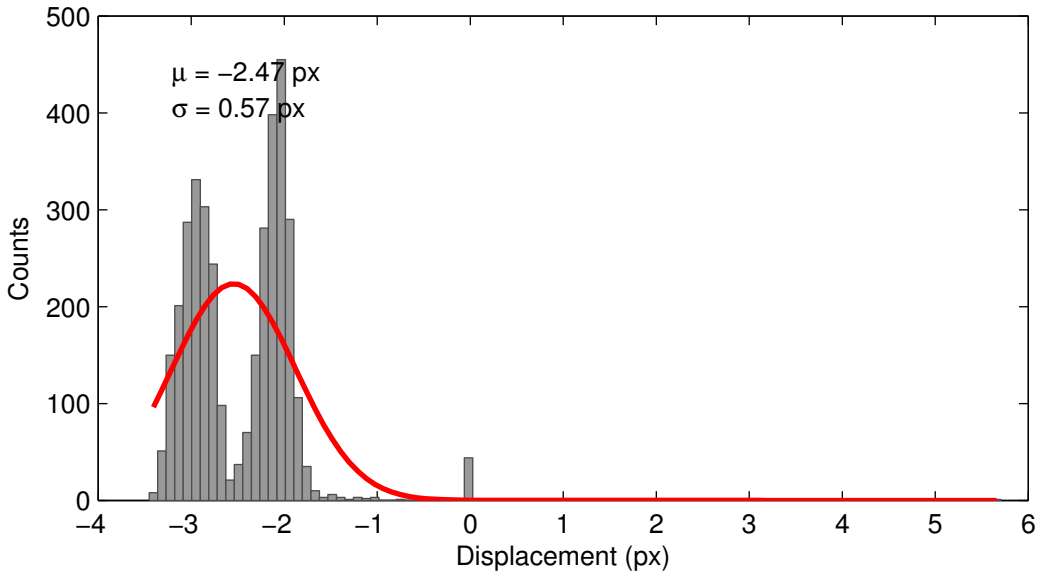


Figure 7.8: Histogram of z-displacements. 100 bins used to highlight peak-locking effect more clearly.

To confirm these ideas, a vector field is shown in Figure 7.9 where the mean displacement has been subtracted from the actual displacement, giving a view at the spatial distribution of the bias errors. It is clear through inspection of this figure that there appears to be an inward bias towards the edge of the interrogation volume, and the outliers corresponding to zero displacement are found to occur at the edges of the volume, where there could potentially be few particle images. Note that in this figure, the vectors have been scaled by a factor of 4 to highlight the effects.

These findings indicate that z-displacements can be measured; however, they are highly sensitive to peak-locking effects. By modifying the thresholding and rendering procedures in future efforts, it may be possible to reduce the peak-locking effect and potentially reduce biases at the edge of the measurement volume. Histograms of the displacement in both the x-direction and y-direction are shown in figures 7.10 and 7.11, respectively. As in the previous case of uniform x-displacement, the errors in other coordinate directions are minimal and correspond to the magnitude of the random error in PIV.

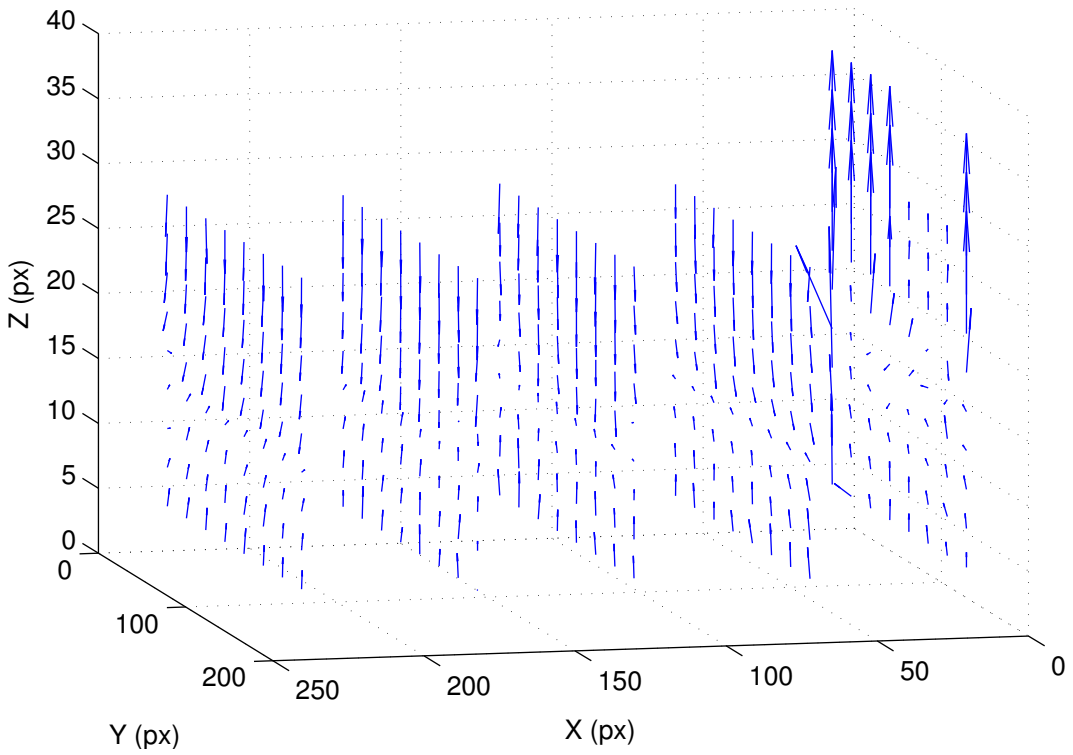


Figure 7.9: Vector field for uniform z-displacements (same as Figure 7.7). Mean value of the displacement subtracted to show location and direction of bias.

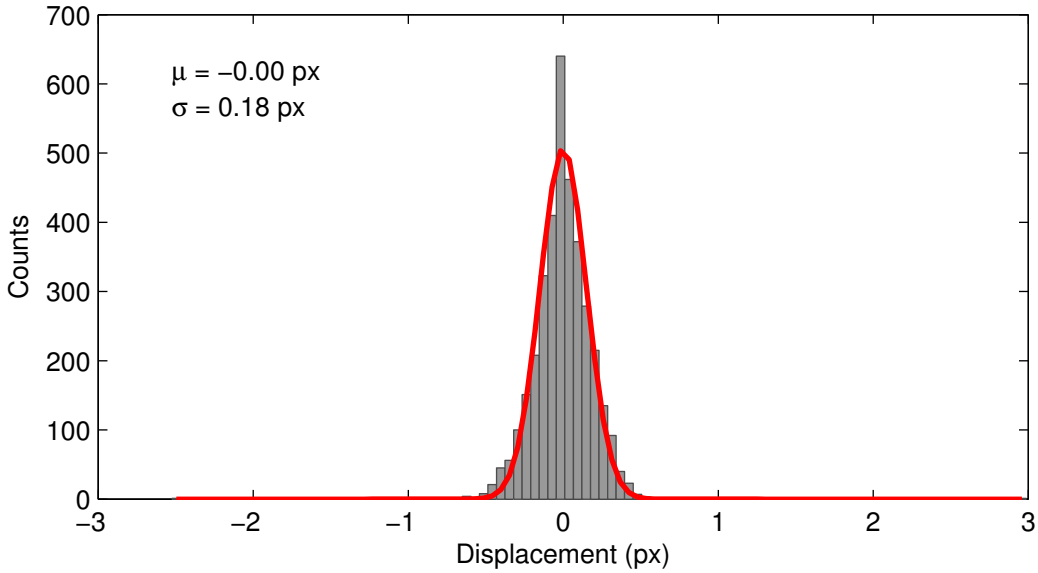


Figure 7.10: Histogram of x-displacements. 100 bins used over 3600 data points.

7.3 Variation of Particle Number Density

As previously noted, the particle number density is a critical parameter in PIV, which specifies the maximum spatial resolution that can typically be achieved. The number density

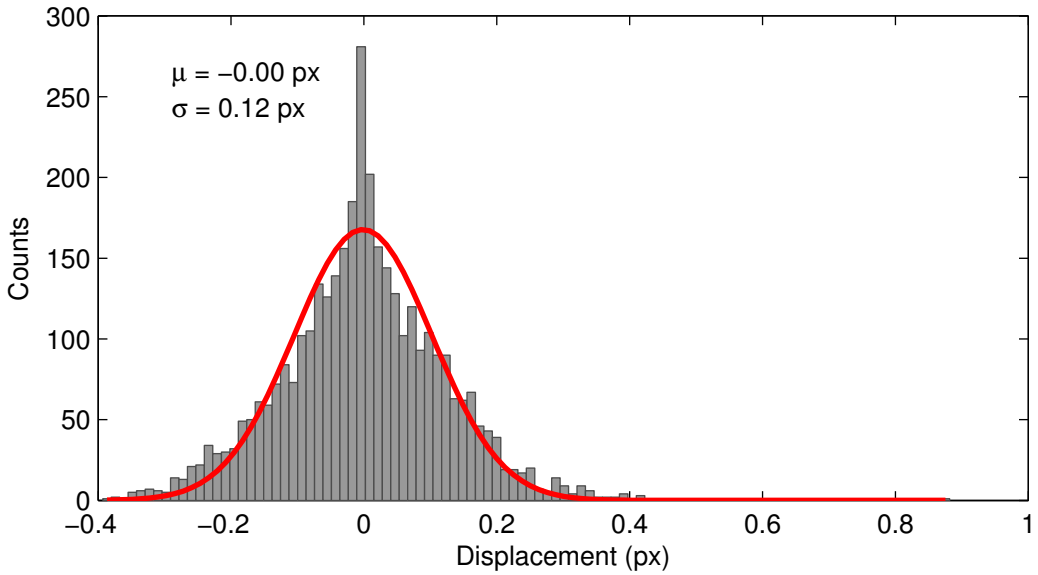
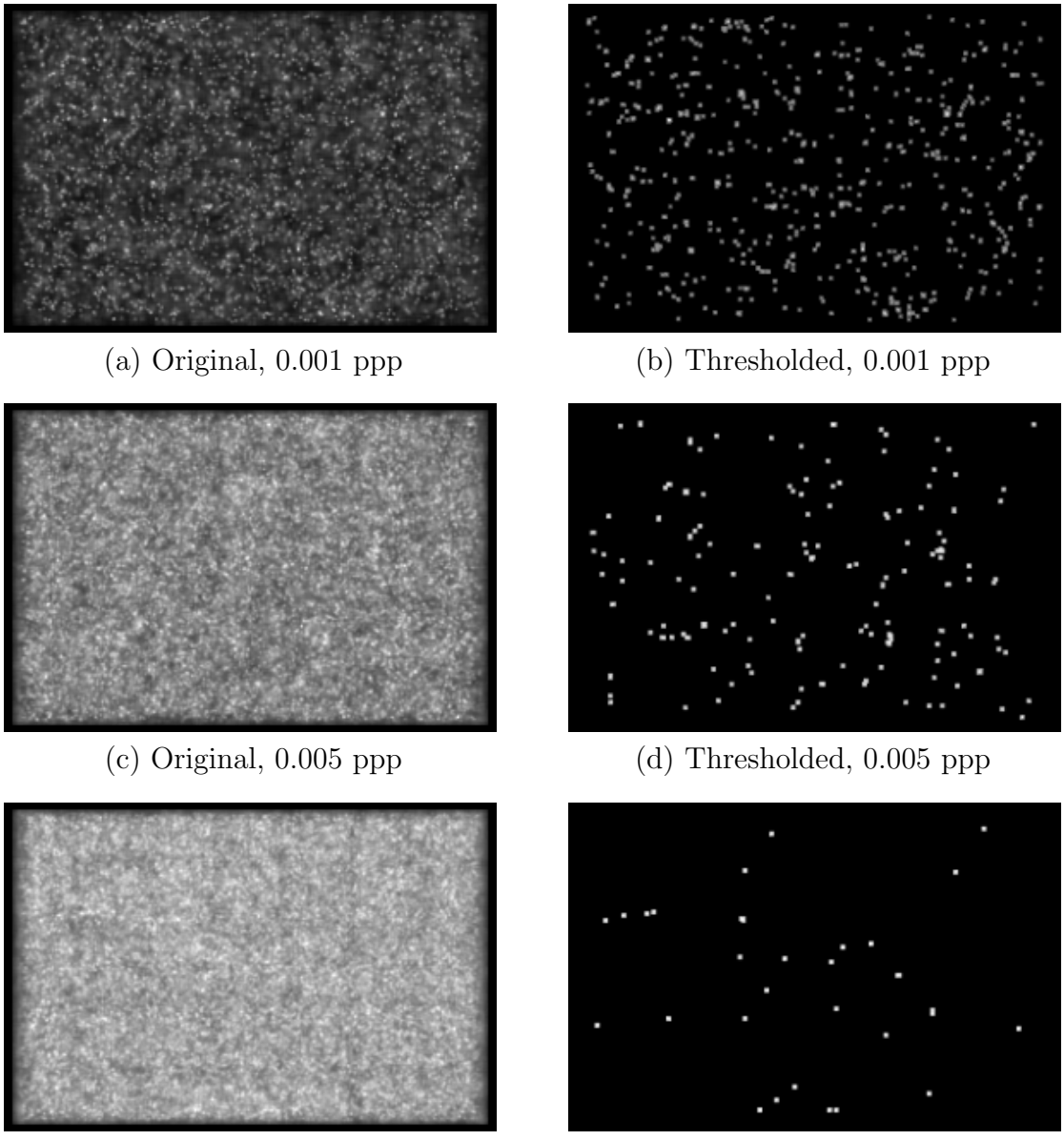


Figure 7.11: Histogram of y-displacements. 100 bins used over 3600 data points.

was increased in the simulations from the base value of 0.001 ppp to 0.005 and also 0.01 ppp. To demonstrate the effect of this increase on the refocused planes, Figure 7.12 shows an example refocused plane from each case, with and without the thresholding applied. An interesting result can be found from this figure that the thresholding technique performs poorly at high particle densities. The reduced number of particles in the thresholded images precludes performing any PIV interrogation with these data sets.

To understand why the thresholding procedure deteriorates at high particle image densities, the intensity histograms are compared for both 0.001 and 0.01 ppp in figures 7.13 and 7.14. Clearly, at high particle densities the intensity distribution becomes noticeably more Gaussian and the threshold level is increased. It should be noted that the Gaussian intensity distribution and high threshold levels demonstrated in Figure 7.14 also exist for the individual planes of the rendered volume, not just the entire volume. This indicates that the use of a unique threshold calculated for each focal plane would have a minimal effect for improving the discrimination of particles.

Based on these results, PIV was not attempted at the higher particle densities of 0.005 and 0.01 ppp, since the low number of particles in the thresholded images would result in



(a) Original, 0.001 ppp

(b) Thresholded, 0.001 ppp

(c) Original, 0.005 ppp

(d) Thresholded, 0.005 ppp

(e) Original, 0.01 ppp

(f) Thresholded, 0.01 ppp

Figure 7.12: Example of resizing operation applied to a set of refocused planes. Note that in these images the thresholding operation has not been applied to give a clearer view of the resizing procedure.

interrogation volumes with few or no particles and thus invalid results. As a rule of thumb, accurate PIV evaluation requires greater than 10 particle images within each volume [3, p. 350].

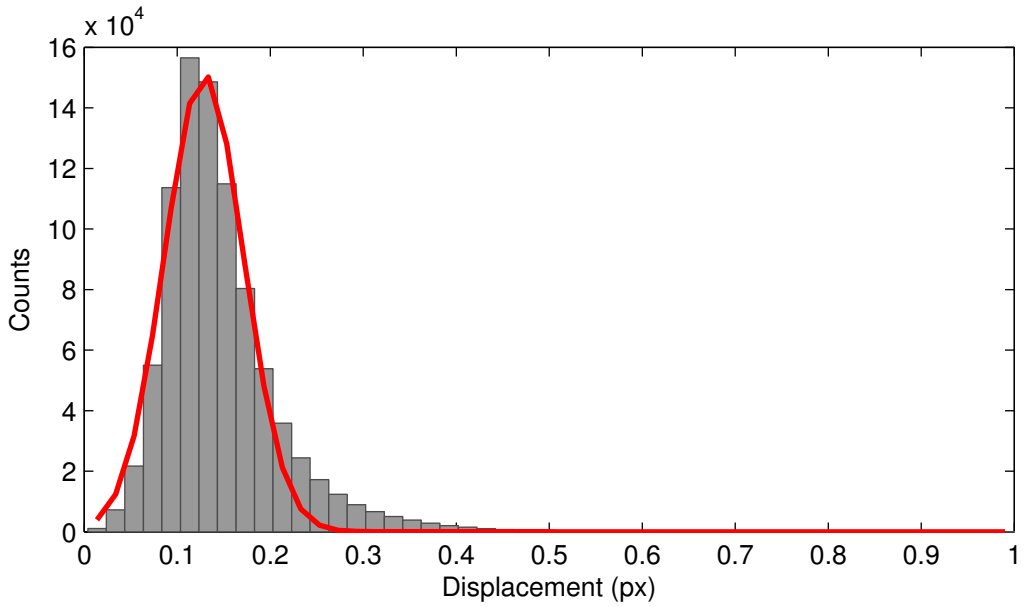


Figure 7.13: Histogram of the rendered intensity volume corresponding to Figure 7.12a. The 3σ threshold level for this case is 0.34.

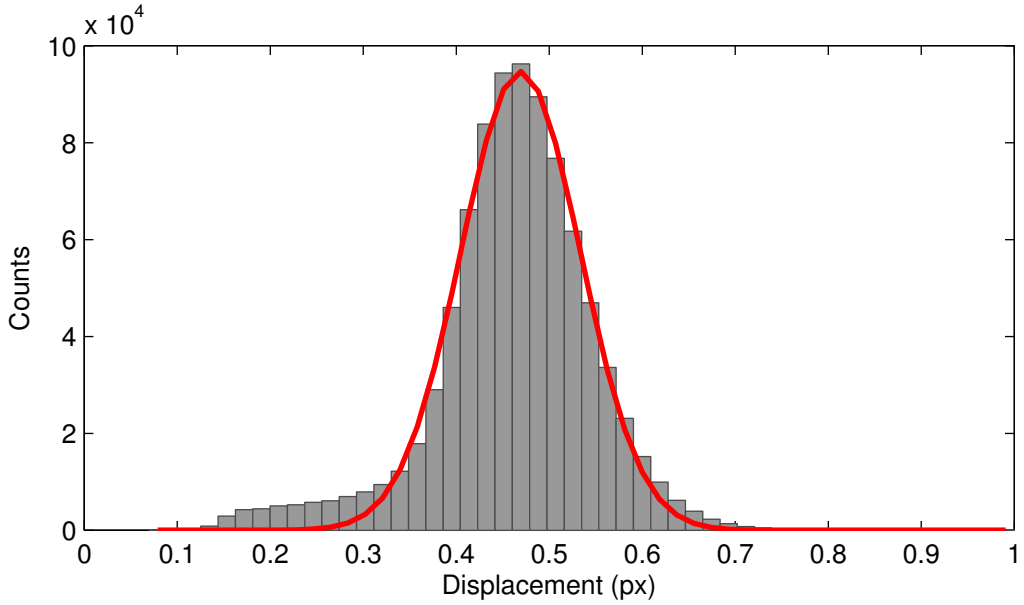


Figure 7.14: Histogram of the rendered intensity volume corresponding to Figure 7.12e. The 3σ threshold level for this case is 0.72.

7.4 Velocity Field of a Simulated Vortex

The previous two sections evaluated the performance of the PIV and rendering codes on uniform displacement fields to check for bias errors and peak-locking effects. With these

errors now quantified, a true 3-D velocity field can be tested to evaluate the algorithms for the case of displacements in all three directions. The Hill's spherical vortex is used for this purpose. Figure 7.15 shows the 3-D velocity field generated by the rendering and PIV code. Qualitatively, the large-scale vortical structure can be easily deduced with good resolution. Figures 7.16 and 7.17 present projective views of the 3-D velocity field to provide additional insight.

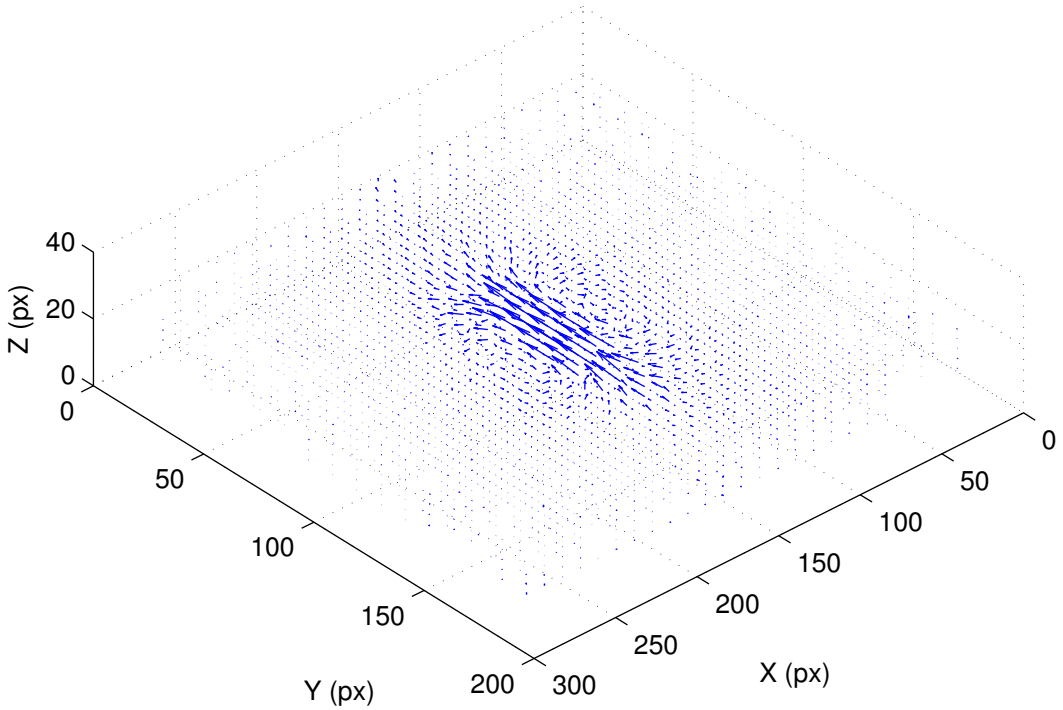


Figure 7.15: 3-D velocity field of a inviscid spherical vortex. All vectors plotted.

As in the previous cases, the histograms of displacement are shown to evaluate the effects of peak locking or any bias. Figures 7.18 through 7.20 show these histograms for x, y, and z-displacements, respectively. Most notable is the reduced effect of peak locking for z-displacements, which may simply be caused by the lower percentage of vectors with appreciable displacement in the z-direction.

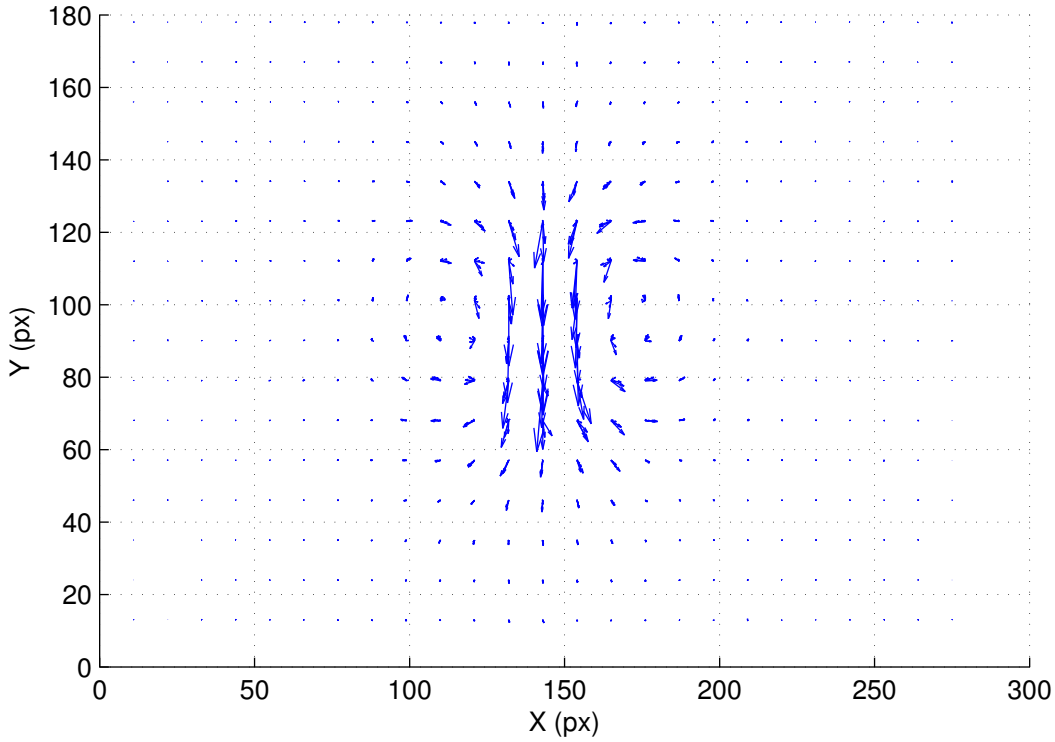


Figure 7.16: X-Y projection of the 3-D velocity field in Figure 7.15. All vectors plotted.

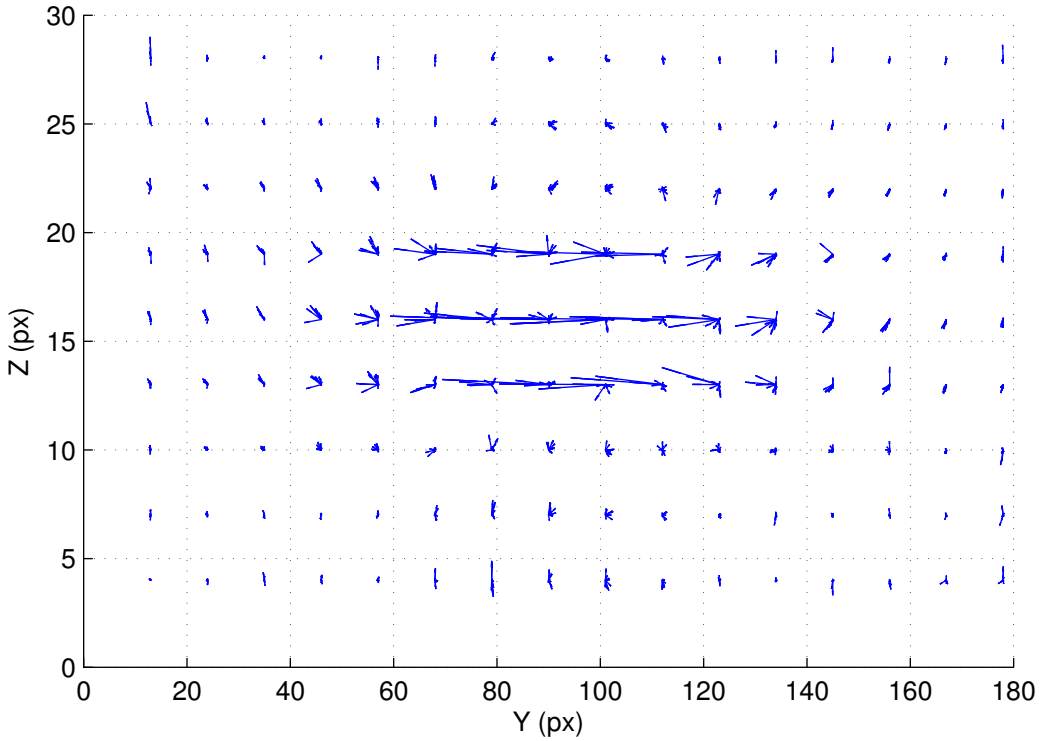


Figure 7.17: Y-Z projection of the 3-D velocity field in Figure 7.15. All vectors plotted.

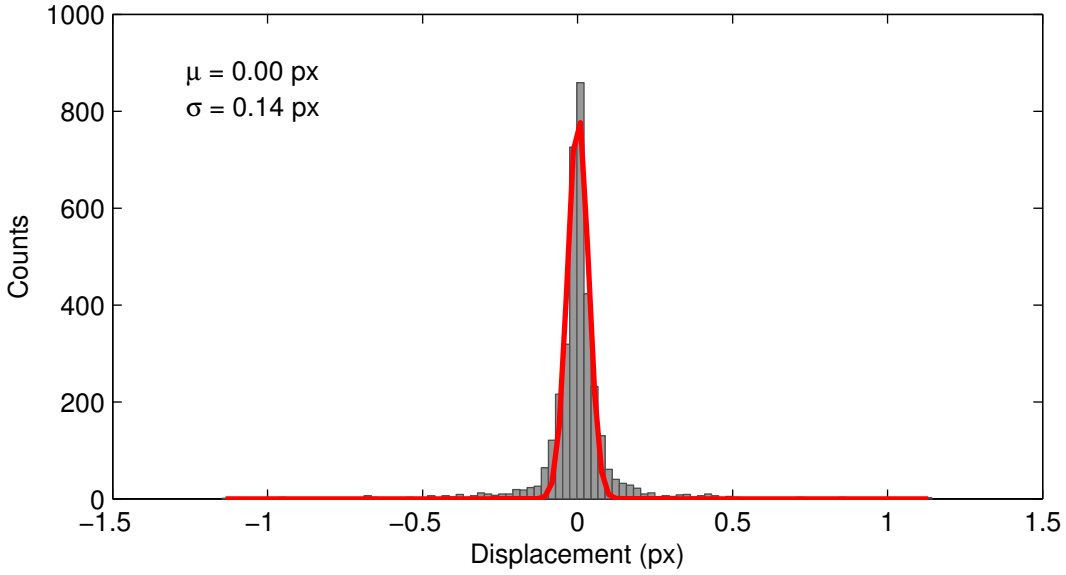


Figure 7.18: Histogram of x-displacements for the velocity field given in Figure 7.15

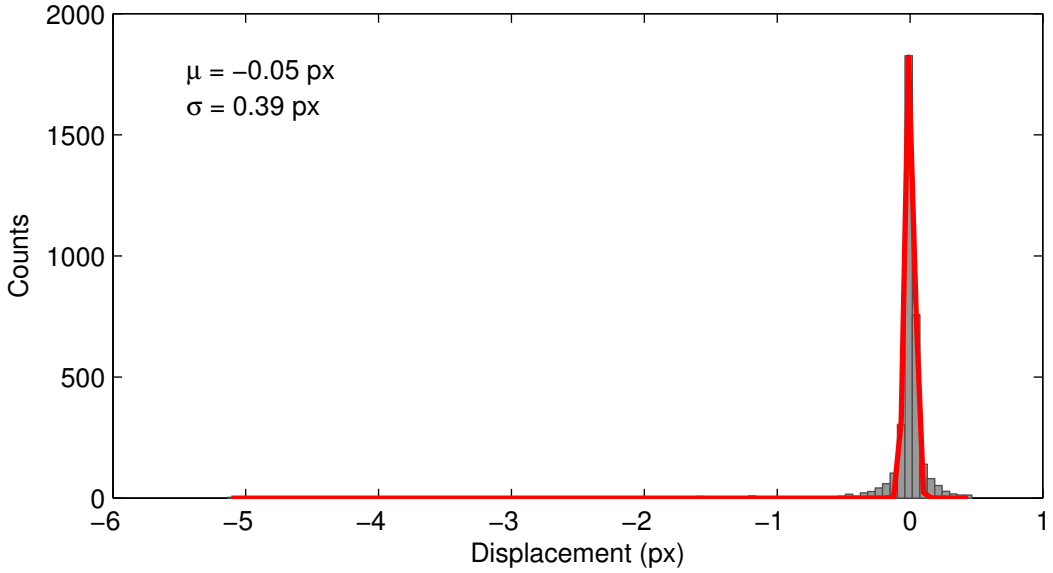


Figure 7.19: Histogram of y-displacements for the velocity field given in Figure 7.15

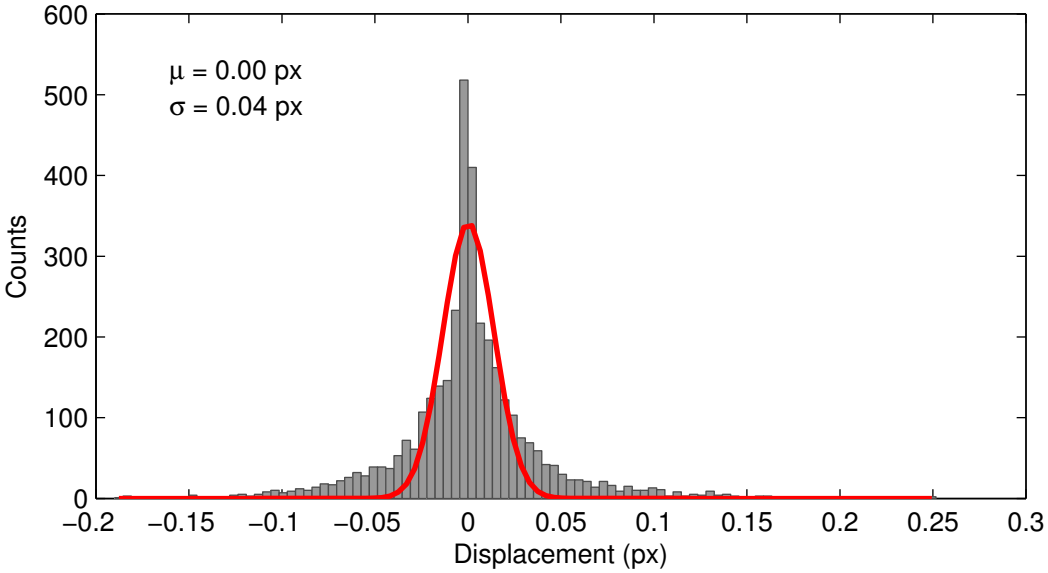


Figure 7.20: Histogram of z-displacements for the velocity field given in Figure 7.15

Chapter 8

Concluding Remarks

This thesis has detailed the development of a novel 3-D particle image velocimetry technique based on light field imaging by describing the concept of light fields and plenoptic photography, detailing the development of rendering codes, and finally integrating the rendering codes with a complete simulation of the camera and a 3-D PIV algorithm. Conclusions can be drawn from each of these.

The application of light field imaging to particle imaging was able to be explored in this thesis. For example, the depth and angular resolution can be traded through the design of the lenslet array. Increasing pitch of the lenslet increases the number of pixels covered by the lenslet and the number of unique focal planes available to be generated but also results in a reduction in the spatial resolution of each refocused image. The choice of lenslet size used in this work was primarily to evaluate the performance of a commercially available lenslet array; however, custom lenslet arrays can be manufactured. As an aside, the capability of trading depth resolution for spatial resolution could allow for innovative new ways to perform dual-plane PIV without multiple cameras and polarization filters. Another approach could be increase the depth resolution of the technique and use the camera with particle tracking velocimetry (PTV) algorithms. Advances in sensor technology may allow for smaller pixels to be reliably manufactured in the future and allow both the angular and spatial resolution to be sampled in much greater detail.

Other design parameters include the optical parameters such as magnification, the working distances s_i and s_o, and the f-number. In this thesis, it was demonstrated that increasing the depth resolution required the lenslet size (and thus circle of confusion) to be made smaller the camera to be operated at a magnification closer to 1 so the cone of rays generated at

different refocus planes do not intersect in an elongated volume. This finding has practical implications in that performing high resolution measurements will be easier within small measurement volumes. It also constrains the design space for lenslet arrays. In most cases, a lenslet with a low f-number is desired. Currently, the lowest f-number available for commercial order is $f/4$, but custom designs may allow this to be reduced to smaller numbers. This has the combined effect of potentially increasing the depth resolution of the camera as well as increasing the light gathering capacity. However, it must be kept in mind that the f-number of the camera lens is fixed by the effective f-number of the lenslets and is generally lower than the lenslet f-number. The f-number of the lenslets should thus not be designed in a manner where the main lens cannot properly match the required effective f-number.

The rendering codes developed in this thesis have successfully allowed for multiple planes to be refocused. However, the rendering process still requires additional work to fully understand the performance and limitations. A particular interest would be comparing rendering techniques used by other light field research groups, such as Fourier slice refocusing, to the current technique. Additionally, more basic questions persist and must be answered in future efforts. Namely, there is not a clear understanding of the number of focal planes that can be generated from a light field image and if generating a greater number of planes has any positive effect on the PIV interrogation. The current viewpoint taken in this work was originally given by Levoy, who uses the concept of uniqueness to restrict the number of generated focal planes to the number of pixels under a lenslet in any one direction. This perhaps could be strengthened for the case of particle imaging by developing a firmer theoretical background.

A notable deficiency of the rendering codes, specifically the intensity thresholding, is the inability to handle high particle number densities. This represents an area where tremendous progress can still be made in the technique by introducing more sophisticated 3-D deconvolution or limited angle tomographic schemes to eliminate out-of-focus particles. Improvements in this step of the rendering process have a direct effect in increasing the accuracy of PIV by allowing a larger number of particles to be placed within each interrogation volume and

allowing low intensity scatterers (which would be eliminated by the thresholding procedure) also to contribute to the correlation.

Finally, the PIV code developed in this thesis provided a reasonably accurate interrogation with adequate dynamic spatial and velocity range to evaluate the refocused images generated from the simulator and rendering codes. However, enormous developments in PIV techniques have occurred since the technique used in this thesis (discrete window offset) was proposed in 1999. The challenges in correctly implementing advanced correlation schemes ideally require dedicated efforts and suggest a collaborative effort should be formed with others in the 3-D PIV community such that the predominant focus in plenoptic PIV development is on the rendering codes and design studies.

In light of the difficulties expressed above, the plenoptic PIV concept has been demonstrated in this thesis to be certainly feasible using currently available hardware and software and should continue to be developed. The attractiveness of a single-camera solution for 3-D PIV cannot be discounted; however, rather than seeing plenoptic PIV as a replacement for tomographic or holographic PIV, perhaps it should be considered a unique complement to these techniques, whose simplicity could allow it to be used in laboratories worldwide.

Bibliography

[1] F. Pereira, M. Gharib, D. Dabiri, and D. Modarress, "Defocusing digital particle image velocimetry: a 3-component 3-dimensional DPIV measurement technique. Application to bubbly flows," *Experiments in Fluids*, vol. 29, pp. S78–S84, 2000.

[2] H. Meng, G. Pan, Y. Pu, and S. H. Woodward, "Holographic particle image velocimetry: from film to digital recording," *Measurement Science and Technology*, vol. 15, pp. 673–685, 2004.

[3] R. J. Adrian and J. Westerweel, *Particle Image Velocimetry*. Cambridge University Press, 2011.

[4] A. N. Kolmogorov, "Dissipation of energy in the locally isotropic turbulence," *C. R. Acad. Sci*, vol. 32, pp. 16–18, 1941.

[5] A. A. Townsend, *The Structure of Turbulent Shear Flow*. University Press, 1st ed., 1956.

[6] M. Samimy and M. P. Wernet, "Review of planar multiple-component velocimetry in high-speed flows," *AIAA Journal*, vol. 38, pp. 553–574, 2000.

[7] M. Raffel, C. Willert, S. Wereley, and J. Kompenhans, *Particle Image Velocimetry: A Practical Guide*. Springer, 2nd ed., 2007.

[8] G. S. Elliot and T. J. Beutner, "Molecular filter based planar doppler velocimetry," *Progress in Aerospace Sciences*, vol. 35, pp. 799–845, 1999.

[9] B. Thurow, N. Jiang, W. Lempert, and M. Samimy, "Development of megahertz-rate planar doppler velocimetry for high-speed flows," *AIAA Journal*, vol. 43, pp. 500–511, 2005.

[10] J. P. Crimaldi, "Planar laser induced fluorescence in aqueous flows," *Experiments in Fluids*, vol. 44, pp. 851–863, 2008.

[11] I. van Cruyningen, A. Lozano, and R. K. Hanson, "Quantitative imaging of concentration by planar laser-induced fluorescence," *Experiments in Fluids*, vol. 10, pp. 41–49, 2000.

[12] A. Cessou, U. Meier, and D. Stepowski, "Applications of planar laser induced fluorescence in turbulent reacting flows," *Measurement Science and Technology*, vol. 11, pp. 887–901, 2000.

[13] B. Stier and M. M. Koochesfahani, "Molecular tagging velocimetry (MTV) measurements in gas phase flows," *Experiments in Fluids*, vol. 26, pp. 297–304, 1999.

[14] H. Hu and M. M. Koochesfahani, "Molecular tagging velocimetry and thermometry and its application to the wake of a heated circular cylinder," *Measurement Science and Technology*, vol. 17, pp. 1269–1281, 2006.

[15] P. A. Davidson, *Turbulence: An introduction for scientists and engineers*. Oxford University Press, 2004.

[16] C. Brucker, "3-D scanning-particle-image-velocimetry: Technique and application to a spherical cap wake flow," *Appl. Sci. Res.*, vol. 56, pp. 157–179, 1996.

[17] C. Brucker, "3-D scanning PIV applied to an air flow in a motored engine using digital high-speed video," *Measurement Science Technology*, vol. 8, pp. 1480–1492, 1997.

[18] W. Zhang, R. Hain, and C. J. Kahler, "Scanning PIV investigation of the laminar separation bubble on a SD7003 airfoil," *Experiments in Fluids*, vol. 45, pp. 725–743, 2008.

[19] C. E. Willert and M. Gharib, "Three-dimensional particle imaging with a single camera," *Experiments in Fluids*, vol. 12, pp. 353–358, 1992.

[20] D. Lin, N. C. Angarita-Jaimes, S. Chen, A. H. Greenaway, C. E. Towers, and D. P. Towers, "Three-dimensional particle imaging by defocusing method with an annular aperture," *Optics Letters*, vol. 33, pp. 905–907, 2008.

[21] F. Pereira and M. Gharib, "Defocusing digital particle imaging velocimetry and the three-dimensional characterization of two-phase flows," *Measurement Science and Technology*, vol. 13, pp. 683–694, 2002.

[22] M. Gharib, F. Pereira, D. Dabiri, J. R. Hove, and D. Modarress, "Quantitative flow visualization: Toward a comprehensive flow diagnostic tool," *Integrative and Comparative Biology*, vol. 42, pp. 964–970, 2002.

[23] G. E. Elsinga, F. Scarano, B. Wieneke, and B. W. van Oudheusden, "Tomographic particle image velocimetry," *Experiments in Fluids*, vol. 41, pp. 933–947, 2006.

[24] G. E. Elsinga, J. Westerweel, F. Scarano, and M. Novara, "On the velocity of ghost particles and the bias errors in tomographic-PIV," *Experiments in Fluids*, 2010. Published Online.

[25] R. A. Humble, G. E. Elsinga, F. Scarano, and B. W. van Oudheusden, "Three-dimensional instantaneous structure of a shock wave/turbulent boundary layer interaction," *Journal of Fluid Mechanics*, vol. 622, pp. 33–62, 2009.

[26] F. Scarano and C. Poelma, "Three-dimensional vorticity patterns of cylinder wakes," *Experiments in Fluids*, vol. 47, pp. 69–83, 2009.

[27] D. Violato, P. Moore, and F. Scarano, "Lagrangian and eulerian pressure field evaluation of rod-airfoil flow from time-resolved tomographic PIV," *Experiments in Fluids*, 2010. Published Online.

[28] J. Sheng, E. Malkiel, and J. Katz, "Using digital holographic microscopy for simultaneous measurements of 3D near wall velocity and wall shear stress in a turbulent boundary layer," *Experiments in Fluids*, vol. 45, pp. 1023–1035, 2008.

[29] M. P. Arroyo and K. D. Hinsch, *Recent Developments of PIV towards 3D Measurements*, pp. 127–154. Springer, 2008.

[30] A. Svizher and J. Cohen, "Holographic particle image velocimetry system for measurement of hairpin vortices in air channel flow," *Experiments in Fluids*, vol. 40, pp. 708–722, 2006.

[31] B. Tao, J. Katz, and C. Meneveau, "Geometry and scale relationships in high reynolds number turbulence determined from three-dimensional holographic velocimetry," *Physics of Fluids*, vol. 12, pp. 941–944, 2000.

[32] J. Belden, T. T. Truscott, M. C. Axiak, and A. M. Techet, "Three-dimensional synthetic aperture particle image velocimetry," *Measurement Science and Technology*, vol. 21, pp. 1–21, 2010.

[33] V. Vaish, G. Garg, E. Talvala, E. Antunez, B. Wilburn, M. Horowitz, and M. Levoy, "Synthetic aperture focusing using a shear-warp factorization of the viewing transform," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 3, 2005.

[34] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*, pp. 3–20, MIT Press, 1991.

[35] E. H. Adelson and J. Y. A. Wang, "Single lens stereo with a plenoptic camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 99–106, 1992.

[36] R. Ng, M. Levoy, M. Bredif, G. Duval, M. Horowitz, and P. Hanrahan, "Light field photography with a hand-held plenoptic camera," tech. rep., Stanford University, 2005.

[37] M. Levoy, N. R, A. Adams, M. Footer, and M. Horowitz, "Light field microscopy," *ACM Transactions on Graphics*, vol. 25, pp. 924–934, 2006.

[38] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pp. 31–42, 1996.

[39] M. Levoy, "Light fields and computational imaging," *Computer*, vol. 39, pp. 46–55, 2006.

[40] B. Wilburn, N. Joshi, V. Vaish, E. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," *ACM Transactions on Graphics*, vol. 24, pp. 765–776, 2005.

[41] M. Levoy, Z. Zhang, and I. McDowall, "Recording and controlling the 4D light field in a microscope," *Journal of Microscopy*, vol. 235, pp. 144–162, 2009.

[42] A. Lumsdaine and T. Georgiev, "Full resolution lightfield rendering," tech. rep., Adobe Systems, 2009.

[43] A. Lumsdaine and T. Georgiev, "The focused plenoptic camera," ICCP 2009, 2009.

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

[45] T. Georgeiv and C. Intwala, "Light field camera design for integral view photography," tech. rep., Adobe Systems, 2003.

[46] L. Ahrenberg and M. Magnor, "Light field rendering using matrix optics," in *Proceedings of WSCG 2006*, 2006.

[47] W. J. Smith, *Modern Optical Engineering*. McGraw-Hill, 4th ed., 2007.

[48] Kodak Company, "Kodak KAI-16000 Image Sensor Device Performance Specification," 2010.

[49] S. I. Green, *Fluid vortices*. Springer, 1995.

[50] R. Ng, "Fourier slice photography," *ACM Transactions on Graphics*, vol. 24, pp. 735–744, 2005.

[51] J. Sibarita, "Deconvolution microscopy," *Advanced Biochemical Engineering and Biotechnology*, vol. 95, pp. 201–243, 2005.

[52] J. G. McNally, T. Karpova, J. Cooper, and J. A. Conchello, "Three-dimensional imaging by deconvolution microscopy," *Methods*, vol. 19, pp. 373–385, 1999.

[53] J. S. Park and K. D. Kihm, "Three-dimensional micro-ptv using deconvolution microscopy," *Experiments in Fluids*, vol. 40, pp. 491–499, 2006.

[54] R. J. Adrian, "Twenty years of particle image velocimetry," *Experiments in Fluids*, vol. 39, pp. 159–169, 2005.

[55] C. E. Willert and M. Gharib, "Digital particle image velocimetry," *Experiments in Fluids*, vol. 10, pp. 181–193, 1991.

[56] J. Westerweel, D. Dabiri, and M. Gharib, "The effect if a discrete window offset on the accuracy of cross-correlation analysis of digital piv recordings," *Experiments in Fluids*, vol. 23, pp. 20–28, 1997.

[57] F. Scarano and M. L. Riethmuller, "Iterative multigrid approach in piv image processing with discrete window offset," *Experiments in Fluids*, vol. 26, pp. 513–523, 1999.

[58] F. Scarano and M. L. Riethmuller, "Advances in iterative multigrid piv image processing," *Experiments in Fluids*, vol. 29, pp. S51–S60, 2000.

[59] F. Scarano and M. L. Riethmuller, "Iterative image deformation methods in piv," *Measurement Science and Technology*, vol. 13, pp. R1–R19, 2002.

[60] S. T. Wereley and C. D. Meinhart, "Second-order accurate particle image velocimetry," *Experiments in Fluids*, vol. 31, pp. 258–268, 2001.

[61] R. Theunissen, F. Scarano, and M. L. Riethmuller, "Spatially adaptive piv interrogation based on data ensemble," *Experiments in Fluids*, vol. 48, pp. 875–887, 2010.

[62] T. Liu and S. L., "Fluid flow and optical flow," *Journal of Fluid Mechanics*, vol. 614, pp. 253–291, 2008.

[63] M. Stanislas, K. Okamoto, C. J. Kähler, J. Westerweel, and F. Scarano, "Main results of the third international piv challenge," *Experiments in Fluids*, vol. 45, pp. 27–71, 2008.

[64] J. Westerweel and F. Scarano, "Universal outlier detection for piv data," *Experiments in Fluids*, vol. 39, pp. 1096–1100, 2005.

Appendix A: Fortran 2-D Simulation Code

```
program plenoptic_simulator

! Bring in random number module.
! http://www.netlib.org/random/
use random

implicit none

real :: f_l, p_l, p_p, f_m, FOV_x, FOV_y, M, s_i, s_o, fnum_l, fnum_m, p_m
real :: sigma_m, sigma_l, o_x, o_y, p_ccd_x, p_ccd_y, coll_angle
real :: phi_max, phi_min, phi_total, theta_max, theta_min, theta_total, phi, theta
real :: randa, randb, sx, sy

integer :: n_p_x, n_p_y, n_rays, n_l_x, n_l_y, npts, nrays_phi, nrays_theta
integer :: x_ccd, y_ccd, nsx, nsy
integer :: i, j

real, allocatable, dimension(:,:) :: image, image_2d
real, allocatable, dimension(:) :: dx, dy, dz, intensity
real, dimension(4,4) :: T1, T2, T3, L1, L2, A1
real, dimension(4) :: ray, ray2, ray3, ray4, ray5

character(40) :: optics_filename, field_filename, output_filename, t_output_filename

! Temporary Variables
real :: tmp_n_p_y, tmp_n_p_x, tmp_n_l_x, tmp_n_l_y, tmp_n_rays
character(17) :: dump

write(*,*) 'Plenoptic Image Simulator'
write(*,*) 'Kyle Lynch (lynchkp@auburn.edu)'
write(*,*)

call get_command_argument(1,field_filename)
call get_command_argument(2,output_filename)

! Load in optics settings from file.
optics_filename = 'optics2.txt'
open(unit=8,file=optics_filename,status='old')
read(8,100) dump, f_l
read(8,100) dump, p_l
read(8,100) dump, p_p
read(8,100) dump, tmp_n_p_x
read(8,100) dump, tmp_n_p_y
read(8,100) dump, f_m
read(8,100) dump, FOV_x
read(8,100) dump, FOV_y
read(8,100) dump, M
read(8,100) dump, s_i
read(8,100) dump, s_o
read(8,100) dump, fnum_l
read(8,100) dump, fnum_m
read(8,100) dump, p_m
read(8,100) dump, sigma_m
read(8,100) dump, sigma_l
read(8,100) dump, tmp_n_l_x
read(8,100) dump, tmp_n_l_y
read(8,100) dump, o_x
```

```
   read(8,100) dump, o_y
59 read(8,100) dump, tmp_n_rays
   n_p_x = nint(tmp_n_p_x)
61 n_p_y = nint(tmp_n_p_y)
   n_l_x = nint(tmp_n_l_x)
63 n_l_y = nint(tmp_n_l_y)
   n_rays = nint(tmp_n_rays)
65 allocate(image(n_p_y,n_p_x))
   allocate(image_2d(n_p_y,n_p_x))
67 100 format(A,F14.6)
   close(8)
69
   ! Load in particle field
71 !field_filename = 'field.txt'
   open(unit=9,file=field_filename)
73 read(9,'(I14)') npts
   allocate(dx(npts))
75 allocate(dy(npts))
   allocate(dz(npts))
77 allocate(intensity(npts))
   do i = 1,npts
79         read(9,'(F14.6,1X,F14.6,1X,F14.6,1X,F14.6)') dx(i), dy(i), dz(i), intensity(i)
   enddo
81 close(9)

83 !output_filename = 'image.txt'
   open(unit=10,file=output_filename)
85
   t_output_filename = 'image2D.txt'
87 open(unit=11,file=t_output_filename)

89 !-------------------------------------------------------------------------
   !-------------------------------------------------------------------------
91 !
   !        Begin Image Ray Tracing
93 !
   !-------------------------------------------------------------------------
95 !-------------------------------------------------------------------------

97 ! Initialize image arrays with zeros.
   do j = 1,n_p_y
99 do i = 1,n_p_x
         image(j,i) = 0.0
101      image_2d(j,i) = 0.0
   enddo
103 enddo

105 ! Lenslet -> CCD Ray Transfer Matrix
   T3(1,:) = (/1.,0.,f_l,0./)
107 T3(2,:) = (/0.,1.,0.,f_l/)
   T3(3,:) = (/0.,0.,1.,0./)
109 T3(4,:) = (/0.,0.,0.,1./)

111 ! Main Lens Ray Transfer Matrix
   L1(1,:) = (/1.,0.,0.,0./)
113 L1(2,:) = (/0.,1.,0.,0./)
   L1(3,:) = (/-1./f_m,0.,1.,0./)
115 L1(4,:) = (/0.,-1./f_m,0.,1./)

117 ! Lenslet Ray Transfer Matrix
   L2(1,:) = (/1.,0.,0.,0./)
119 L2(2,:) = (/0.,1.,0.,0./)
   L2(3,:) = (/-1./f_l,0.,1.,0./)
121 L2(4,:) = (/0.,-1./f_l,0.,1./)

123 ! Main Lens -> Lenslets Ray Transfer Matrix
   T2(1,:) = (/1.,0.,s_i,0./)
125 T2(2,:) = (/0.,1.,0.,s_i/)
```

```
   T2(3,:) = (/0.,0.,1.,0./)
127 T2(4,:) = (/0.,0.,0.,1./)

129 A1 = matmul(T2,L1)

131 p_ccd_x = n_p_x*p_p
   p_ccd_y = n_p_y*p_p
133
   coll_angle = 2 * atan2(p_m/2, s_o)
135
   do i = 1,npts
137
         write(*,*) i
139
         T1(1,:) = (/1.,0.,s_o+dz(i),0./)
141      T1(2,:) = (/0.,1.,0.,s_o+dz(i)/)
         T1(3,:) = (/0.,0.,1.,0./)
143      T1(4,:) = (/0.,0.,0.,1./)

145      phi_max = atan2(p_m/2-dy(i),s_o+dz(i));
         phi_min = atan2(-p_m/2-dy(i),s_o+dz(i));
147      phi_total = phi_max - phi_min;

149      theta_max = atan2(p_m/2-dx(i),s_o+dz(i));
         theta_min = atan2(-p_m/2-dx(i),s_o+dz(i));
151      theta_total = theta_max - theta_min;

153      nrays_phi = nint((phi_total / coll_angle) * sqrt(real(n_rays)));
         nrays_theta = nint((theta_total / coll_angle) * sqrt(real(n_rays)));
155
         do j = 1,(nrays_phi*nrays_theta)
157
               call random_number(randa)
159            call random_number(randb)

161            theta = theta_min + (randa*(theta_max-theta_min))
               phi = phi_min + (randb*(phi_max-phi_min))
163            ray = (/dx(i),dy(i),theta,phi/)

165            ! Propogate ray from point to the main lens
               ray2(1) = T1(1,1)*ray(1) + T1(1,2)*ray(2) + T1(1,3)*ray(3) + T1(1,4)*ray(4);
167            ray2(2) = T1(2,1)*ray(1) + T1(2,2)*ray(2) + T1(2,3)*ray(3) + T1(2,4)*ray(4);
               ray2(3) = T1(3,1)*ray(1) + T1(3,2)*ray(2) + T1(3,3)*ray(3) + T1(3,4)*ray(4);
169            ray2(4) = T1(4,1)*ray(1) + T1(4,2)*ray(2) + T1(4,3)*ray(3) + T1(4,4)*ray(4);

171            if (sqrt(ray2(1)**2 + ray2(2)**2) < (p_m/2)) then

173               ! Propogate ray through the main lens and to the lenslet array.
                  ray3(1) = A1(1,1)*ray2(1) + A1(1,2)*ray2(2) + &
175                       A1(1,3)*ray2(3) + A1(1,4)*ray2(4);
                  ray3(2) = A1(2,1)*ray2(1) + A1(2,2)*ray2(2) + &
177                       A1(2,3)*ray2(3) + A1(2,4)*ray2(4);
                  ray3(3) = A1(3,1)*ray2(1) + A1(3,2)*ray2(2) + &
179                       A1(3,3)*ray2(3) + A1(3,4)*ray2(4);
                  ray3(4) = A1(4,1)*ray2(1) + A1(4,2)*ray2(2) + &
181                       A1(4,3)*ray2(3) + A1(4,4)*ray2(4);

183               ! DIFFRACTION AT LENSLET
                  ray3(1) = ray3(1) + sigma_l*random_normal();
185               ray3(2) = ray3(2) + sigma_l*random_normal();

187               ! 2D Image Capture.
                  x_ccd = nint( ( ray3(1) + (p_ccd_x+p_p)/2 ) / p_p );
189               y_ccd = nint( ( ray3(2) + (p_ccd_y+p_p)/2 ) / p_p );
                  if (x_ccd>0 .and. y_ccd>0 .and. x_ccd<(n_p_x+1) .and. y_ccd<(n_p_y+1)) then
191                     image_2d(y_ccd,x_ccd) = image_2d(y_ccd,x_ccd) + intensity(i);
                  endif
193
```

```
                        ! Find lenslet
195                     nsx = nint( ( ray3(1) + p_ccd_x/2 - o_x - p_l/2 ) / p_l );
                        nsy = nint( ( ray3(2) + p_ccd_y/2 - o_y - p_l/2 ) / p_l );
197                     sx = nsx * p_l + o_x + p_l/2 - p_ccd_x/2;
                        sy = nsy * p_l + o_y + p_l/2 - p_ccd_y/2;
199
                        ! Propogate ray through lenslet
201                     ray4(1) = (L2(1,1)*ray3(1) + L2(1,2)*ray3(2) + &
                                L2(1,3)*ray3(3) + L2(1,4)*ray3(4)) + 0.;
203                     ray4(2) = (L2(2,1)*ray3(1) + L2(2,2)*ray3(2) + &
                                L2(2,3)*ray3(3) + L2(2,4)*ray3(4)) + 0.;
205                     ray4(3) = (L2(3,1)*ray3(1) + L2(3,2)*ray3(2) + &
                                L2(3,3)*ray3(3) + L2(3,4)*ray3(4)) + sx/f_l;
207                     ray4(4) = (L2(4,1)*ray3(1) + L2(4,2)*ray3(2) + &
                                L2(4,3)*ray3(3) + L2(4,4)*ray3(4)) + sy/f_l;
209
                        ! Propogate ray to the CCD
211                     ray5(1) = T3(1,1)*ray4(1) + T3(1,2)*ray4(2) + &
                                T3(1,3)*ray4(3) + T3(1,4)*ray4(4);
213                     ray5(2) = T3(2,1)*ray4(1) + T3(2,2)*ray4(2) + &
                                T3(2,3)*ray4(3) + T3(2,4)*ray4(4);
215                     ray5(3) = T3(3,1)*ray4(1) + T3(3,2)*ray4(2) + &
                                T3(3,3)*ray4(3) + T3(3,4)*ray4(4);
217                     ray5(4) = T3(4,1)*ray4(1) + T3(4,2)*ray4(2) + &
                                T3(4,3)*ray4(3) + T3(4,4)*ray4(4);
219
                        ! DIFFRACTION AT PIXEL
221                     ray5(1) = ray5(1) + sigma_m*random_normal();
                        ray5(2) = ray5(2) + sigma_m*random_normal();
223
                        ! Determine which pixel the ray will strike.
225                     x_ccd = nint( ( ray5(1) + (p_ccd_x+p_p)/2 ) / p_p );
                        y_ccd = nint( ( ray5(2) + (p_ccd_y+p_p)/2 ) / p_p );
227
                        ! Ensure it falls within sensor and record the signal
229                     if (x_ccd>0 .and. y_ccd>0 .and. x_ccd<(n_p_x+1) .and. y_ccd<(n_p_y+1)) then
                                image(y_ccd,x_ccd) = image(y_ccd,x_ccd) + intensity(i);
231                     endif

233                 endif

235         enddo

237 enddo

239 ! Write out image data
    write(10,'(I5)') n_p_x
241 write(10,'(I5)') n_p_y
    do j = 1,n_p_y
243         do i = 1,n_p_x
                write(10,'(I6)') nint( image(j,i) )
245         enddo
    enddo
247 close(10)

249 write(*,*) 'Successfully exported image data to ', output_filename

251 ! Write out image data
    write(11,'(I5)') n_p_x
253 write(11,'(I5)') n_p_y
    do j = 1,n_p_y
255         do i = 1,n_p_x
                write(11,'(F8.1)') image_2d(j,i)
257         enddo
    enddo
259 close(11)

261 write(*,*) 'Successfully exported image data to ', t_output_filename
```

```
263 end program plenoptic_simulator
```

Appendix B: Matlab Simulator Scripts

```
function particle_gen(filename,particle_density,x_span,y_span,z_span,...
    i_min,i_max,type)
%PARTICLE_GEN_2 Generate Particle Field
%   [] = PARTICLE_GEN_2(FILENAME,PARTICLE_DENSITY,X_SPAN,Y_SPAN,Z_SPAN, ...
%           I_MIN,I_MAX)
%   Outputs a text file containing the number of points (particles) and the
%   x, y, and z coordinates for the particle. Intensity coefficents are
%   also output.
%
%   Inputs:
%   FILENAME is the file to output particle positions.
%   PARTICLE_DENSITY is the density of particles, in number per mm^3.
%   X_SPAN is the total size of the x-coordinate of the volume, which will
%   be centered with respect to the optical axis.
%   Y_SPAN is the total size of the y-coordinate of the volume, which will
%   be centered with respect to the optical axis.
%   Z_SPAN is the total size of the z-coordinate of the volume, which will
%   be centered with respect to the optical axis.
%   I_MIN is the minimum particle intensity coefficent.
%   I_MIN is the maximum particle intensity coefficent.
%
%   Outputs:
%   None.
%
%   Kyle Lynch (lynchkp@auburn.edu) January 30, 2011

% Define the ranges.
x_min = -x_span + x_span/2;
y_min = -y_span + y_span/2;
z_min = -z_span + z_span/2;
x_max = x_span - x_span/2;
y_max = y_span - y_span/2;
z_max = z_span - z_span/2;

if strcmp(type,'volumetric')
    volume = (x_max-x_min)*(y_max-y_min)*(z_max-z_min);
    n_pts = round(particle_density*volume);
elseif strcmp(type,'pinhole');
    n_pts = 750000;
    z_max = 0;
    z_min = 0;
elseif strcmp(type,'perpixel')
    npx = 4872*3248;
    n_pts = round(particle_density*npx);
else
    fprintf('Improper type. Either "normal" or "pinhole."\n');
    return;
end

% Generate points
xval = x_min + (x_max-x_min).*rand(n_pts,1);
yval = y_min + (y_max-y_min).*rand(n_pts,1);
zval = z_min + (z_max-z_min).*rand(n_pts,1);
ival = i_min + (i_max-i_min).*rand(n_pts,1);

% Write the file
fprintf('Number of Particles: %d\n',n_pts)
fprintf('Exporting to %s\n',filename);
```

```
fid = fopen(filename,'w+');
fprintf(fid,'%14d\n',n_pts);
for i = 1:n_pts
    fprintf(fid,'%14.6f %14.6f %14.6f %14.6f\n',xval(i),yval(i),zval(i),ival(i));
end
fclose(fid);

%% Plotting Functions
% if z_max-z_min > 0
%     figure
%     plot3(zval,xval,yval,'.')
%     xlabel('Z')
%     ylabel('X')
%     zlabel('Y')
%     view(40,10)
%     axis equal
%     axis ij
%     axis([z_min z_max x_min x_max y_min y_max])
%     title('3D Particle Locations')
% end
%
% figure
% plot(xval,yval,'.')
% xlabel('X')
% ylabel('Y')
% axis equal
% axis xy
% axis([x_min x_max y_min y_max])
% title('2D Particle Projections')

function optics_gen(filename, f_l, p_l, p_p, n_p_x, n_p_y, f_m, M, n_rays, type)
%OPTICS_GEN_2 Generate Optical Parameters
%   [] = OPTICS_GEN_2(FILENAME, F_L, P_L, P_P, N_P_X, N_P_Y, F_M, M, N_RAYS)
%   Outputs a text file containing the optical parameters of the system.
%
%   Inputs:
%   FILENAME is the file to output the optical settings.
%   F_L is the focal length of the lenslets, in mm.
%   P_L is the pitch of the lenslets, in mm.
%   P_P is the pitch of the pixels, in mm.
%   N_P_X is the number of pixels horizontally.
%   N_P_Y is the number of pixels vertically.
%   F_M is the focal length of the main lens.
%   M is the magnification of the optical system.
%   N_RAYS is the number of rays to be generated for each point.
%
%   Outputs:
%   None.
%
%   Kyle Lynch (lynchkp@auburn.edu) January 30, 2011

% Derived parameters
FOV_y = -n_p_y*p_p / M;
FOV_x = -n_p_x*p_p / M;
% M = -(p_p*n_p_y / FOV_y);
s_i = f_m*(1-M);
s_o = -s_i / M;
fnum_l = f_l / p_l;

if strcmp(type,'normal')
    fnum_m = fnum_l / (1-M);
elseif strcmp(type,'pinhole')
    fnum_m = 16;
else
    fprintf('Improper type. Either "normal" or "pinhole."\n');
    return;
end

```

```
p_m = f_m / fnum_m;
p_ccd_x = n_p_x*p_p;
p_ccd_y = n_p_y*p_p;

fnum_scale = [0.5 0.7 1.0 1.4 2 2.8 4 5.6 8 11 16 22 32 45 64 90 128];
fnum_m = min(fnum_scale((fnum_scale-fnum_m)>=0));

% Diffraction Parameters. Adrian and Westerweel use positive
% magnifications.
lam = 532E-9 * 1E3; % convert to mm
betasq = 3.67;
M1 = s_i/s_o;
ds_m = 2.44 * (1 + M1) * fnum_m * lam;
sigma_m = sqrt((ds_m^2) / (8*betasq));
diffract_m_percent = (ds_m / p_l) * 100;
M2 = f_l / s_i;
ds_l = 2.44 * (1 + M2) * fnum_l * lam;
sigma_l = sqrt((ds_l^2) / (8*betasq));
diffract_l_percent = (ds_l / p_p) * 100;
% keyboard

fprintf('CCD Resolution: %d px X, %d px Y\n',n_p_x,n_p_y)
fprintf('Pixel Pitch: %f mm X, %f mm Y\n',p_p,p_p);
fprintf('Physical Size of CCD: %f mm X, %f mm Y\n\n',p_ccd_x,p_ccd_y);

fprintf('Main Lens Diffraction Spot Size: %f\n',ds_m);
fprintf('Main Lens Diffraction Std. Dev: %f\n',sigma_m);
fprintf('Main Lens Diffraction Percentage of Lenslet Pitch: %f\n', ...
    diffract_m_percent);
fprintf('Lenset Diffraction Spot Size: %f\n',ds_l);
fprintf('Main Lens Diffraction Std. Dev: %f\n',sigma_l);
fprintf('Main Lens Diffraction Percentage of Lenslet Pitch: %f\n\n', ...
    diffract_l_percent);

% keyboard;

% Determine the grid offset to center the grid within the image.
n_l_y = 0;
for (y = 0:p_l:(p_ccd_y))
    n_l_x = 0;
    for (x = 0:p_l:(p_ccd_x))
        n_l_x = n_l_x + 1;
    end
    n_l_y = n_l_y + 1;
end
n_l = n_l_y*n_l_x;
o_x = ((p_ccd_x - (x+p_l)) / 2);
o_y = ((p_ccd_y - (y+p_l)) / 2);
fprintf('Lenslet Pitch: %f X, %f Y\n',p_l,p_l);
fprintf('Lenslet Offset: %f X, %f Y\n',o_x,o_y);
fprintf('Number of Lenslets: %d X, %d Y\n',n_l_x,n_l_y)
fprintf('Total Number of Lenslets: %d\n',n_l)
fprintf('Focal Length of Lenslets: %f mm\n',f_l);

% Step 8: Export all data.
fprintf('Exporting to %s\n',filename);
fid = fopen(filename,'w');
fprintf(fid,'%15s = %14.6f\n','f_l',f_l);          % 1
fprintf(fid,'%15s = %14.6f\n','p_l',p_l);          % 2
fprintf(fid,'%15s = %14.6f\n','p_p',p_p);          % 3
fprintf(fid,'%15s = %14.6f\n','n_p_x',n_p_x);      % 4
fprintf(fid,'%15s = %14.6f\n','n_p_y',n_p_y);      % 5
fprintf(fid,'%15s = %14.6f\n','f_m',f_m);          % 6
fprintf(fid,'%15s = %14.6f\n','FOV_x',FOV_x);      % 7
fprintf(fid,'%15s = %14.6f\n','FOV_y',FOV_y);      % 8
fprintf(fid,'%15s = %14.6f\n','M',M);              % 9
fprintf(fid,'%15s = %14.6f\n','s_i',s_i);          % 10
fprintf(fid,'%15s = %14.6f\n','s_o',s_o);          % 11
```

```
    fprintf(fid,'%15s = %14.6f\n','fnum_l',fnum_l);      % 12
108 fprintf(fid,'%15s = %14.6f\n','fnum_m',fnum_m);          % 13
    fprintf(fid,'%15s = %14.6f\n','p_m',p_m);              % 14
110 fprintf(fid,'%15s = %14.6f\n','sigma_m',sigma_m);        % 15
    fprintf(fid,'%15s = %14.6f\n','sigma_l',sigma_l);        % 16
112 fprintf(fid,'%15s = %14.6f\n','n_l_x',n_l_x);          % 17
    fprintf(fid,'%15s = %14.6f\n','n_l_y',n_l_y);          % 18
114 fprintf(fid,'%15s = %14.6f\n','o_x',o_x);              % 19
    fprintf(fid,'%15s = %14.6f\n','o_y',o_y);              % 20
116 fprintf(fid,'%15s = %14.6f\n','n_rays',n_rays);        % 21

118 return;

120 % Plotting Functions
    lenslet_loc = zeros(n_l_y,n_l_x,2);
122 for (y = 1:1:n_l_y)
        for (x = 1:1:n_l_x)
124         lenslet_loc(y,x,1) = (x-1)*p_l + o_x + p_l/2;
            lenslet_loc(y,x,2) = (y-1)*p_l + o_y + p_l/2;
126     end
    end
128 lenslet_loc(:,:,1) = lenslet_loc(:,:,1)-(p_ccd_x/2);
    lenslet_loc(:,:,2) = lenslet_loc(:,:,2)-(p_ccd_y/2);
130
    h = figure('Position',[100 100 650 315]);
132 subplot(121)
    hold on
134 line([-p_ccd_x/2 p_ccd_x/2],[p_ccd_y/2 p_ccd_y/2])
    line([p_ccd_x/2 p_ccd_x/2],[p_ccd_y/2 -p_ccd_y/2])
136 line([p_ccd_x/2 -p_ccd_x/2],[-p_ccd_y/2 -p_ccd_y/2])
    line([-p_ccd_x/2 -p_ccd_x/2],[-p_ccd_y/2 p_ccd_y/2])
138 line([0 0],[-p_ccd_y/2 p_ccd_y/2],'Color','k')
    line([-p_ccd_x/2 p_ccd_x/2],[0 0],'Color','k')
140 plot(lenslet_loc(:,:,1),lenslet_loc(:,:,2),'ro','MarkerSize',5);
    xlabel('X (mm)'), ylabel('Y (mm)')
142 axis image

144 subplot(122)
    hold on
146 line([-p_ccd_x/2 p_ccd_x/2],[p_ccd_y/2 p_ccd_y/2])
    line([p_ccd_x/2 p_ccd_x/2],[p_ccd_y/2 -p_ccd_y/2])
148 line([p_ccd_x/2 -p_ccd_x/2],[-p_ccd_y/2 -p_ccd_y/2])
    line([-p_ccd_x/2 -p_ccd_x/2],[-p_ccd_y/2 p_ccd_y/2])
150 line([0 0],[-p_ccd_y/2 p_ccd_y/2],'Color','k')
    line([-p_ccd_x/2 p_ccd_x/2],[0 0],'Color','k')
152 plot(lenslet_loc(:,:,1),lenslet_loc(:,:,2),'ro','MarkerSize',5);
    xlabel('X (mm)'), ylabel('Y (mm)')
154 axis image

156 % Depth of field parameters
    DN = (s_o * f_m^2) / (f_m^2 + fnum_m*p_l*(s_o-f_m));
158 DF = (s_o * f_m^2) / (f_m^2 - fnum_m*p_l*(s_o-f_m));
    DOF = abs(DF - DN)

    function sim_displacement_gen(filename,output_filename,func,dt,nsteps)
 2 %SIM_DISPLACEMENT_GEN Generate Particle Displacements
    %   [] = SIM_DISPLACEMENT_GEN(FILENAME,OUTPUT_FILENAME,DT,NSTEPS) takes an
 4 %   input particle field file, applies a velocity field to the particle
    %   locations over a number of steps, and saves the new particle locations
 6 %   to a file.
    %
 8 %   Inputs:
    %   FILENAME is the file containing the initial particle positions.
10 %   OUTPUT_FILENAME is the file to output final particle positons.
    %   FUNC is the velocity field function to be used.
12 %   DT is the duration of each timestep, in seconds.
    %   NSTEPS is the number of timesteps taken.
14 %
```

```
%   Outputs:
%   None.
%
%   Kyle Lynch (lynchkp@auburn.edu) October 12, 2010

fid = fopen(filename,'r+');
% type = fgetl(fid);
n_pts = str2num(fgetl(fid));
% n_rays =  str2num(fgetl(fid));

for i = 1:n_pts
    tmp = str2num(fgetl(fid));
    xval(i) = tmp(1);
    yval(i) = tmp(2);
    zval(i) = tmp(3);
    ival(i) = tmp(4);
end
fclose(fid);

a = 5;
u0 = 1;

step_dt = dt / nsteps;
for t = 0:step_dt:dt
    X = xval;
    Y = zval;
    Z = yval;
    [TH,R,Z] = cart2pol(X,Y,Z);

    [u,v] = feval(func,Z,R,a,u0);
    [vx, vy ,vz] = pol2cart(TH,v,u);

    vx2 = vx;
    vy2 = vz+u0;
    vz2 = vy;

    xval = xval + vx2*step_dt;
    yval = yval + vy2*step_dt;
    zval = zval + vz2*step_dt;
end

% Write the file
fid2 = fopen(output_filename,'w+');
% fprintf(fid2,'%14s\n','particles');
fprintf(fid2,'%14d\n',n_pts);
% fprintf(fid2,'%14d\n',n_rays);

for i = 1:n_pts
    fprintf(fid2,'%14.6f %14.6f %14.6f %14.6f\n',xval(i),yval(i),zval(i),ival(i));
end

fclose(fid2);

end

function [u,v] = sim_hill_vortex(Z,R,a,u0)
%SIM_HILL_VORTEX Generate Particle Displacements
%   [U,V] = SIM_HILL_VORTEX(Z,R,A,U0)
%   Simulates velocity field of Hill's Spherical Vortex, from "Fluid
%   Vortices" by Sheldon Green. (1995)
%
%   Inputs:
%   Z is the coordinate lying in the direction of induced vortex velocity.
%   R is the coordinate perpendicular to the direction of induced velocity.
%   A is the size of the vortex.
%   U0 is the freestream velocity.
%   NSTEPS is the number of timesteps taken.
%
```

```
%   Outputs:
%   U is the x-component of velocity.
%   V is the y-component of velocity.
%
%   Kyle Lynch (lynchkp@auburn.edu) October 12, 2010

u_int = (3/2) .* u0 .* (1 - ((2.*R.^2 + Z.^2) ./ (a.^2)));
v_int = (3/2) .* u0 .* ((Z.*R)./(a.^2));

u_ext = u0 .* (((a.^2./(Z.^2 + R.^2)).^(5./2)).*((2.*Z.^2 - R.^2)./(2.*a.^2))-1);
v_ext = (3/2) .* u0 .* ((Z.*R)./(a.^2)) .* ((a.^2)./(Z.^2 + R.^2)).^(5./2);

test_int = (R.^2 + Z.^2)<=a.^2;
test_ext = (R.^2 + Z.^2)>a.^2;

u = u_int.*test_int + u_ext.*test_ext;
v = v_int.*test_int + v_ext.*test_ext;

end
```

Appendix C: Light Field Rendering Codes

```
function [cx, cy] = lf_calibration(filename)
%LF_CALIBRATION Light Field Image Coordinate Calibration
%   [CX, CY] = LF_CALIBRATION(FILENAME) registers
%   lenslet images and assigns and (i,j) coordinate system to each lenslet.
%   Additionally, edge lenslets are identified and discarded.
%
%   Outputs:
%   CX is an 2-D array specifying the j-coordinate of each lenslet.
%   CY is an 2-D array specifying the i-coordinate of each lenslet.
%
%   Kyle Lynch (lynchkp@auburn.edu) September 12, 2010

image = imread(filename);
image = double(image);

% Define grid. Must be done manually.
nlx = 288;
nly = 192;
sz = 16.973;
offset_x = 0;
offset_y = 2;

[xgrid ygrid] = meshgrid(1:sz:sz*nlx,1:sz:sz*nly);
xgrid = xgrid + offset_x;
ygrid = ygrid + offset_y;

% Calculate centroid locations.
tic
% figure
cx = zeros(nly-1,nlx-1);
cy = zeros(nly-1,nlx-1);
for yidx = 1:nly-1
    for xidx = 1:nlx-1
        xpt = round(xgrid(yidx,xidx));
        ypt = round(ygrid(yidx,xidx));
        subset = image(ypt:ypt+round(sz),xpt:xpt+round(sz));
        [p_fit] = gauss_2d_fit(subset);
        cx(yidx,xidx) = (xpt-1) + p_fit(2);
        cy(yidx,xidx) = (ypt-1) + p_fit(3);
    end
    fprintf('Peak-fit: Line %d of %d\n',yidx,nly-1);
end
toc

end
```

```
function [x_array, y_array, theta_array, phi_array, radiance] = ...
    lf_radiance_gen(filename,cx,cy)
%LF_RADIANCE_GEN Light Field Radiance Generation
%   [X_ARRAY,Y_ARRAY,THETA_ARRAY,PHI_ARRAY,RADIANCE] =
%       LF_RADIANCE_GEN(FILENAME,CX,CY) generates all information
%       regarding radiance elements, including their intensity, pixel coordinates,
%       spatial coordinates, and angular coordinates.
%
%   Inputs:
%   FILENAME is a string defining the lenslet image file.
%   CX is an 2-D array specifying the x-coordinate of each lenslet.
```

```
%   CY is an 2-D array specifying the y-coordinate of each lenslet.
%
%   Outputs:
%   X_ARRAY is a 4-D array specifying the x-coordinates of each radiance element.
%   Y_ARRAY is a 4-D array specifying the y-coordinates of each radiance element.
%   THETA_ARRAY is a 4-D array specifying the theta angle of each radiance element.
%   PHI_ARRAY is a 4-D array specifying the phi angle of each radiance element.
%   RADIANCE is a 4-D array specifying the intensity of each radiance element.
%
%   Kyle Lynch (lynchkp@auburn.edu) October 4, 2010

if nargin < 3
    fprintf('Not enough input arguments.\n');
    return;
end

% Determine number of lenslets.
[n_l_y, n_l_x] = size(cx);

% Read in image file.
image = imread(filename);
image = double(image);

% Miscellaneous initializations.
[n_p_y, n_p_x] = size(image);
p_p = 0.0074;
f_l = 0.500;
s_i = 100;
p_ccd_x = p_p*n_p_x;
p_ccd_y = p_p*n_p_y;

% Determine the average spacing between the centers of each lenslet. This
% allows us to determine the effective number of pixels underneath each
% lenslet. The floor function rounds this number down to the nearest
% integer value to ensure no overlap occurs. Additionally, a check is
% performed to ensure that the number of pixels used in both the x and y
% directions is equal.
avg_x = mean((cx(:,n_l_x)-cx(:,1)) / (n_l_x-1)); %%%% SHOULD THIS BE NLX-1?
avg_y = mean((cy(n_l_y,:)-cy(1,:)) / (n_l_y-1));
% keyboard;
if (floor(avg_x) ~= floor(avg_y))
    fprintf('Error: Unequal Lenslet Size'); return;
else
    n_p_l = floor(avg_x);
    r = n_p_l / 2;
end
n_p_l = 14;
r = 7;

% Initialize the data arrays. These are 4-D arrays, the size of the first
% two are equal to the number of lenslets in each direction, and the second
% two are equal to the number of pixels under each lenslet as defined
% above.
% n_array = zeros(n_l_y,n_l_x,n_p_l,n_p_l);
% m_array = zeros(n_l_y,n_l_x,n_p_l,n_p_l);
x_array = zeros(n_l_y, n_l_x, n_p_l, n_p_l);
y_array = zeros(n_l_y, n_l_x, n_p_l, n_p_l);
theta_array = zeros(n_l_y, n_l_x, n_p_l, n_p_l);
phi_array = zeros(n_l_y, n_l_x, n_p_l, n_p_l);
radiance = zeros(n_l_y, n_l_x, n_p_l, n_p_l);

% imshow(image,[]), hold on

% Loop through each lenslet, first in the horizontal (x-direction), and
% then in the vertical (y-direction).
for y = 1:n_l_y
    fprintf('Radiance Gen: %d of %d\n',y,n_l_y);
    for x = 1:n_l_x
```

```
81          % Grab the centroid location in (x,y) coordinates from the original
            % pixel coordinates. Note that because the y-coordinate is flipped
83          % in image coordinates, a corresponding flip has to occur here.
            l_x = cx(y,x)*p_p - p_ccd_x/2;
85          l_y = p_ccd_y/2 - cy(y,x)*p_p;

87          % Find the theta and phi angle for the lenslet centroid location.
            % This is known as the bulk angle and is calculated as the
89          % arctangent of the ratio of lenslet position divided by reference
            % image distance. This is also called the "bulk angle".
91          l_theta = atan2(l_x, s_i);
            l_phi = atan2(l_y, s_i);
93
            % With the lenslet information defined, the pixels underneath the
95          % lenslet must be identified. The pixels are found within a radius
            % defined previously. The pixel coordinates in the x-direction are
97          % stored as m_ind, and in the y-direction as n_ind.
            dist_x = abs((1:n_p_x) - cx(y,x));
99          dist_y = abs((1:n_p_y) - cy(y,x));
            m_ind = find(dist_x < r); % X
101         n_ind = find(dist_y < r); % Y

103         % Create an array of spatial coordinates equal to the size of the
            % block of pixels covered by the lenslet. These coordinates are
105         % homogenous within a single lenslet.
            x_c = ones(length(n_ind),length(m_ind))*l_x;
107         y_c = ones(length(n_ind),length(m_ind))*l_y;

109         % Determine the angles for each of the lenslet pixels. This is
            % known as the "local angle." First the internal spatial
111         % coordinates are defined for each pixel (effectively just an
            % offset from the lenslet centroid location), then the arctangent
113         % is used to calculate the angle.
            x2 = m_ind*p_p - p_ccd_x/2 - l_x;
115         y2 = p_ccd_y/2 - n_ind*p_p - l_y;
            [X2,Y2] = meshgrid(x2, y2);
117         p_theta = atan2(X2, f_l) + l_theta;
            p_phi = atan2(Y2, f_l) + l_phi;
119
            % Save all calibration information to the specified arrays.
121         x_array(y,x,:,:) = x_c;
            y_array(y,x,:,:) = y_c;
123         theta_array(y,x,:,:) = p_theta;
            phi_array(y,x,:,:) = p_phi;
125         radiance(y,x,:,:) = image(n_ind,m_ind);

127     end

129 end

131 end
```

```
1  function [image, scalefac] = lf_render(radiance,x_array,y_array,theta_array,phi_array,t)
   %LF_RENDER Light Field Rendering/Refocusing
3  %    [IMAGE] = LF_RENDER(RADIANCE,X_ARRAY,Y_ARRAY,THETA_ARRAY,PHI_ARRAY,T)
   %    integrates signal from each lenslet, rendering an image.
5  %
   %    Inputs:
7  %    RADIANCE is a 4-D array specifiying the intensity of each radiance element.
   %    X_ARRAY is a 4-D array specifying the x-coordinates of each radiance element.
9  %    Y_ARRAY is a 4-D array specifying the y-coordinates of each radiance element.
   %    THETA_ARRAY is a 4-D array specifying the theta-coordinates of each radiance element.
11 %    PHI_ARRAY is a 4-D array specifying the phi-coordinates of each radiance element.
   %    T is an optional argument specifying the distance to translate the
13 %    focus plane. By default, t = 0.
   %
15 %    Outputs:
```

```
%   IMAGE is a 2-D double-precision image.
%
%   Kyle Lynch (lynchkp@auburn.edu) October 4, 2010

if nargin < 5
    fprintf('Not enough input arguments.\n');
    return;
elseif nargin < 6
    t = 0;
end

% Set rendered image dimensions, which are equal to the number of lenslets
% in each of the coordinate directions.
[ny, nx, np, nt] = size(radiance);
image = zeros(ny, nx);

si = 100;

% Determine the effective sensor size, which is equal to the size of the
% lenslet array. Subtrating x_min from x_max will determine the distance
% between the outermost lenslet centroid locations. To find the total
% distance to the outsides of the lenslets, an additional lenslet diameter
% must be added.
scalefac = ((t+si)/si);
x_min = min(x_array(:));
y_min = min(y_array(:));
x_max = max(x_array(:));
y_max = max(y_array(:));
avg_x = mean( (x_max - x_min) / (nx-1) ) * scalefac;
avg_y = mean( (y_max - y_min) / (ny-1) ) * scalefac;
p_ccd_x = (x_max - x_min)*scalefac + avg_x;
p_ccd_y = (y_max - y_min)*scalefac + avg_y;

% Use simple 2-D ray tracing to propagate the radiance to a different
% position. This distance is denoted by the variable t.
if (t ~= 0)
    x_array = x_array + t*theta_array;
    y_array = y_array + t*phi_array;
end

% Accelerate the rendering by only calculating rays with nonzero intensity.
radiancetest = find(radiance>0);

% Render each ray.
for r = 1:length(radiancetest)

    % Grab all information about the particular ray from the information
    % arrays.
    ray = radiancetest(r); % Ray index.
    ray_radiance = radiance(ray); % Ray intensity.
    ray_x = x_array(ray); % Ray x-coordinate
    ray_y = y_array(ray); % Ray y-coordinate

    % Determine the location the ray intersects the focal plane in terms of
    % the lenslet number (floating-point number).
    initial_x = (p_ccd_x/2 + ray_x + avg_x/2) / (avg_x);
    initial_y = (p_ccd_y/2 - ray_y + avg_y/2) / (avg_y);

    % Determine the four surrounding pixels.
    L_x = floor(initial_x);
    R_x = ceil(initial_x);
    L_x_coeff = R_x - initial_x;
    R_x_coeff = initial_x - L_x;

    B_y = floor(initial_y);
    T_y = ceil(initial_y);
    B_y_coeff = T_y - initial_y;
    T_y_coeff = initial_y - B_y;
```

```
85      % Determine the contribution of the ray to each pixel.
        TL = T_y_coeff*L_x_coeff;
87      TR = T_y_coeff*R_x_coeff;
        BL = B_y_coeff*L_x_coeff;
89      BR = B_y_coeff*R_x_coeff;

91      % Render the final image through a summation of the image array. The
        % multiple if statements are used to block rays that are propogated
93      % outside of the image coordinates.
        if T_y > 0 && T_y <= ny
95          if L_x > 0 && L_x <= nx
                % Top Left Pixel
97              image(T_y,L_x) = image(T_y,L_x) + ray_radiance*TL;
            end
99          if R_x > 0 && R_x <= nx
                % Top Right Pixel
101             image(T_y,R_x) = image(T_y,R_x) + ray_radiance*TR;
            end
103     end

105     if B_y > 0 && B_y <= ny
            if L_x > 0 && L_x <= nx
107             % Bottom Left Pixel
                image(B_y,L_x) = image(B_y,L_x) + ray_radiance*BL;
109         end
            if R_x > 0 && R_x <= nx
111             % Bottom Right Pixel
                image(B_y,R_x) = image(B_y,R_x) + ray_radiance*BR;
113         end
        end
115
    end
```

Appendix D: Matlab 3-D PIV Codes

```
clear all
clc

% Specify correlation settings and input files.
scalefactor = 4; % Vector display scale factor.
zero_padding = 'false'; % Zero-padded correlation.
spof = 'false'; % Symmetric phase-only filtering.
nmt_eps = 0.1; % Error value for normalized median test (usually 0.1-0.2).
nmt_threshlvl = 2; % Threshold value for normalized median test (usually 2).
start_x_spacing = 32; % Spacing between interrogation windows in x-direction.
start_y_spacing = 32; % Spacing between interrogation windows in y-direction.
start_z_spacing = 4; % Spacing between interrogation windows in y-direction.
start_x_size = 64; % Size of interrogation windows in x-direction.
start_y_size = 64; % Size of interrogation windows in y-direction.
start_z_size = 8; % Size of interrogation windows in y-direction.
refinement_factor = 3; % Refinement factor (size_initial/size_final)
file_prefix = 'C:\Users\lynchkp\Desktop\Light_Field_Rendering_Codes\001x\'; % Base file directory.
file_A_prefix = 'imgA';
file_B_prefix = 'imgB';

% Load files. All images converted to double-precision format.
nfiles = 16;
idx = 1;
for i = 1:1:nfiles
    imgA(:,:,idx) = double( imread([file_prefix file_A_prefix num2str(i,'%03d') '.tif']) );
    imgB(:,:,idx) = double( imread([file_prefix file_B_prefix num2str(i,'%03d') '.tif']) );
    idx = idx + 1;
end
[image_height, image_width, image_depth] = size(imgA);

h = figure;
imshow(imgA(:,:,4),[])
set(h,'Color','none');


% export_fig('C:\Users\lynchkp\Desktop\Masters Thesis\figures\001_plane4.pdf',h);
break;

for R = 1:refinement_factor

    % Define the window parameters for the particular grid refinement
    % level. Note the grid refinement is exponential because the refinement
    % factor is in the denominator.
    x_spacing = round(start_x_spacing / R);
    y_spacing = round(start_y_spacing / R);
    z_spacing = round(start_z_spacing / R);
    x_size = round(start_x_size / R);
    y_size = round(start_y_size / R);
    z_size = round(start_z_size / R);

    % Ensure both dimensions of each window are even, due to the
    % performance of the fftshift function.
    if mod(x_size,2) == 1
        x_size = x_size + 1;
    end
    if mod(y_size,2) == 1
        y_size = y_size + 1;
```

```
59      end
        if mod(z_size,2) == 1
61          z_size = z_size + 1;
        end
63
        % Generate the grid. For the initial grid, no information needs to be
65      % saved. For the remaining grids, old grid data must be stored to build
        % the predictor.
67      if R > 1
            old_cent_x = cent_x; old_cent_y = cent_y; old_cent_z = cent_z;
69          old_nx = nx; old_ny = ny; old_nz = nz;
            old_dx = dx; old_dy = dy; old_dz = dz;
71      end
        [node_x, node_y, node_z, cent_x, cent_y, cent_z, nx, ny, nz] = ...
73          generate_grid_3d(image_height, image_width, image_depth, ...
            x_size, y_size, z_size, x_spacing, y_spacing, z_spacing);
75
        % Initialize the predictor. For the initial grid, no correlation exists
77      % and thus it can be set to zero for all windows. However, for the
        % remaining grids, there exists correlation data to use for
79      % initializing a predictor.
        if R > 1
81          [px, py, pz] = ...
                generate_predictor_discrete_3d(old_cent_x, old_cent_y, old_cent_z, old_dx, old_dy, old_dz, ...
83              old_nx, old_ny, old_nz, cent_x, cent_y, cent_z, nx, ny, nz);
        else
85          px = zeros(size(node_x));
            py = zeros(size(node_y));
87          pz = zeros(size(node_z));
        end
89
        % Multi-pass analysis. The number of passes are dictated by a
91      % convergence criteria, such that the RMS values of the corrector
        % differ by less than 0.1 pixels, and less than 10 passes are
93      % performed.
        old_rms = 0; new_rms = 1E6; pass = 1;
95      while (abs(new_rms - old_rms) > 0.1) && (pass < 10)

97          [cx, cy, cz, R_vec, IA_vec, IB_vec] = piv_pass_discrete_3d(imgA, imgB, node_x, node_y, node_z, ...
                x_size, y_size, z_size, px, py, pz, zero_padding, spof);
99
            %%% Update Displacement %%%
101         dx = px + cx;
            dy = py + cy;
103         dz = pz + cz;

105         %%% Validate Displacement %%%
            old_total = 0; new_total = 1; pass2 = 1;
107         while (abs(new_total - old_total) > 0.1) && (pass2 < 10)
                [test] = normalized_median_test_3d(dx, dy, dz, nx, ny, nz, nmt_threshlvl, nmt_eps);
109             [dx, dy, dz] = bilinear_interpolation_3d(dx, dy, dz, nx, ny, nz, test);
                old_total = new_total;
111             new_total = ((numel(test)-numel(find(test==0)))/numel(test))*100;
                fprintf('Validation Percentage: %.2f\n',new_total);
113             pass2 = pass2 + 1;
            end
115
            %%% Update Predictor and Convergence Check %%%
117         pass = pass + 1;
            px = round(dx);
119         py = round(dy);
            pz = round(dz);
121         old_rms = new_rms;
            new_rms = sqrt(mean(cx.^2 + cy.^2));
123         fprintf('Corrector RMS: %.4f\n',new_rms);

125     end
```

```
end

load scalevec2
dx = dx .* median(scalevec2);
dy = dy .* median(scalevec2);

h = figure('Position',[100 100 600 400]);
scalefac = 40;
skip = 5;
quiver3(cent_x(1:skip:end),cent_y(1:skip:end),cent_z(1:skip:end), ...
    dx(1:skip:end)*scalefac - mean(dx(1:skip:end)*scalefac),dy(1:skip:end)*scalefac*0,dz(1:skip:end)*scalef
view([138 70])
xlabel('X (px)');
ylabel('Y (px)');
zlabel('Z (px)');
box off
set(h,'Color','none');
export_fig('C:\Users\lynchkp\Desktop\Masters Thesis\figures\001x_vectors_non.pdf',h);

h = figure('Position',[100 100 600 300]);
[n,x] = hist(dx,50);
hist(dx,50)

h2 = findobj(gca,'Type','patch');
set(h2,'FaceColor',[.6 .6 .6],'EdgeColor',[.3 .3 .3])

hold on
[p] = gauss_1d_fit(n,x)
ztmp = p(1) * exp( -(0.5*(x-p(2)).^2)/p(3)^2 );
r = plot(x,ztmp,'r-','LineWidth',2)
set(r,'LineWidth',2)
xlabel('Displacement (px)');
ylabel('Counts');

annotation('textbox',[.18 .68 .18 .13],...
    'FontSize',10,'BackgroundColor','none','EdgeColor','none',...
    'String',['\sigma = ' num2str(std(dx),'%.2f') ' px']);
annotation('textbox',[.18 .74 .18 .13],...
    'FontSize',10,'BackgroundColor','none','EdgeColor','none',...
    'String',['\mu = ' num2str(mean(dx),'%.2f') ' px']);

set(h,'Color','none');
export_fig('C:\Users\lynchkp\Desktop\Masters Thesis\figures\001x_vectors_histx_non.pdf',h);
```

```
function [cx, cy, cz, R_vec, IA_vec, IB_vec] = piv_pass_discrete_3d(imgA, imgB, node_x, node_y, node_z, ...
    x_size, y_size, z_size, px, py, pz, zero_padding, spof)
%PIV_PASS_DISCRETE PIV interrogation with discrete window offset.
%   [] = PIV_PASS_DISCRETE(IMGA, IMGB, NODE_X, NODE_Y, NODE_Z,
%       X_SIZE, Y_SIZE, Z_SIZE, PX, PY, PZ, ZERO_PADDING, SPOF)
%   FFT-based discrete spatial correlation. Based on the implementation
%   suggested in Adrian and Westerweel (2011), pp. 369-376.
%
%   Inputs:
%   IMGA is the full data of the first image.
%   IMGA is the full data of the second image.
%   NODE_X are the upper-left node points of each window (x-coordinate).
%   NODE_Y are the upper-left node points of each window (y-coordinate).
%   NODE_Z are the upper-left node points of each window (z-coordinate).
%   X_SIZE is the interrogation window size in pixels (x-coordinate).
%   Y_SIZE is the interrogation window size in pixels (y-coordinate).
%   Z_SIZE is the interrogation window size in pixels (z-coordinate).
%   PX is the discrete integer predictor of x-coordinate displacement.
%   PY is the discrete integer predictor of y-coordinate displacement.
%   PZ is the discrete integer predictor of z-coordinate displacement.
%   ZERO_PADDING is a boolean for using zero-padded interrogation windows.
%   SPOF is a boolean for using symmetric phase-only filtering.
%
%   Outputs:
```

```
%   CX is the fractional corrector of x-coordinate displacement.
%   CY is the fractional corrector of y-coordinate displacement.
%   CZ is the fractional corrector of y-coordinate displacement.
%   R_VEC is a vector containing each correlation volume for analysis.
%
%   Kyle Lynch (lynchkp@auburn.edu) February 27, 2011

tic

[image_height, image_width, image_depth] = size(imgA);

% Set parameters that depend on correlation plane sizing.
if strcmp(zero_padding,'true')
    U = y_size*2;
    V = x_size*2;
    W = z_size*2;
else
    U = y_size;
    V = x_size;
    W = z_size;
end

% Initialize storage arrays.
R_vec = zeros(U, V, W, numel(node_x));
IA_vec = zeros(y_size, x_size, z_size, numel(node_x));
IB_vec = zeros(y_size, x_size, z_size, numel(node_x));

IA = zeros(y_size, x_size, z_size);
IB = zeros(y_size, x_size, z_size);
cx = zeros(1, numel(node_x));
cy = zeros(1, numel(node_y));
cz = zeros(1, numel(node_y));

for idx = 1:numel(node_x)

    % Generate windows.
    for z = 1:z_size
        for y = 1:y_size
            for x = 1:x_size

                yi = y + node_y(idx);
                xi = x + node_x(idx);
                zi = z + node_z(idx);
                yip = y + node_y(idx) + py(idx);
                xip = x + node_x(idx) + px(idx);
                zip = z + node_z(idx) + pz(idx);

                if ((xi <= 0) || (xi > image_width) || (yi <= 0) || (yi > image_height) || (zi <= 0) || (z
                    IA(y,x,z) = 0;
                else
                    IA(y,x,z) = imgA(yi,xi,zi);
                end

                if ((xip <= 0) || (xip > image_width) || (yip <= 0) || (yip > image_height) || (zip <= 0)
                    IB(y,x,z) = 0;
                else
                    IB(y,x,z) = imgB(yip,xip,zip);
                end

            end
        end
    end

    % Calculate standard deviation of windows.
    IA_std = std(IA(:));
    IB_std = std(IB(:));

    % Calculate mean of windows.
```

```
93      IA_mean = mean(IA(:));
        IB_mean = mean(IB(:));
95
        % Subtract mean from both windows.
97      IA_s = IA - IA_mean;
        IB_s = IB - IB_mean;
99
        % Perform FFT.
101     FA = fftn(IA_s, [U, V, W]);
        FB = fftn(IB_s, [U, V, W]);
103
        % Multiply by complex conjugate.
105     S = conj(FA) .* FB;

107     % Take inverse FFT and rescale the data.
        R  = ifftn(S,'symmetric') ./ (U*V*W*IA_std*IB_std);
109
        % Reshuffle FFT plane.
111     R = fftshift(R);

113     [x0, y0, z0] = gaussian_estimator_3d(R);

115     % Corrector Calculation.
        cx(idx) = x0 - V/2 -1;
117     cy(idx) = y0 - U/2 -1;
        cz(idx) = z0 - W/2 -1;
119

121     if (max(IA(:)) == 0) || (max(IB(:)) == 0)
            cx(idx) = 0;
123         cy(idx) = 0;
            cz(idx) = 0;
125     end

127     R_vec(:,:,:,idx) = R;
        IA_vec(:,:,:,idx) = IA;
129     IB_vec(:,:,:,idx) = IB;

131 end


133


135 end
```

```
1  function [test] = ...
       normalized_median_test_3d(dx, dy, dz, nx, ny, nz, nmt_threshlvl, nmt_eps)
3
   tic
5
   radius = 1;
7
   % Convert 1-D field to 3-D field to make indexing and neighboring data
9  % easier to process.
   idx = 1;
11 dx2 = zeros(ny,nx,nz);
   dy2 = zeros(ny,nx,nz);
13 dz2 = zeros(ny,nx,nz);
   for z = 1:nz
15     for y = 1:ny
           for x = 1:nx
17             dx2(y,x,z) = dx(idx);
               dy2(y,x,z) = dy(idx);
19             dz2(y,x,z) = dz(idx);
               idx = idx + 1;
21         end
       end
23 end
```

```
test2 = ones(ny,nx,nz);
for z = 1:1:nz
    for y = 1:1:ny
        for x = 1:1:nx

            % Extract data neighborhood excluding the center point.
            neigh_x = []; neigh_y = []; neigh_z = []; idx = 1;
            for k = -radius:1:radius
                for j = -radius:1:radius
                    for i = -radius:1:radius
                        xi = x + i;
                        yj = y + j;
                        zk = z + k;
                        if ((xi <= nx) && (xi > 0) && ...
                                (yj <= ny) && (yj > 0) && ...
                                (zk <= nz) && (zk > 0) && ...
                                ((j ~= 0) || (i ~= 0) || (k ~= 0)))
                            neigh_x(idx) = dx2(yj,xi,zk);
                            neigh_y(idx) = dy2(yj,xi,zk);
                            neigh_z(idx) = dz2(yj,xi,zk);
                            idx = idx + 1;
                        end
                    end
                end
            end

            % Calculate median of neighboring values
            neighmed_x = median(neigh_x);
            neighmed_y = median(neigh_y);
            neighmed_z = median(neigh_z);

            % Fluctuations with respect to the median
            medfluct_x = dx2(y,x,z) - neighmed_x;
            medfluct_y = dy2(y,x,z) - neighmed_y;
            medfluct_z = dz2(y,x,z) - neighmed_z;

            % Absolute value of residuals within the neighborhood
            neighmedfluct_x = neigh_x - neighmed_x;
            neighmedfluct_y = neigh_y - neighmed_y;
            neighmedfluct_z = neigh_z - neighmed_z;

            % Median of absolute residuals
            medneighmedfluct_x = median(abs(neighmedfluct_x));
            medneighmedfluct_y = median(abs(neighmedfluct_y));
            medneighmedfluct_z = median(abs(neighmedfluct_z));

            % Normalized fluctuations
            normfluct_x = abs(medfluct_x / (medneighmedfluct_x + nmt_eps));
            normfluct_y = abs(medfluct_y / (medneighmedfluct_y + nmt_eps));
            normfluct_z = abs(medfluct_z / (medneighmedfluct_z + nmt_eps));

            if (sqrt(normfluct_x^2 + normfluct_y^2 + normfluct_z^2) > nmt_threshlvl)
                test2(y,x,z) = 0;
            end

        end
    end
end

% Convert 3-D test to 1-D test.
idx = 1;
test = zeros(1,ny*nx*nz);
for z = 1:nz
    for y = 1:ny
        for x = 1:nx
            test(idx) = test2(y,x,z);
            idx = idx + 1;
        end
```

```
93      end
    end
95
    % Calculate percentage of erroneous vectors
97  % percent = (count / (nx*ny*nz)) * 100;
    % percent = ( ( numel(dx)-numel(find(test==0)) ) / numel(dx) ) * 100;
99
    % fprintf('Normalized Median Test complete, elapsed time %.2f seconds.\n',toc);
101
    end

    function [dx_out, dy_out, dz_out] = bilinear_interpolation_3d(dx, dy, dz, nx, ny, nz, test)
2
    tic
4
    % Convert 1-D field to 3-D field to make indexing and neighboring data
6   % easier to process.
    idx = 1;
8   dx2 = zeros(ny,nx,nz);
    dy2 = zeros(ny,nx,nz);
10  dz2 = zeros(ny,nx,nz);
    test2 = zeros(ny,nx,nz);
12  for z = 1:nz
        for y = 1:ny
14          for x = 1:nx
                dx2(y,x,z) = dx(idx);
16              dy2(y,x,z) = dy(idx);
                dz2(y,x,z) = dz(idx);
18              test2(y,x,z) = test(idx);
                idx = idx + 1;
20          end
        end
22  end

24  dx_out2 = dx2;
    dy_out2 = dy2;
26  dz_out2 = dz2;
    for z = 1:1:nz
28      for y = 1:1:ny
            for x = 1:1:nx
30              if test2(y,x,z) == 0

32                  % X-Component of Bilinear Interpolation
                    idx = 1;
34                  neigh_x = [];
                    for i = [-1, 1]
36                      xi = x + i;
                        yj = y;
38                      zk = z;
                        if ( (xi <= nx) && (xi > 0) && (test2(yj,xi,zk)==1) )
40                          neigh_x(idx) = dx2(yj,xi,zk);
                            idx = idx + 1;
42                      end
                    end
44                  if numel(neigh_x) >= 1
                        dx_out2(y,x,z) = sum(neigh_x) / numel(neigh_x);
46                  end

48                  % Y-Component of Bilinear Interpolation
                    idx = 1;
50                  neigh_y = [];
                    for i = [-1, 1]
52                      xi = x;
                        yj = y + i;
54                      zk = z;
                        if ( (yj <= ny) && (yj > 0) && (test2(yj,xi,zk)==1) )
56                          neigh_y(idx) = dy2(yj,xi,zk);
                            idx = idx + 1;
```

```
                    end
                end
                if numel(neigh_y) >= 1
                    dy_out2(y,x,z) = sum(neigh_y) / numel(neigh_y);
                end

                % Z-Component of Bilinear Interpolation
                idx = 1;
                neigh_z = [];
                for i = [-1, 1]
                    xi = x;
                    yj = y;
                    zk = z + i;
                    if ( (zk <= nz) && (zk > 0) && (test2(yj,xi,zk)==1) )
                        neigh_z(idx) = dz2(yj,xi,zk);
                        idx = idx + 1;
                    end
                end
                if numel(neigh_z) >= 1
                    dz_out2(y,x,z) = sum(neigh_z) / numel(neigh_z);
                end


            end
        end
    end
end

% Convert 2-D field to 1-D array.
idx = 1;
for z = 1:nz
    for y = 1:ny
        for x = 1:nx
            dx_out(idx) = dx_out2(y,x,z);
            dy_out(idx) = dy_out2(y,x,z);
            dz_out(idx) = dz_out2(y,x,z);
            idx = idx + 1;
        end
    end
end

% fprintf('Bilinear vector replacement complete, elapsed time %f seconds.\n',toc);

end
```

```
function [node_x, node_y, node_z, cent_x, cent_y, cent_z, nx, ny, nz] = ...
    generate_grid_3d(image_height, image_width, image_depth, ...
    x_size, y_size, z_size, x_spacing, y_spacing, z_spacing)
%GENERATE_GRID_3D Generate 3D PIV Grid
%   [NODE_X, NODE_Y, NODE_Z, CENT_X, CENT_Y, CENT_Z, NX, NY, NZ] =
%       GENERATE_GRID(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_DEPTH,
%           X_SIZE, Y_SIZE, Z_SIZE, X_SPACING, Y_SPACING, Z_SPACING)
%   Creates a grid for PIV evaluation, based on the dimensions and
%   separations of interrogation windows.
%
%   Inputs:
%   IMAGE_HEIGHT is the height of the image (y-coordinate).
%   IMAGE_WIDTH is the width of the image (x-coordinate).
%   IMAGE_DEPTH is the width of the image (z-coordinate).
%   X_SIZE is the interrogation window size in pixels (x-coordinate).
%   Y_SIZE is the interrogation window size in pixels (y-coordinate).
%   Z_SIZE is the interrogation window size in pixels (z-coordinate).
%   X_SPACING is the interrogation window spacing in pixels (x-coordinate).
%   Y_SPACING is the interrogation window spacing in pixels (y-coordinate).
%   Z_SPACING is the interrogation window spacing in pixels (z-coordinate).
%
%   Outputs:
%   NODE_X are the upper-left node points of each window (x-coordinate).
```

```
%   NODE_Y are the upper-left node points of each window (y-coordinate).
%   NODE_Z are the upper-left node points of each window (z-coordinate).
%   CENT_X are the centroid points of each window (x-coordinates).
%   CENT_Y are the centroid points of each window (y-coordinates).
%   CENT_Z are the centroid points of each window (z-coordinates).
%   NX are the number of windows in the x-direction.
%   NY are the number of windows in the y-direction.
%   NZ are the number of windows in the z-direction.
%
%   Kyle Lynch (lynchkp@auburn.edu) March 3, 2011

% Determine the grid offset to center the grid within the image.
nz = 0;
for z = 1:z_spacing:image_depth-z_size
    ny = 0;
    for y = 1:y_spacing:image_height-y_size
        nx = 0;
        for x = 1:x_spacing:image_width-x_size
            nx = nx + 1;
        end
        ny = ny + 1;
    end
    nz = nz + 1;
end
x_offset = round((image_width-x_size-x) / 2);
y_offset = round((image_height-y_size-y) / 2);
z_offset = round((image_depth-z_size-z) / 2);
fprintf('Grid Generation...\n');
fprintf('X Size:    %3d, Y Size:    %3d, Z Size:    %3d\n',x_size,y_size,z_size);
fprintf('X Spacing: %3d, Y Spacing: %3d, Z Spacing: %3d\n',x_spacing,y_spacing,z_spacing);
fprintf('X Regions: %3d, Y Regions: %3d, Z Regions: %3d\n',nx,ny,nz);
fprintf('X Offset:  %3d, Y Offset:  %3d, Z Offset:  %3d\n',x_offset,y_offset,z_offset);
fprintf('Number of Regions: %6d\n',nx*ny*nz);

% Build grid nodal and centroid points and store them in row-major format.
idx = 1;
node_x = zeros(1,nx*ny*nz); cent_x = zeros(1,nx*ny*nz);
node_y = zeros(1,nx*ny*nz); cent_y = zeros(1,nx*ny*nz);
node_z = zeros(1,nx*ny*nz); cent_z = zeros(1,nx*ny*nz);
for z = 1:1:nz
    for y = 1:1:ny
        for x = 1:1:nx
            node_x(idx) = x_offset + (x-1)*x_spacing;
            cent_x(idx) = x_offset + (x-1)*x_spacing + x_size/2;

            node_y(idx) = y_offset + (y-1)*y_spacing;
            cent_y(idx) = y_offset + (y-1)*y_spacing + y_size/2;

            node_z(idx) = z_offset + (z-1)*z_spacing;
            cent_z(idx) = z_offset + (z-1)*z_spacing + z_size/2;

            idx = idx + 1;
        end
    end
end

end


function [px, py, pz] = ...
    generate_predictor_discrete_3d(old_cent_x, old_cent_y, old_cent_z, old_dx, old_dy, old_dz, ...
    old_nx, old_ny, old_nz, new_cent_x, new_cent_y, new_cent_z, new_nx, new_ny, new_nz)
%GENERATE_PREDICTOR_DISCRETE Generate discrete integer predictor
%   [] = GENERATE_PREDICTOR_DISCRETE(OLD_CENT_X, OLD_CENT_Y, OLD_DX, OLD_DY,
%       OLD_NX, OLD_NY, NEW_CENT_X, NEW_CENT_Y, NEW_NX, NEW_NY)
%   Creates the predictor array for PIV interrogation, based on a linear
%   interpolation to new grid coordinates.
%
%   Inputs:
```

```
%   OLD_CENT_X is an array of old window centroid locations (x-coordinate).
%   OLD_CENT_Y is an array of old window centroid locations (y-coordinate).
%   OLD_DX is the array of old displacements (x-coordinate).
%   OLD_DY is the array of old displacements (y-coordinate).
%   OLD_NX is the number of old interrogation windows in the x-direction.
%   OLD_NY is the number of old interrogation windows in the y-direction.
%   NEW_CENT_X is an array of new window centroid locations (x-coordinate).
%   NEW_CENT_Y is an array of new window centroid locations (y-coordinate).
%   NEW_NX is the number of new interrogation windows in the x-direction.
%   NEW_NY is the number of new interrogation windows in the y-direction.
%
%   Outputs:
%   PX is the discrete integer predictor for the x-coordinate.
%   PY is the discrete integer predictor for the y-coordinate.
%
%   Kyle Lynch (lynchkp@auburn.edu) February 27, 2011

% Convert 1-D field to 3-D field to make indexing and neighboring data
% easier to process.
idx = 1;
dx2 = zeros(old_ny,old_nx,old_nz);
dy2 = zeros(old_ny,old_nx,old_nz);
dz2 = zeros(old_ny,old_nx,old_nz);
for z = 1:old_nz
    for y = 1:old_ny
        for x = 1:old_nx
            old_cent_x2(y,x,z) = old_cent_x(idx);
            old_cent_y2(y,x,z) = old_cent_y(idx);
            old_cent_z2(y,x,z) = old_cent_z(idx);
            old_dx2(y,x,z) = old_dx(idx);
            old_dy2(y,x,z) = old_dy(idx);
            old_dz2(y,x,z) = old_dz(idx);
            idx = idx + 1;
        end
    end
end

idx = 1;
dx2 = zeros(old_ny,old_nx,old_nz);
dy2 = zeros(old_ny,old_nx,old_nz);
dz2 = zeros(old_ny,old_nx,old_nz);
for z = 1:new_nz
    for y = 1:new_ny
        for x = 1:new_nx
            cent_x2(y,x,z) = new_cent_x(idx);
            cent_y2(y,x,z) = new_cent_y(idx);
            cent_z2(y,x,z) = new_cent_z(idx);
            idx = idx + 1;
        end
    end
end

% Perform interpolation.
px2 = round(interp3(old_cent_x2, old_cent_y2, old_cent_z2, old_dx2, cent_x2, cent_y2, cent_z2, 'linear', 0
py2 = round(interp3(old_cent_x2, old_cent_y2, old_cent_z2, old_dy2, cent_x2, cent_y2, cent_z2, 'linear', 0
pz2 = round(interp3(old_cent_x2, old_cent_y2, old_cent_z2, old_dz2, cent_x2, cent_y2, cent_z2, 'linear', 0

% Convert 2-D arrays back to 1-D arrays.
idx = 1;
for z = 1:new_nz
    for y = 1:new_ny
        for x = 1:new_nx
            px(idx) = px2(y,x,z);
            py(idx) = py2(y,x,z);
            pz(idx) = pz2(y,x,z);
            idx = idx + 1;
        end
    end
```

```
    end
80
    end
```